Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Pedro Nuno Lourenço Ferreira

Mecanismos de controlo de acesso para redes 6LoWPAN

Relatório de Projeto
Trabalho Final de Mestrado

Orientado por:

Professor Doutor Luís Miguel Lopes de Oliveira

Projeto de mestrado apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em
Engenharia Informática – Internet das Coisas

Resumo

A IoT tem suscitado o interesse generalizado da comunidade académica e da indústria, motivado pelos potenciais ganhos que podem surgir com a incorporação deste tipo de tecnologia. Apesar da IoT não estar limitada a um tipo específico de tecnologias é consensual que grande parte dos dispositivos apresentam restrições relativamente à capacidade de retenção de energia, capacidade de armazenamento e de processamento. Por esta razão, é comum o uso do protocolo IEEE 802.15.4 para assegurar a conectividade na segunda camada e o 6LoWPAN em conjunto com o protocolo IPv6 como camada de adaptação. Apesar do uso da pilha protocolar IP em todos os dispositivos e da quantidade de soluções propostas, há ainda problemas por resolver no que diz respeito à integração contínua e sem interrupções das soluções baseadas em IoT na Internet e de criar as condições para a adoção generalizada deste tipo de tecnologias. Uma grande parte dos problemas está relacionada com os aspetos de segurança. Suportar mecanismos de segurança quando estão envolvidos nós com baixos recursos de computação e retenção de energia e quando grande parte das comunicações são entre nós onde os utilizadores não têm intervenção é um grande desafio. Os mecanismos de criptografia estão normalmente envolvidos na conceção de soluções de segurança, razão pela qual a gestão de chaves criptográficas é um aspeto crítico. Controlar quais os nós que podem aceder à rede torna a rede mais segura e simultaneamente mais fácil de gerir. No âmbito deste trabalho é proposto um mecanismo de controlo de acessos à rede e que simultaneamente agiliza a gestão das chaves criptográficas simétricas em ambientes multi-hop. O mecanismo proposto recorre ao uso de Diffie Hellman para o acordo de chaves e à critpografia de curvas elípticas. Para validar a solução proposta foi construída uma testbed laboratorial composta por equipamentos de baixo custo e com recurso a software aberto e de utilização livre.

Palavras Chave (Tema): Redes de Sensores, IoT, Segurança em Redes de Sensores

Palavras Chave (Tecnologias): RPL, IEEE 802.15.4, 6LoWPAN, DTL

Abstract

IoT has attracted widespread interest from the academic community and industry, motivated by the potential gains that can arise with the incorporation of this type of technology. Although IoT is not limited to a specific type of technology, it is agreed that most of the devices have restrictions on the energy retention capacity, storage and processing capacity. For this reason, it is common to use the IEEE 802.15.4 protocol to ensure connectivity in the second layer and 6LoWPAN in conjunction with the IPv6 protocol as the adaptation layer. Despite the use of the IP protocol stack in all devices and the number of proposed solutions, there are still unresolved issues regarding continuous and uninterrupted integration of IoT-based Internet solutions and to create the conditions for the widespread adoption of this type of technology. A large part of the problems are related to the security aspects. It is a great challenge to support security mechanisms when nodes with low power and computing resources are involved and when most communications are between nodes where users do not have intervention. Encryption mechanisms are usually involved in the design of security solutions, which is why cryptographic key management is a critical aspect. Controlling which nodes can access the network makes the network more secure and simultaneously easier to manage. In this work, a network access control mechanism is proposed that simultaneously expedites the management of symmetric cryptographic keys in multi-hop environments. A laboratory testbed consisting of low-cost equipment and using open and free software was built to validate the proposed solution.

Keywords Wireless Sensor Networks, IoT, Security in Wireless Sensor Networks, IEEE 802.15.4, 6LoWPAN, DTLS

.

Agradecimentos

Gostaría de agradecer, em primeiro lugar, ao Professor Luís Miguel Lopes de Oliveira da Escola Superior de Tecnologia de Tomar do Instituto Politécnico de Tomar, na qualidade de orientador, pelo apoio, disponibilidade e empenho demonstrado.

Um agradecimento ao Pedro Dias e em especial ao meu irmão João Ferreira, pela ajuda e apoio que foram de extrema importância na concretização deste trabalho.

Aos colegas e amigos, André Farinha, Ricardo Anacleto e Nelson Gomes por toda a ajuda e suporte, não apenas em termos pessoais mas também ao longo do meu trajeto escolar.

Ao Instituto Politécnico de Tomar, em especial ao projecto VitaSenior que proporcionou excelentes condições de trabalho, tanto equipamentos como espaços.

Por fim, gostaría de agradecer à minha família, amigos e a todas as pessoas que são especialmente chegadas, por todo o apoio prestado, que de uma forma ou de outra, facilitaram o caminho desta minha etapa.

Este trabalho está integrado e foi suportado financeiramente pelo projeto IC&DT VITASENIOR-MT CENTRO-01-0145-FEDER-023659 com fundos do FEDER através dos programas operacionais CENTRO2020 e FCT.

Índice

Índice de Figuras

Notação e Glossário

AES	**A**dvanced **E**ncryption **S**tandard
API	**A**pplication **P**rogramming **I**nterface
BR	**B**order **R**outer
COAP	**C**onstrained **A**pplication **P**rotocol
COAPS	**C**onstrained **A**pplication **P**rotocol Secure
DAG	**D**irected **A**cyclic **G**raph
DH	**D**iffie-**H**ellman
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DODAG	**D**estination **O**riented **D**irected **A**cyclic **G**raph
DoS	**D**enial of **S**ervice
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity
ECC	**E**lliptic-**C**urve **C**ryptography
ECDHE	**E**lliptic-**C**urve **D**iffie–**H**ellman
ECDSA	**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
EUI	**E**xtended **U**nique **I**dentifier
FFD	**F**ull **F**unction **D**evice
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol
KDC	**K**ey **D**istribution **C**enter
LoWPAN	**L**ow-**P**ower **W**ireless **P**ersonal **A**rea **N**etworks
LWM2M	**L**ightweight **M**achine to **M**achine
MAC	**M**essage **A**uthentication **C**ode

M2M	**M**achine to **M**achine
NA	**N**eighbour **A**dvertising
ND	**N**eighbour **D**iscovery
NS	**N**eighbour **S**olicitation
PSK	**P**re-**S**hared **K**ey
RA	**R**outer **A**dvertising
RFD	**R**educed **F**unction **D**evice
RPK	**R**aw **P**ublic **K**ey
RPL	**R**outing **P**rotocol for **L**ow-Power and Lossy Networks
RS	**R**outer **A**dvertising
RSSF	**R**edes de **S**ensores **S**em **F**ios
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
UDP	**U**ser **D**atagram **P**rotocol

1 Introdução

1.1 Motivação e enquadramento

A evolução dos últimos anos na microeletrónica e dos sistemas de comunicações tornou possível a construção de *hardware* de baixo custo que pode ser embebido em objetos que usamos no nosso dia-a-dia, sendo por isso possível ligá-los à Internet de forma a interagir e a controlar o meio ambiente envolvente. Este tipo de soluções é designado por Internet of Things (IoT) e devido à sua aplicabilidade tem suscitado o interesse tanto do meio académico como da indústria. O controlo de processos industriais, a monitorização ambiental, o apoio à gestão do sistema de iluminação de uma cidade ou aplicações na área da saúde são algumas das áreas onde as soluções baseadas em IoT podem ser aplicadas. Ao contrário do que acontece atualmente na Internet, em que a maioria das comunicações têm um humano num dos extremos da comunicação, no IoT as comunicações são maioritariamente entre máquinas (M2M).

As redes de sensores são um dos pilares da IoT e têm características muito específicas, que estão essencialmente relacionadas com as restrições de recursos de *hardware* e de retenção de energia, para além disso são dispositivos que estão embebidos em objetos do dia-a-dia e têm a capacidade de medir e atuar sobre variáveis físicas. Estas características condicionam a sua operação, nomeadamente a forma como comunicam, processam e armazenam os dados.

São vários os projetos que recorrem ao uso de redes de sensores com resultados muito promissores, no entanto existem ainda muitos problemas para os quais não existem ainda soluções satisfatórias [1], nomeadamente a gestão de energia. É ainda necessário ter em conta que é frequente os nós mudarem de posição ou serem alteradas as relações de vizinhança, sendo por isso necessário considerar soluções que enderecem também os problemas decorrentes da mobilidade dos nós. É também frequente os nós serem instalados em locais de difícil acesso e expostos a condições de funcionamento difíceis, tais como ambientes industriais.

Após um período inicial instável devido à dificuldade em ter *hardware* viável e o desenvolvimento de software ser diferente para dispositivos de fabricantes diferentes, as redes de sensores atravessam agora um período de maior estabilidade. Por um lado, o *hardware* evoluiu ao ponto de existirem dispositivos com capacidade de

processamento, comunicação e baterias correspondentes a baixo custo. Por outro, foi fundamental a normalização de protocolos de modo a facilitar a integração de nós de diferentes fabricantes na mesma rede de soluções.

O facto dos nós terem suporte para a pilha protocolar TCP/IP facilita o desenvolvimento de aplicações, assim como a ligação à internet, independentemente dos protocolos das camadas inferiores e do facto de apresentarem restrições ao nível dos recursos [2].

O protocolo IEEE 802.15.4 [3] da camada de acesso à rede foi proposto com o objetivo de endereçar os requisitos das redes de sensores sem fios, caracterizadas pelos módulos de rádio de baixa potência, recursos muito limitados e com grandes taxas de erros na transmissão e receção de pacotes. Estas redes são designadas por *LoWPAN* [4].

Uma rede de sensores é normalmente composta por três tipos de nós. Os nós *Reduced Function Device* (RFD), têm transdutores (som, luz, presença, temperatura, humidade, movimento, posição, entre outros) que recolhem dados e enviam para a rede. Os nós encaminhadores *Full Function Device* (FFD) que podem ter transdutores como os RFD e adicionalmente suportam encaminhamento *multi-hop*. Normalmente, tanto os RFDs como os FFDs, são dispositivos com recursos muito limitados. Existem ainda os do tipo nós *Border Router*, que normalmente são nós com mais recursos, tal como maior capacidade de processamento e com menos restrições ao nível do consumo de energia, sendo por isso utilizados como interface com a Internet. [5]

Numa rede *LoWPAN* é frequente os nós não estarem ao alcance do *Border Router*, sendo por isso necessário utilizar comunicações *multi-hop* para que os pacotes cheguem ao destino. Desta forma os nós têm de ter a capacidade de receber um pacote e de o retransmitir para o nó seguinte. Desta forma, é necessário um protocolo de encaminhamento que opere em redes sem fios por rádio frequência, de preferência em malha.

O uso do protocolo IPv6 nas redes IoT é aceite de forma generalizada pela comunidade académica. No entanto, o protocolo IPv6 não foi especificado para funcionar sobre o protocolo IEEE 802.15.4, sendo por isso necessário utilizar uma camada de adaptação. A camada de adaptação 6LowPAN [6] foi definida entre a camada de ligação à rede e a camada de rede para permitir comunicações IPv6 em redes IEEE 802.15.4. [7].

Os sistemas operativos *open-source*, tais como o Contiki, o tinyOS e o FreeRTOS, desenhados para sistemas de baixos recursos, tornaram possível o desenvolvimento comum para várias plataformas, com suporte a pilhas protocolares e protocolos *standard*. Estes sistemas operativos tornam o desenvolvimento de serviços para as redes de sensores menos dispendiosos.

Uma rede de sensores pode conter milhares de nós que cooperam entre si. A dimensão da rede e o facto de não existir nenhuma infraestrutura estável tornam fundamental a adoção de mecanismos e processos automáticos para a sua gestão. A gestão de uma rede de sensores inclui o processo de gerir, monitorizar e controlar os nós. Atualmente, existem protocolos que são necessários para assegurar o funcionamento da rede e que podem também ser utilizados nas tarefas de gestão de uma rede de sensores, tais como o RPL, ND, DTLS, COAP.

Numa rede de sensores sem fios a segurança nas comunicações é importante, começando pela seleção de um processo criptográfico. Muitas aproximações adotam criptografia de chave simétrica, introduzindo um processo complexo de gestão de chaves. Com os desenvolvimentos tecnológicos é agora possível utilizar, de uma forma viável, criptografia de chave assimétrica. Estas operações requerem muito em termos computacionais e energéticos [8].

A gestão de chaves é fundamental para a criptografia. É o mecanismo que permite gerar, distribuir e manter as chaves da rede em segurança. A maior parte dos sistemas utilizam a pré-distribuição de chaves de forma a facilitar a gestão. Alguns protocolos utilizam teorias probabilísticas para calcular a probabilidade de um nó vizinho partilhar uma das suas chaves, outros associam a identidade do nó na gestão de chaves, outros esquemas utilizam *deployment* por conhecimento, para que alguns nós fiquem próximos geograficamente [8].

1.2 Conceitos fundamentais

A gestão de uma infraestrutura inclui as operações e os mecanismos que mantêm os serviços a funcionar de acordo com os requisitos, e por conseguinte é um aspecto fundamental em todas as redes de dados e em especial, nas redes com baixos recursos como é o caso das redes de sensores. São três os aspetos fundamentais da gestão; i) a

visibilidade, ii) o controlo e iii) a segurança. A **visibilidade** permite conhecer todos os recursos ligados em rede e de que forma estão organizados. O **controlo** permite tomar decisões com o objetivo de manter a rede operacional. Impedir que um nó comunique com outro ou que use a rede para comunicar com a Internet é um dos tipos de decisão que se pode tomar. A **segurança** é composta pelos mecanismos e procedimentos que garantem a privacidade, a autenticidade e a disponibilidade da infraestrutura e dos dados. Note-se que a segurança só é possível quando existe visibilidade e controlo, pois apenas conhecendo a rede é possível controlar os eventos, dispositivos e aplicar medidas de segurança [9].

Enquanto nas redes convencionais a configuração e monitorização das redes fica normalmente a cargo de um gestor de rede, nas redes de sensores, dado o número de nós envolvidos e o facto de não ser uma rede estável de acordo com a topologia e número de nós, pretende-se que os nós usem, tanto quanto possível, mecanismos de autoconfiguração de forma a funcionarem autonomamente desde o instante da instalação até a sua desativação. Em muitos cenários de aplicação não é conhecida a localização de cada um dos nós sensores que compõem a rede, ou sendo conhecida, não é prático ou viável o acesso a esses nós. Esta característica coloca desafios não só em termos de configuração, mas também em termos de segurança. [1]

O modelo de gestão funcional de uma rede denominada de *FCAPS (fault, configuration, accounting, performance and security)* [10], pode ser dividido em cinco categorias: **Gestão de falhas** – que tem como função gerar notificações, guardar erros de notificações e testar os recursos da rede para identificar falhas; **Gestão da configuração** – para monitorizar os sistemas da rede e a informação das configurações para que fiquem registadas as várias versões de *hardware* e *software*; **Gestão da contabilização** – para a recolha de informação relativa ao processamento para efeitos de faturação e cobrança; **Gestão do desempenho** – para determinar a disponibilidade da informação, a determinação da carga na rede e seus sistemas, tanto em condições naturais como artificiais. Deve também ter a capacidade de recolher informação sobre o desempenho da rede periodicamente para gerar estatísticas e conseguir planear atividades; **Gestão da segurança** – a segurança na gestão é a habilidade de autenticar utilizadores e/ou aplicações para garantir a confidencialidade, integridade e prevenir contra acessos não autorizados à informação de gestão. [10] .

A segurança e a privacidade dos dados são desafios a ter em conta nas redes de sensores. A Confidencialidade, Integridade e Disponibilidade são essenciais na troca de dados entre dispositivos IoT, a inteligência e autonomia dos dispositivos requerem uma responsabilidade para a proteção contra a corrupção dos dispositivos e a sua influência na rede [11]. Para mitigar esses problemas podem ser utilizados diferentes tipos de criptografia e soluções baseadas em processos. Desta forma, os sistemas de IoT não precisam apenas destes serviços mas também de implementações otimizadas [12].

Os sistemas de IoT são, em grande parte, redes sem fios, que são vulneráveis a todos os tipos de intrusões, incluindo acessos não autorizados, configurações defeituosas, interferências, ataques de *man-in-the-middle*, *spoofing*, ataques de *Denial of Service* (DoS), ataques de *brute force*, injeção de trafego, *etc* [13]. Para a proteção de ataques contra a confidencialidade, integridade e disponibilidade existem mecanismos criptográficos. Algoritmos de cifra protegem contra confidencialidade assim como o *Message Authentication Codes* (MAC) assegura integridade e autenticidade. Anteriormente, as implementações estavam preparadas para lidar com um modelo de ataques que necessitava de acesso físico aos nós, ou ataques de nós que estão ligados à rede, mas com a evolução das redes de sensores sem fios, o modelo de ataques também mudou. Com a disponibilização das redes de sensores na Internet através da comunicação *end-to-end* (sensor-internet) o risco de segurança aumentou, ou seja, aumentou a probabilidade de ataques exteriores diretamente aos nós da rede, que são normalmente dispositivos com baixas capacidades computacionais [12].

A confidencialidade, integridade e autenticidadepodem ser implementadas nas várias camadas da pilha protocolar, como IEE 802.15.4 para a camada de acesso à rede, e DTLS na camada de aplicação, com vários tipos de criptografia, de chave simétrica ou assimétrica e com vários esquemas de gestão de chaves, como por exemplo ter uma chave para se conectar à rede e uma chave para comunicar, sendo possível dar e revogar acessos à rede [14]. Normalmente a chave de rede é pré distribuída no *provisioning* e é utilizada pelo IEEE 802.15.4 com AES [15]. Em relação à implementação de um sistema com chave de comunicação pode ser utilizado o DTLS [16], tanto na variante de *Pre-shared Keys* (PSK) como na *Raw Public Keys* (RPK). Podem também ser utilizados processos auxiliares para combinar as chaves a serem utilizadas, como o algoritmo de *Diffie-Hellman* para que, de uma forma segura, seja possível estabelecer uma chave para utilizar no DTLS no modo PSK.

A presente dissertação tem como objectivo a criação de mecanismos de acesso a redes redes 6LoWPAN com multi-hop, para que seja possível efectuar a identificação e autenticação dos nós. Para isto é necessário considerar a utilização de mecanismos criptográficos em dispositivos com com restrições ao nível da capacidade de processamento e de armazenamento e com baixa capacidade de retenção de energia. É necessário ter em conta a gestão de chaves, como por exemplo o tempo de vida de uma chave e a sua exposição, pois é com base nestas que os mecanismos criptográficos funcionam.

1.3 Objetivos

A presente dissertação tem como principais objetivos:

- Identificar um nó na rede;
- Autenticar um nó na rede de uma forma segura;
- Garantir confidencialidade e integridade nas comunicações de uma rede de sensores sem fios.

1.4 Estrutura da dissertação

A presente dissertação encontra-se estruturada na seguinte forma:

No capítulo 2 são descritos os fundamentos teóricos relativos à gestão de uma rede de sensores sem fios, onde são aprofundados os conceitos de visibilidade, controlo e segurança, detalhando os protocolos e os mecanismos mais relevantes, tais como o DTLS.

No capítulo 3 é descrita a solução proposta no âmbito deste trabalho para a gestão da segurança de uma rede de sensores sem fios, as ferramentas utilizadas e a arquitetura da solução.

O capítulo 4 apresenta o ambiente de testes e os procedimentos realizados para validar a solução proposta.

No capítulo 5 são apresentadas as conclusões finais do trabalho realizado e são apontadas tópicos em aberto que podem ser endereçados em trabalhos futuros.

2 Gestão de segurança

Neste capítulo são descritos os fundamentos teóricos relativos à gestão de uma rede de sensores sem fios, onde são aprofundados os conceitos de visibilidade, controlo e segurança, detalhando protocolos e mecanismos utilizados, dando relevo ao DTLS.

2.1 Gestão

A gestão de uma rede é um conjunto de ações que asseguram que os seus recursos são utilizados de acordo com o fim para o qual a rede foi projetada. A gestão da rede compreende o conhecimento detalhado da infraestrutura sendo designado por visibilidade e controlo das ações que são efetuadas.

2.1.1 Visibilidade

A visibilidade é composta pelo conjunto de mecanismos, protocolos e procedimentos que permitem conhecer quais os dispositivos e de que forma se encontram ligados à rede. Para garantir a visibilidade é possível usar protocolos que normalmente são utilizados para assegurar a conectividade dos nós, não sendo por isso necessário desenvolver novos protocolos nem sobrecarregar os nós da rede. O *neighbor discovery* (ND) [17] e o RPL [18] são exemplos deste tipo de protocolos que transportam informação relevante acerca dos nós e da topologia da rede.

Os protocolos ND e o RPL são importantes para a gestão de uma rede, pois são utilizados pelos nós para descobrirm os seus vizinhos, para construírem as suas tabelas de encaminhamento,mantelas actualizadas, [9] resolver endereços da camada 2 e para a autoconfiguração dos parâmetros da rede. [5]

O Protocolo de ND do IPv6 convencional não é suportado pelo IEEE 802.15.4, por vários motivos, entre eles, o facto de as mensagens serem transportadas em pacotes *multicast*, e pelo tamanho dos cabeçalhos [19]. Foi necessário adaptar e otimizar o ND do IPv6 para suportar as necessidades do 6LoWPAN.

O protocolo ND adaptado situa-se entre a camada MAC e a camada de rede [19] é compatível com os mecanismos de gestão de energia normalmente utilizados pelo

hardware das redes IoT (i.e. CPU em modo de baixo consumo e com o módulo de rádio desligado), elimina a resolução de endereços através do uso de mensagens *multicast* com a introdução de um modelo de registo do nó no *Border Router*. Desta forma o *Border Router* tem um registo de todos os nós da rede, que é utilizado para o prefixo de mensagens *multi-hop* e compressão de cabeçalhos. [5]

Uma ligação é caracterizada por ser de fraca qualidade, baixa potência, de curta distância, e muitas vezes os nós podem estar adormecidos por longos períodos de tempo para poupar energia [19]. Desta forma, o ND foi adaptado para estar preparado para estas premissas.

Para colmatar o processo de resolução de nomes através do *multicast* foi criado um mecanismo para registo de endereços, o que também elimina a necessidade de enviar, periodicamente, uma mensagem de *router advertisement*, dando assim ao *host* a capacidade se anunciar quando chega à rede. Em muitos casos as mensagens de *multicast* foram substituídas por mensagens *unicast*. A aproximação utilizada no método de *routing*, *route-over ou mesh-under* [20] não influência o comportamento do protocolo [5].

O *Border Router* tem um papel importante nas redes *LowPan*, pois tem a responsabilidade de fazer de interface entre a rede *LowPan* e a Internet e ainda tem a responsabilidade de propagar o prefixo IPv6 e a compressão dos cabeçalhos. [5]

O *Border Router* mantém, também os endereços IPv6 dos nós da sua rede, e os Identificadores únicos (EUI-64) de modo a ter a capacidade de fazer a tradução de nomes e detetar endereços duplicados na camada de acesso à rede. O DHCPv6 pode ser utilizado para assegurar que os endereços na rede são únicos. Existem ainda outros métodos para detetar endereços duplicados e distribuir novos endereços, como a definição de três novas opções nas mensagens ICMPv6: *Address Registration Option* (ARO), *Authoritative Order Router Option* (ABRO) e 6LoWPAN *Context Options* (6CO). Foram criados ainda dois novos tipos de mensagens ICMPv6 para a deteção de endereços duplicados: *Duplicate Address Request* (DAR), *Duplicate Address Confirmation* (DAC). [5]

O processo para o registo de um nó na rede começa no ND, para determinar a vizinhança do nó e selecionar o melhor nó para comunicar. Quando um nó é inicializado é estabelecida uma ligação baseada no identificador único. Este manda uma mensagem

de *Router Solicitation* incluindo o *Source Link-Layer Address* (SLLA) para que o *router* possa responder com uma mensagem de *unicast* de *router advertisement*. A mensagem de *router advertisement* pode incluir as opções PIO, 6CO, ABRO, SLLAO. Figura 1

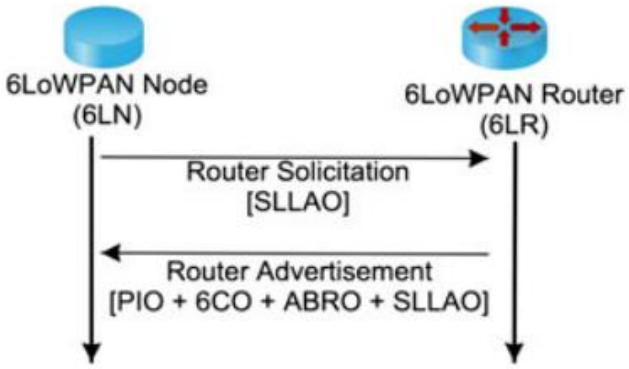


Figura 1 - Nó iniciou o processo de descoberta [5]

Quando um nó já tem a sua configuração, é enviada uma mensagem de *Neighbor solicitation* (NS) com a opção de registo no *Border Router*, que responde, em *unicast*, com uma mensagem do tipo *Neighbor Advertisement* (NA) com a opção ARO e o estado do registo. O estado indica se o registo foi efetuado com sucesso ou não, Figura 2. A falha pode estar associada ao facto de existerem endereços duplicados ou pela cache estar cheia [5].

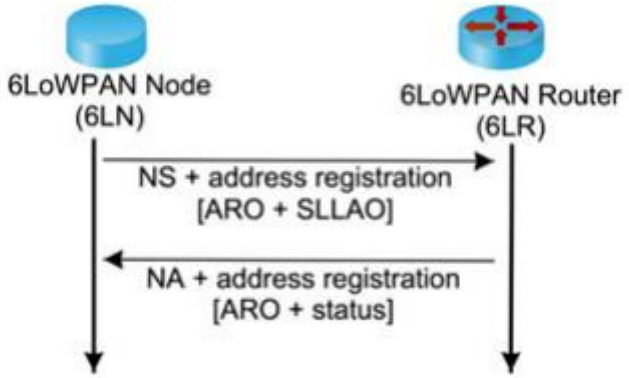


Figura 2 - Registo do nó [5]

Após o registo do ND estar concluído, o RPL é inicializado. O nó envia uma mensagem do tipo DODAG Information Solicitation (DIS). O Router envia uma mensagem do tipo DODAG Information Object (DIO), que contém informação sobre a hierarquia da rede, métricas e o prefixo IPv6 (PIO). Os *routers* enviam mensagens do tipo DIO em intervalos regulares, para manter a rede atualizada, por exemplo, se for recebida uma

mensagem com uma hierarquia melhor, o nó efetua um novo registo, enviando de novo um *Neighbour solicitation,* Figura 3. Quando uma mensagem do tipo DIO é recebida, fica estabelecido o caminho dos pacotes enviados pelo nó ao Border Router. [21]

Depois o nó envia uma mensagem do tipo Destination Advertisement Object (DAO) para a sua *default route,* Figura 3, ou seja, nó escolhido para enviar os pacotes, para serem encaminhadas até ao *Border Router*. O RPL [18] utiliza as DAO para estabelecer uma rota para que o *Border Router* consiga comunicar com o nó. Ao receber a mensagem DAO, o *Border Router* responde com uma confirmação (DAO ACK), considerando assim o processo de registo da rede completo. [17]

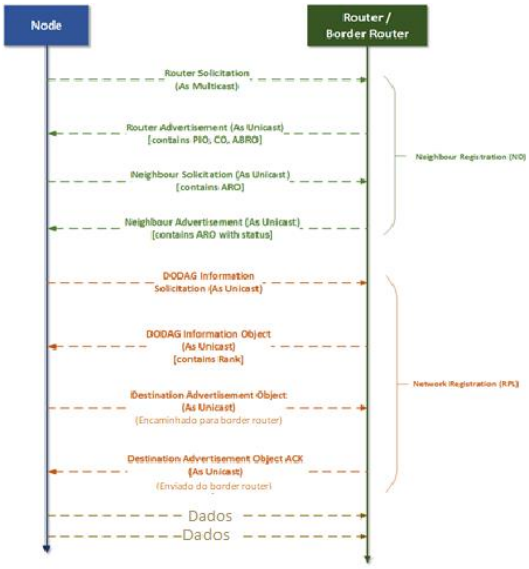


Figura 3 - Processo de registo de um nó no ND e RPL [17]

O mecanismo de registo e a opção SLLA nas mensagens de *router advertisement* têm informação suficiente para que os nós e os routers consigam traduzir endereços IPv6 para endereços da camada de ligação à rede. A informação enviada nas mensagens de NA têm um tempo de vida associado, se expirar é necessário repetir todo o processo.

Os nós podem receber mensagem de RA de múltiplos *Border Routers*. Neste caso estes devem tentar o registo em mais do que um *Border Router* de forma a aumentar a resistência, e redundância da rede. As mensagens de NS também são utilizadas para detetar se os nós vizinhos ficaram incontactáveis. Esta funcionalidade é utilizada para validar se o *default router* está ao alcance.

A deteção de endereços duplicados é um processo explicado na Figura 4. Este pode ser utilizado em redes do tipo *route over* para assegurar que os endereços não únicos para endereços que não são baseados no identificador único (EUI-64). O processo é semelhante ao processo de registo normal, com a diferença de um nó encaminhador ter de confirmar com o *Border Router* se é possível o registo do nó em causa. Este processo é efetuado utilizando as mensagens ICMPv6 DAR e DAC.

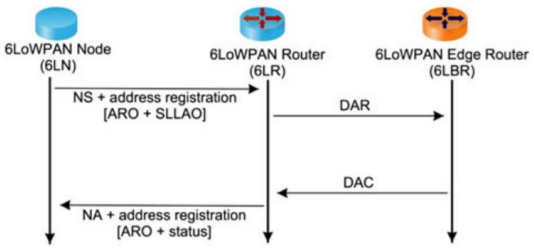


Figura 4 - Registo de um host com multihop e DAD [5]

Depois de completo o ND o RPL é inicializado para efetuar o registo do nó na rede (*Border Routers*).

O RPL é o protocolo de encaminhamento para as redes de sensores sem fios. É do tipo *distance-vector* (DV), e *source routing protocol* que foi desenhado para operar sobre vários mecanismos, como IEEE 802.15.4 PHY e a camada de acesso ao meio. Os nós enviam periodicamente métricas para um ponto central com mensagens do tipo multiponto-ponto (*Upwards*), assim como o ponto central envia mensagens do tipo ponto-multiponto (*Downwards*) para a rede de forma a manter a rede atualizada, ainda são suportadas mensagens do tipo ponto-ponto. [22] O RPL representa um protocolo de encaminhamento para redes de sensores sem fios, onde existem recursos limitados, como a energia, recursos computacionais e a perda de pacotes [18]. O RPL tem em consideração duas características das RSSF, a largura de banda baixa, e a existência de taxas de erro na comunicação muito elevadas nas ligações. Uma ligação instável, não é apenas caracterizada pela elevada taxa de erro, mas também, pelo facto de estar longos períodos de tempo inacessível, o que faz com que o protocolo tenha de ter a capacidade de ser altamente adaptável às condições da rede para dar rotas alternativas quando as *default routes* estão inacessíveis. O protocolo utiliza o conceito topológico de *Directed Acyclic Graphs* (DAGs), que definem uma estrutura em árvore onde são designadas *default routes* entre os nós. Contudo a estrutura é mais do que uma típica árvore, pois

os nós podem estar associados a vários nós-pai, ou seja o RPL organiza os nós como *Destination-Oriented DAGs* (DODAGs) onde os nós com mais ligações (*sinks*) ou os nós que servem de *default route* para a Internet são *roots* das DAGs [18]. Uma rede pode ter mais do que uma DODAGs que formam uma instância de RPL identificada por um identificador único, chamado *RPLInstanceID*. Uma rede pode ter várias instâncias de RPL concorrentes, sendo que são, em termos lógicos, separadas.

Outro facto relevante do RPL é que combina topologias *mesh* e hierárquicas, ou seja, por um lado força os nós a organizarem-se e criarem uma hierarquia criando as DODAGs, baseado numa relação pai-filho, por outro lado suporta uma topologia *mesh*, porque possibilita encaminhamento através de irmãos em vez de pais ou filhos. Esta combinação permite uma grande flexibilidade no encaminhamento e a gestão da rede. [18].

As principais características são: Auto configuração – RPL está em conformidade com o IPv6, logo as RSSF beneficiam das características básicas de um protocolo de encaminhamento IP, principalmente a descoberta dinâmica de caminhos e destinos, garantido nas RSSF pelo *Neighbor Discovery*; Habilidade de se reparar – RPL adapta-se a alterações logicas da rede e à falha de nós, pois implementa mecanismos onde são escolhidos mais do que um pai para cada nó de forma a existir redundância e a mitigar os possíveis erros. Deteção de *Loops* – Uma DAG é acíclica por natureza logo um nó numa DAG deve ter um *rank* mais elevado que os seus pais. O RPL implementa mecanimos para detetar *loops* em caso de mudanças de tecnologia, e também sistemas para a reparação global e local para quando existirem *loops;* Indepência e transparência – RPL foi desenhado para trabalhar numa arquitetura IP e funcionar sobre vários tipos de camadas de acesso, tanto das RSSF ou redes convencionais; Múltiplos *Border Routers* – É possível construir várias DAGs numa rede RPL, onde cada DAG tem uma *root* . Um nó pode pertencer a várias instâncias e pode desempenhar diferentes funções em cada instância, dando assim à rede alta disponibilidade e balanceamento de carga [18].

2.1.2 Controlo

Uma das características das redes 6LoWPAN é a sua capacidade dos nós se auto organizarem e de se auto configurarem, pois assim minimizam a necessidade da

intervenção do gestor, para além de aumentar a robustez da rede devido a alterações na topologia. A autoconfiguração também pode ser explorada para ataques à segurança, como por exemplo a comunicação entre nós não autorizados e nós autorizados (i.e. nós da rede que fazem parte da infraestrutura) e a comunicação entre nós não autorizados e nós ligados à Internet. A solução para este problema pode passar por um mecanismo criptográfico no qual são admitidos apenas os nós que conheçam uma chave. Neste caso, os mecanismos de criptografia são utilizados para autenticar os nós. [15]

Devido às características intrínsecas dos nós da rede, tais como a baixa capacidade de processamento e de retenção de energia, existem algumas limitações em relação ao uso dos mecanismos criptográficos, sendo por isso crítico a escolha do mecanismo a usar. Os mecanismos criptográficos podem ser classificados quanto ao número de chaves envolvidas e quanto ao tipo de chaves, chave simétrica e chave assimétrica. Na criptografia de chave simétrica é utilizada apenas uma chave, a qual terá de ser do conhecimento de ambas as partes antes de ser iniciada a comunicação. Tem a vantagem de ser mais leve em termos computacionais e mais rápido quando comparado com os mecanismos de chave assimétrica. Neste tipo de algoritmos o texto/mensagem é cifrado diretamente com uma chave pré-partilhada e decifrada com essa mesma chave. Este tipo de encriptação garante autenticação, integridade e autenticidade do nó. [23]. Na criptografia de chave assimétrica, ao contrário da criptografia de chave simétrica, não é necessário ambas as partes conhecerem a chave para encriptação da mensagem. Neste tipo de criptografia existem duas chaves, uma privada, do conhecimento apenas do seu "proprietário" e uma pública, que pode ser divulgada livremente. Com recurso a certificados digitais podemos, com este tipo de criptografia, garantir a autenticidade da mensagem tendo, no entanto, o custo computacional é bastante elevado [24]

Os algoritmos de criptografia como o RSA são muito pesados computacionalmente para as redes de sensores sem fios. Para mitigar este problema foi criado um mecanismo baseado em curvas elípticas (ECC). De acordo com [25], a chave ECC de 160 bits tem o mesmo nível de segurança que a de 1024 bits do RSA. Este número de bits não é constante pois cerca de 224 bits em curvas elípticas equivalem a 2048 bits em RSA.

A gestão de chaves pode ser feita com recurso a uma chave simétrica pré distribuída de modo a garantir a autenticidade do nó. Esta chave pode também ter como objetivo cifrar os pacotes na camada IEEE 802.15.4. Com esta medida de segurança é possível ter

controlo sobre quais os nós que se ligam à rede, diminuindo assim as mensagens propagadas por nós não autorizados [15].

Também é possível utilizar protocolos como DTLS que utiliza criptografia de chave assimétrica [24] na camada de aplicação. Este tem vários modos de funcionamento entre eles *Pre shared key* (PSK) [15] onde é necessário que ambos os nós conheçam a chave. Este modo de funcionamento é complementado por uma gestão de chaves do tipo pré distribuição, ou *Diffie-Hellman* onde é combinada uma chave que apenas os dois conhecem, através de mecanismos de criptografia de chave assimétrica, garantindo assim a autenticidade, integridade e confidencialidade das mensagens na rede. Este modelo de criptografia na camada de aplicação tem a vantagem de poder existir uma entidade na rede que pode revogar ou substituir as chaves, sem alterar a topologia da rede [15].

2.2 Gestão de redes LoWPAN

Existem soluções de gestão para as redes *LowPAN* que foram originalmente propostas para as redes locais (i.e. redes nas quais os nós intermédios não apresentam constrangimentos tão severos ao nível da capacidade de computação e de retenção de energia), tais como o SNMP (*Simple Network Management Protocol*) [26], ou o desenvolvimento e utilização de *frameworks* como [27], onde é criada uma camada de abstração, na forma de uma *API RestFull* de forma a ser mais fácil manipular (i.e. mecanismos de descoberta e interação entre nós) dispositivos e dispositivos terminais. Existem ainda protocolos com o RPL e o ND que têm mecanismos para a descoberta de nós na rede e encaminhamento de pacotes [28]. Muitas soluções implicam a implementação de um *gateway* à medida, como [29] e [30] que apresentaram uma implementação de uma *gateway* programável e de baixo custo para sistemas embebidos, utilizada para monitorização. No artigo referido os autores apresentam as plataformas de *hardware* utilizadas, assim como, sistema operativo e os protocolos em detalhe. Estes defendem que a *gateway* pode fazer a ponte entre as redes tradicionais e as redes de sensores [31].

A arquitetura da *gateway* tem módulos com diferentes protocolos de comunicação de forma a suportar dispositivos com diferentes protocolos da camada 2. O

desenvolvimento de novas funcionalidades torna-se mais simples pela unificação das interfaces exteriores. O artigo também menciona um protocolo de tradução para os diferentes tipos de dados de sensores de forma a ficarem com formato uniforme. De forma a evitar o trabalho repetitivo de configuração de um nó sensor é proposto um sistema onde o servidor assiste nas configurações como em [32].

2.3 Mecanismos para a segurança

Considerando as limitações de uma rede sem fios é possível verificar que os mecanismos/protocolos de segurança existentes para as redes convencionais, não se adequam. Existem várias propostas para protocolos *standard* a serem definidos pelo IEEE, que vão ter um papel fundamental no futuro das aplicações de IoT. [33]

Nesta secção vão ser analisados os aspetos relativos à segurança dos protocolos da pilha protocolar de uma rede 6LowPAN. Nas camadas inferiores é frequente usar-se o protocolo IEEE 802.15.4. Desta forma este protocolo impõe as regras para a comunicação nas camadas inferiores da pilha protocolar. A segurança nesta camada é garantida por uma extensão do próprio protocolo que suporta criptografia de chave simétrica AES, tendo vários modos de funcionamento [33]; Na camada de adaptação é utilizado 6LoWPAN de forma a permitir a utilização de IPv6, na camada de transporte, nas redes de sensores. Nesta camada ainda não existem métodos definidos para a segurança**,** apenas propostas de adaptação do IPSec e IKE tentando adaptar o AH e o ESP [33]; Na camada de rede é utilizado o protocolo de encaminhamento RPL desenhado especificamente para as redes de sensores sem fios. Este protocolo define versões seguras de várias mensagens de controlo assim como três modos de funcionamento, sendo estes: Sem Segurança; Pré-instalado, onde uma chave simétrica é pré configurada de forma ao nó ter a capacidade para se juntar a uma instância de RPL; Autenticado, onde o nó se junta com uma chave pré configurada mas depois obtêm outra chave de forma a conseguir encaminhar mensagens [33]**;** Na camada de aplicação é utilizado COAP, onde a segurança é garantida através do DTLS [33].

IEEE 802.15.4

No IEE 802.15.4 existem dois tipos de pacotes relevantes para a segurança, os de dados, e do que *acknowledgment*. Cada pacote contém um indicador para indicar o seu tipo, se a segurança está ativa ou não, o modo de endereçamento utilizado, se é necessário o destinatário enviar um *acknowledgment*, e opcionalmente o endereço de origem e de destino Figura 5

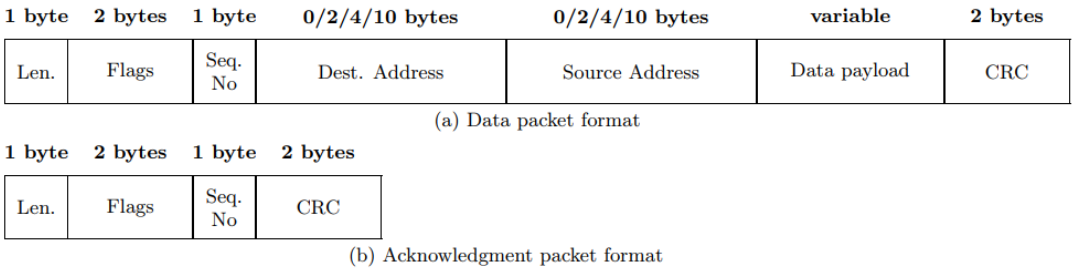


Figura 5 - Formato dos pacotes de dados a) e acknowledment b)

Os pacotes de *acknowledgment* são enviados se assim for pedido, normalmente quando são comunicações *unicast* e não *broadcast*. Este é semelhante ao pacote de dados mas apenas tem o necessário, são retirados os campos de destino e de origem assim como o *payload*, são enviados os indicadores na mesma, assim como o número de sequência ao qual está a responder que recebeu.

Existem vários níveis de segurança como se pode observar na Figura 6. *Null* significa que não tem segurança, AES-CTR apenas garante a confidencialidade dos dados, AES-CBC-MAC apenas garante autenticação e AES-CCM garante confidencialidade e autenticação. Cada categoria que suporta autenticação vem com três variantes a nível do tamanho do MAC.

Nome	Descrição
Null	Sem segurança
AES-CTR	Apenas encriptação, Modo CTR
AES-CBC-MAC-128	128 bit MAC
AES-CBC-MAC-64	64 bit MAC
AES-CBC-MAC-32	32 bit MAC

AES-CCM-128	Encriptação e 128 bit MAC
AES-CCM-64	Encriptação e 64 bit MAC
AES-CCM-32	Encriptação e 32 bit MAC

Figura 6 - Vários níveis de seguraança do AES [34]

Com a adição da segurança os cabeçalhos do protocolo 802.15.4 são alterados, como é possível verificar na Figura 7. Quando incluído o MAC protege todo o pacote. Os restantes campos são adicionados no campo do *DataPayload*.

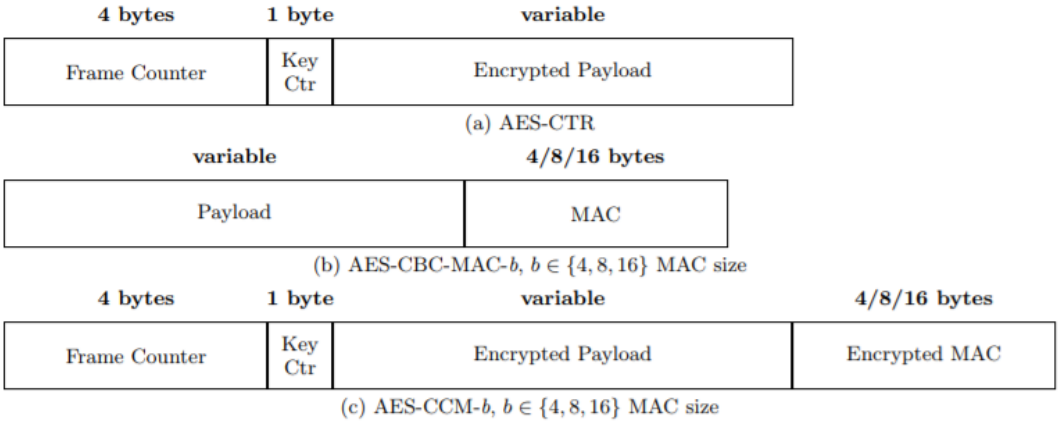


Figura 7 – Formato do campo de dados quando adicionada a segurança [34]

A proteção contra ataques de reenvio previne que uma mensagem trocada entre dois nós autorizados na rede não possa ser enviada mais tarde por um nó não autorizado e esta ser processada. Teoricamente a mensagem era aceite, pois tinha um MAC válido, é necessário recorrer ao número de sequência para detetar estes ataques. O número de sequência vai incrementando com a troca de pacotes entre dois nós autorizados. Desta forma se um nó não autorizado apanhar a mensagem e a enviar de novo para o destinatário, o número de sequência vai estar desatualizado, logo o pacote vai ser descartado.

2.3.1 Gestão de chaves

De forma a existir segurança nas redes é necessário que seja utilizado um sistema de gestão apropriado, que mantenha as chaves em segredo, pois é sobre estas primitivas que o restante processo criptográfico está construído. Se um atacante conseguir encontrar uma chave, todo os sistema fica comprometido. [13] Estes mecanismos

podem ser utilizados para estabelecer e distribuir os vários tipos de chaves criptográficas na rede como chaves individuais, chaves entre dois nós, ou chaves de grupo [8]. O principal objetivo é garantir que existe uma chave para os nós que devem comunicar, deve suportar alterações na rede, como a adição ou remoção de um nó, ser flexível num ambiente de *deployment* indefinido, e nós não autorizados não se devem conseguir juntar à rede. A gestão de segurança deve assentar sobre os princípios de *Resistance*, *Resilience* e *Revocation (*3Rs*)* [9].

Devido à importância e dificuldade para conseguir um esquema de gestão de chaves eficaz para qualquer caso, a segurança de chaves pode ser dividida em vários sub-problemas, sendo estes [35]:

- Pré distribuição de chaves - As chaves são carregadas nos nós no aprovisionamento.
- Descoberta de vizinhos - O nó tem de ser capaz de analisar a vizinhança para descobrir os seus vizinhos e suas chaves.
- Estabelecimento de chaves de comunicação - Cada nó tem de saber a que nós consegue chegar através de *multihop*, e ter as suas chaves.
- Isolar nós comprometidos - A rede tem de ter a capacidade de identificar e isolar os nós danificados ou com comportamentos que não são comuns.
- Redistribuição das chaves - De forma a manter a segurança as chaves devem ser trocadas periodicamente.

Existem várias abordagens propostas para a gestão de chaves que podem ser utilizados no contexto das redes de sensores sem fios, contudo estes esquemas são também utilizados nas redes convencionais.

- Network Keying

É a abordagem mais simples e consequentemente a mais insegura, neste cenário a chave é pré-carregada em todos os nós antes do *deployment* e todos comunicam com essa chave [35]. Tem como pontos fortes a pouca complexidade de implementação e escalabilidade, todos os nós que se juntarem à rede apenas têm de ter a chave pré-carregada. Tem como pontos fracos a grande vulnerabilidade, quando uma chave é utilizada por muito tempo aumenta a probabilidade de um ataque à rede ter sucesso,

neste cenário se um nó for comprometido, toda a rede é comprometida. Não existe maneira de revogar a chave apenas para 1 nó ou gerar outra chave [12].

- Pairwise Keying

Esta abordagem é a mais segura, porém não é escalável e consome muitos recursos. Cada nó é pré-carregado com uma chave para comunicação com todos os outros nós. Para isto são geradas N-1 chaves, onde o N é o número de nós na rede, e todos os nós têm todas as chaves. Tem como pontos fortes a facilidade de revogação de chaves de um sensor, garante a confidencialidade e a autenticidade. Tem como pontos fracos a elevada necessidade de memória é a pouca escalabilidade, pois com a entrada de um novo nó é necessário distribuir a sua chave por todos os outros nós [35].

- Group Keying

As chaves de grupo estão divididas em três grandes grupos, sendo estes [35]:

Centralizado - Existe uma entidade única (centro de distribuição de chaves) - KDC ou Central authority (CA). Mantém o grupo atribuindo uma chave para comunicar na rede aos membros do grupo, e também é responsável por partilhar a chave de encriptação da chave anterior entre todos os elementos do grupo.

Descentralizado - O grupo é subdividido em vários grupos. E cada grupo tem um controlador, como se existissem vários centros de distribuição.

Distribuído - A gestão das chaves é feita por todos.

- Trusted Key Distribution Center (KDC)

Esta aproximação tem como objetivo mitigar os pontos fracos do emparelhamento de chaves. Para isso é construído um centro de distribuição de chaves, este método resolve o problema do furto ou replicação de nós mas continua a não ser escalável, porque cada par de nó obtém previamente a sua chave do centro de distribuição para cada sessão. Este modelo tem um ponto único de falha, o centro de distribuição de chaves, e ainda

tem um grande *overhead* nas comunicações, podendo saturar os nós vizinhos do centro [35].

- Public Key Cryptography in Wireless Sensor Networks

A criptografia de chave assimétrica não é considerada para as redes de sensores devido aos cálculos sofisticados necessários. Contudo têm aparecido alguns algoritmos como o ECC (*Elliptic Curve Cryptography*), que tem sido utilizado ultimamente nas redes de sensores. ECC consegue ter o mesmo nível de segurança comparando com RSA, com uma chave menor. Algumas aproximações híbridas misturam chaves assimétricas com simétricas utilizando uma adaptação do algoritmo de *Diffie-Hellman* [35].

2.4 DTLS

DTLS (*Datagram Transport Layer Security*) [33] é um protocolo de comunicação criado para garantir a segurança, Confidencialidade, Integridade e Autenticidade. O DTLS é na prática TLS, mas garante o funcionamento com UDP [36]. DTLS é utilizado nas redes de sensores pois garante a segurança *end-to-end* na camada de aplicação. Funciona por exemplo com o COAP que apesar de utilizar UDP, que não é confiável pois não confirma a recção de pacotes, tem mecanismos que colmatam as falhas do UDP [37].

O Impacto do DTLS numa rede tem muito a ver com o processo de *handshake*, e com o processamento de cifra das mensagens durante uma comunicação. A utilização de COAP e DTLS dá quatro tipos de segurança. *NoSec*, que na prática são mensagens não cifradas. *PreSharedKey*, onde as chaves são pré-distribuídas e é utilizada criptografia de chave simétrica. *RawPublicKeys*, onde é utilizada criptografia de chave assimétrica, as chaves são trocadas e o nó é reprogramado para utilizar a nova chave. Certificados, este é o método mais seguro, é necessário existir uma entidade confiável [36].

O DTLS utiliza criptografia de chave assimétrica embora não com o algoritmo mais conhecido (RSA), devido a ser muito pesado computacionalmente, mas sim com *Elliptic Curve Cryptography* (ECC). Este suporta autenticação com *Elliptic Curve Digital Signature Algorithm* (ECDSA), a troca de chaves utilizando ECC *Diffie-*

Hellman e, *Elliptic Curve Diffie-Hellman Algorithm with Ephemeral keys* (ECDHE). [38]

O objetivo da criação do DTLS é em primeiro lugar dar suporte a aplicações do tipo cliente-servidor, foi para este tipo de aplicações que o TLS foi desenhado e funciona bem. Idealmente um canal seguro de datagramas iria substituir os fortes mecanismos criptográficos baseados em IP, e teriam autenticação na camada de aplicação. [37]

IPsec é um protocolo peer-to-peer, e era expectável dar suporte a pacotes UDP gerados por aplicações do tipo cliente-servidor, contudo existem problemas associados sendo o principal, o facto de o IPsec residir na camada de rede ao invés da camada de aplicação. O IPsec na prática é uma junção de três protocolos, Authentication Header (AH), *Encapsulating Security Payload* (ESP) e Internet Key Exchange (IKE), onde AH e ESP são implementados diretamente na *stack* (*kernel*), e têm como objetivo assegurar a segurança nas comunicações, e o IKE, implementado como o *deamon* (serviço), utilizado para estabelecer chaves e outros parâmetros de segurança. A residência do IPsec no *kernel* torna este protocolo pouco portável, com isto vem a falta de normalização e a dificuldade em adaptar a API a aplicações. Neste aspeto o TLS tem a vantagem de ter uma interface simples. [37]

A negociação das chaves é mais simples de implementar numa ligação confiável (TCP) do que numa não confiável (UDP). Desta forma uma das possíveis abordagens era a criação de dois *sockets*, um TCP para fazer a negociação de chaves, e outro UDP para as comunicações. Esta aproximação facilita a implementação da negociação das chaves, pois não é necessário implementar um mecanismo para este efeito na camada de aplicação, sendo o problema resolvido na camada de transporte (TCP), contudo introduz o problema da sincronização de dois *sockets*, especialmente na renegociação de chaves. Se o *socket* TCP for terminado no final da negociação, se existir uma negociação, é necessário implementar um mecanismo de retransmissão (eliminando a grande vantagem desta aproximação), se o *socket* permanecer aberto está a consumir recursos desnecessariamente, com a desvantagem dos sistemas operativos não lidarem bem com muitos sockets abertos. Esta aproximação ainda tem a desvantagem do interface com um UDP *server* convencional, que espera *uma stream read-write* em apenas um *socket*, aumentando assim a complexidade na implementação. [37]

Estas considerações levaram à conclusão de que era necessário existir apenas um *socket* onde são realizadas todas as comunicações, o handshake e a comunicação de dados, resolvendo assim o problema da não confiabilidade da UDP na camada de aplicação.

2.4.1 Requisitos

Após as conclusões retiradas sobre as considerações anteriores, concluindo que DTLS teria de ser executado no espaço do utilizador, como uma aplicação, e que todas as comunicações seriam sobre apenas um socket, era necessário adaptar o protocolo TLS ao DTLS. Era necessário manter o protocolo TLS inalterado, sempre que fosse possível, de modo a aproveitar a resiliência que foi ganhando ao longo dos anos, e quando era necessário fazer mudanças a ideia era ir buscar ideias/implementações a protocolos existentes como o IPsec. O DTLS é explicitamente desenhado para ser compatível com a comunicação UDP para facilitar o desenvolvimento e integração nas aplicações desenvolvidas por programadores. [37]

1. **Transporte de datagramas (UDP)** - DTLS deve ser capaz de fazer a negociação das chaves e a comunicação sobre apenas um socket, esta propriedade vai permitir que para assegurar segurança nas aplicações seja apenas necessário substituir um simples socket UDP por um socket UDP gerido pelo DTLS, tornado assim num soket seguro.
2. **Negociação de chaves confiável** - DTLS deve implementar um mecanismo que assegure autenticação de endpoints, negociação e renegociação de chaves de uma forma confiável. Como DTLS é executado sobre UDP, deve implementar um mecanismo para retransmissão de pacotes, para assegurar que o processo de handshake é finalizado com sucesso.
3. **Segurança** - DTLS deve assegurar confidencialidade e integridade na transmissão de dados, deve também opcionalmente detetar pacotes duplicados, para esta funcionalidade são utilizados *cookies*.
4. **Portabilidade** - A possibilidade de implementação do TLS em *user space*, sem alterar o *kernel* tem sido um dos principais fatores para a utilização e aceitação do TLS. Esta funcionalidade permite aos programadores utilizarem DTLS independentemente do sistema operativo. Para que o DTLS seja também utilizado e aceite deve ser implementado em *user space*.

5. **Reduzir alterações** - DTLS deve ser o mais parecido com TLS possível. Com o passar dos anos foi sendo alterado para estar preparado para vários ataques, tornando-o assim mais resiliente e robusto. Minimizar as alterações reduz o risco de introduzir uma falha de segurança no protocolo. Adicionalmente quanto menos alterações mais simples se torna a sua implantação, aproveitando também as implementações de aceleração de *hardware* para criptografia assimétrica.

2.4.2 Operação

DTLS Handshake - Handshake é um algoritmo utilizado para estabelecer uma nova ligação efetuar a renegociação de parâmetros de segurança, como a cifra a ser utilizada, algoritmos de *hash* ou compressão e estabelecimento de chaves. Este processo é iniciado pelo cliente que envia uma mensagem do tipo *ClientHello* que contém a versão do protocolo, a lista de algoritmos, métodos de compressão que o cliente suporta e finalmente um *cookie* para prevenir ataques de *Denial of Service (DoS).* [37]

Uma das alterações do TLS para DTLS no processo de handshake é a introdução da mensagem *HelloVerifyRequest* para validar ou atualizar o *cookie* do cliente. Este *cookie* é gerado de forma a não estar ligado a sessões (*stateless*).

O servidor responde com três mensagens, *ServerHello* que contém a escolha da versão e dos algoritmos a utilizar, assim como um número aleatório. A mensagem do tipo *Certificate* contém a chave do servidor e o *ServerHelloDone* indica que não existem mais mensagens. Esta mensagem é necessária pois o número de mensagens num handshake é variável. [37]

O cliente gera uma sequência de números aleatórios para a *PreMasterSecret* para ser utilizado como base de chave a estabelecer. O cliente cifra a *PreMasterSecret* com as chaves assimétricas e envia para o servidor com através da mensagem *ClientKeyExchange.* Com a mensagem *ChangeChipherSpec* o cliente indica que se encontra a alterar as suas configurações para as negociadas. Finalmente é enviada a mensagem *Finished* que contém o MAC (*Message Authentication Code*) das mensagens anteriores, sendo esta mensagem cifrada com os parâmetros estabelecidos.

Como as mensagens de *Handshake* podem ser perdidas, DTLS necessita de um mecanismo de retransmissão, este é implementado com um timer no cliente e no servidor. [37]

As mensagens de *handshake* podem ser grandes ao ponto de excederem o tamanho máximo de uma mensagem de *Handshake*, desta forma foi necessário criar um mecanismo e uma estrutura para a fragmentação. A mensagem de *handshake* contém o tamanho máximo da mensagem para o espaço ser alocado, sem ter em conta a ordem que os pacotes são recebidos. [37]

Record Layer - Como no TLS, todos os dados apenas podem ser processados se toda a mensagem se encontra disponível. Para evitar lidar com o problema da fragmentação, é necessário os dados a serem transmidos se enquandrarem apenas num datagrama. Esta aproximação tem três grandes vantagens. Não é necessário fazer *buffer* de pacotes, utilizando a memória de uma forma mais inteligente, prevenindo também ataques de DoS. A possibilidade de um fragmento ser perdido, torna os outros fragmentos sem efeito, pois não podem ser processados. Não é claro quanto tempo se deve fazer *buffer* dos fragmentos antes de serem descartados. [37]

Foram introduzidos dois campos no DTLS. *Epoch* utilizado para determinar qual é a cifra utilizada, este campo é incrementado ou gerado cada vez que existe uma renegociação. Número de sequência é utilizado para proteção de ataques de reenvio. [37]

2.4.3 Outros mecanismos do DTLS

Para além do *Handshake* o DTLS contém outros mecanismos necessários para o seu funcionamento [37].

Alert Protocol - Estes mecanismos têm por objetivo alertar para erros ou avisos que possam ocorrer, por exemplo um certificado não ser válido. As mensagens de aviso são meramente informativas, a sessão pode continuar estabelecida. As mensagens de erro encerram a ligação.

Replay Check - Dado que as mensagens são cifradas um atacante não tem a capacidade para alterar as mensagens, mas como a não existe uma sessão é possível copiar uma mensagem e re-enviar. Para mitigar este problema existe o mecanismo de *ReplayCheck*,

durante a comunicação existe um registo do Record Sequence Numbers (RSN), se a mensagem com aquele RSN já foi processada a mensagem é descartada.

Heatbeat Extension - Nas comunicações UDP não existe nenhum mecanismo para saber se no caso do envio de uma mensagem não existir resposta é apenas porque não exista resposta a dar, ou porque o recetor simplesmente se desconectou da rede. DTLS tem dois tipos de mensagem para resolver este problema *HeartbeatRequest* e *HeartbeatResponse*, esta são utilizadas como *keep-alive*, pois quando é recebida uma mensagem do tipo *HeartbeatRequest* é necessário responder com um *HeartbeatResponse.*

2.4.4 Exemplo de Handshake

Numa comunicação por DTLS é necessário exisir um processo de *Handshake*, para principalmente ser feita a troca de chaves, como se pode observar na Figura 8

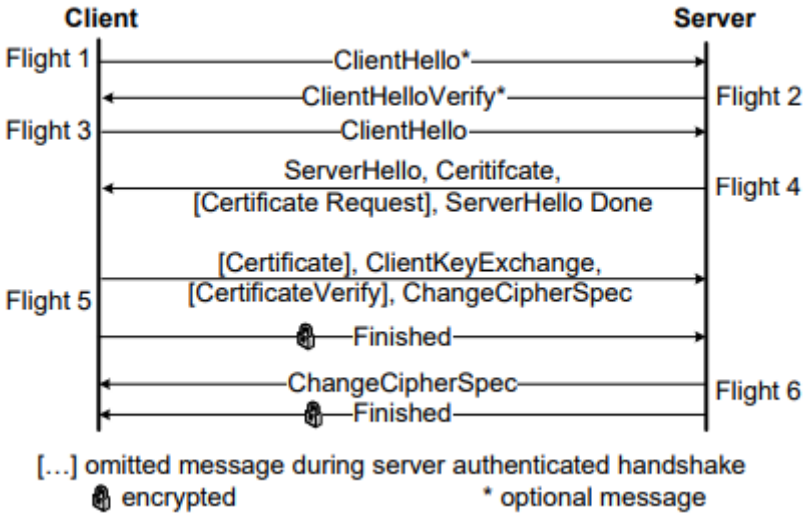


Figura 8 – Handshake de uma autenticação por DTLS [16]

Os primeiros passos 1 e 2 são opcionais, são utilizados para prevenir contra ataques de *Denial of Service (DoS)*. O Cliente tem de provar que consegue receber dados como também enviar. A partir do passo 3, o processo é obrigatório, onde são enviadas informações como a versão do protocolo e uma lista com as cifras que suporta. De seguida o Servidor responde com a cifra escolhida assim como o seu certificado. Se o processo decorrer normalmente o cliente envia o seu certificado. De seguida o servidor envia uma mensagem com metade da chave pré-partilhada com o seu certificado. A

outra metade da chave foi enviada de uma forma desprotegida no *Server Hello*, A chave a ser utilizada para as comunicações é gerada a partir da chave transmitida em duas partes. Desta forma é garantida: Autenticidade: quem recebe as mensagens consegue identificar com quem estão a comunicar; Integridade: é possível detetar se uma mensagem foi alterada; Confidencialidade: os nós não autorizados não conseguem decifrar a mensagem.

3 Solução proposta

3.1 Introdução

Nos mecanismos criptográficos modernos os algoritmos são públicos e apenas a chave é secreta. É por este motivo que a gestão das chaves é considerada uma tarefa crítica em todos os sistemas de segurança que usem criptografia. A gestão de chaves em redes constituídas por dispositivos com restrições ao nível da capacidade de processamento e de armazenamento e com baixa capacidade de retenção de energia é uma tarefa ainda mais complexa. A troca de chaves é a ação mais complexa da gestão de chaves, motivo pelo qual as chaves têm um tempo de vida, e consequentemente de exposição, muito longo. Propor, concretizar e avaliar uma solução que permita impedir que nós, não autorizados, se liguem à rede é objetivo deste trabalho. Uma solução de controlo de acesso à rede não só reduz o número de ataques internos a que a rede está sujeita, como também facilita a gestão da rede. Este mecanismo recorre ao uso de duas chaves, a primeira é utilizada para autenticar o nó e a segunda para cifrar os dados gerados pelo nó. A solução proposta funciona em redes em malha onde as comunicações do tipo *multi-hop* são predominantes.

3.2 Controlo de acessos

O método de gestão de rede proposto utiliza os protocolos necessários para o funcionamento de uma rede. O acréscimo de *overhead* é mínimo, uma vez que a informação adicional é transportada em pacotes gerados pelos mecanismos próprios deste tipo de redes que são necessários para manter a rede operacional, tais como o RPL e o *Neighbor Discovery*. A informação relativa aos nós ligados à rede é determinada a partir do *Neighbor Discovery* e através do RPL é possível determinar a topologia da rede. A maior parte do acréscimo de processamento e de armazenamento é suportado pelo *Border Router*. Note-se que o *Border Router* é normalmente suportado por *hardware* sem restrições ao nível do consumo de energia e com maior capacidade de armazenamento e de processamento que os nós sensores.

Através da ativação da segurança na camada IEEE 802.15.4, é possível criar uma chave de rede para garantir que apenas nós pré configurados com uma determinada chave se podem ligar à rede. Com o DTLS é possível ter uma gestão de chaves dinâmica, com chaves individuais ou de grupo, com criptografia de chave assimétrica e comunicação *end-to-end*, e desta forma, é possível controlar os acessos à rede, revogando ou permitindo chaves no *Border Router* ou noutra entidade responsável pela autenticação.

3.3 Concretização da solução

A normalização e os avanços tecnológicos têm ajudado a desenvolver o IoT, a disponibilidade de nós a baixo custo tem tido um papel fundamental nos avanços na pesquisa, desenvolvimento e a massificação do IoT. O aparecimento de novos sistemas operativos *open-source* como o Contiki[1] e o TinyOs[2] têm um papel determinante não apenas pela possibilidade da portabilidade do código desenvolvido, que facilita a interligação de sensores e microcontroladores de baixo custo, mas também com a possibilidade de partilha de código e soluções. Estas características tornam os sistemas operativos uma ferramenta bastante poderosa de desenvolvimento para dispositivos e cenários IoT [39].

Para a concretização da solução foram exploradas várias versões e aplicações do Contiki, de forma a determinar qual a mais adequada para desenvolver a solução proposta. Foram testadas as versões Contiki, mais antiga, e a Contiki-NG[3], mais recente, estruturada de forma diferente, facilitando o desenvolvimento.

Uma das várias aplicações é o CETIC 6LBR [40] , suportado por um grande número de plataformas de *hardware* comerciais, tais como a Zolertia, o telosB, o OpenWRT, entre outros. Este implementa segurança na camada 802.15.4 com a biblioteca LLSEC, implementa DTLS e tem uma interface gráfica para configurar o *Border Router* que utiliza 6LowPan e RPL com suporte a vários modos de funcionamento: standalone onde é necessário ter um dispositivo com interface para a internet e à rede 802.15.4, e existe ainda o modo Linux host, onde o *Border Router* está a correr num Linux, e apenas

[1] https://github.com/contiki-os/contiki
[2] https://github.com/tinyos
[3] https://github.com/contiki-ng/contiki-ng

necessita de um dispositivo com interface para a rede 802.15.4, que envia os dados, os pacotes da rede 802.15.4, via cabo série. Esta aplicação foi explorada para ajudar a perceber como a biblioteca de DTLS funcionava e como se podia adaptar às necessidades da solução proposta.

O *Sparrow* é outra aplicação do *Contiki*, foi desenhado para ser utilizada sobre UDP e especifica um modelo de objetos, sendo utilizado para a gestão de dispositivos IoT. Esta suporta várias plataformas, como Zolertia Re-Mote, Zolertia Firefly e Yanzi IoT-U10/42. À Semelhança do 6LBR tem vários modos de funcionamento e é parametrizável. Tem uma característica particular que é a implementação de um sistema de *OTA - Over The Air*. Este mecanismo permite que seja feito o *deployment* de código remotamente. Esta aplicação foi explorada com o objetivo de adquirir mais conhecimento sobre o sistema de configuração remota.

3.4 Operação

A solução proposta recorre ao uso de cinco entidades distintas, o nó sensor, o servidor ECDH, o *Bootstrap Server,* o servidor de gestão (MGNT Server) e o servidor de Leshan Figura 9 - Diagrama da solução proposta.

Os nós da rede têm como função ligarem-se à rede e conseguirem comunicar com o *Leshan* de uma forma segura. Possuem uma API Rest com métodos expostos com o objetivo fazer a sua gestão e de disponibilizar dados recolhidos pelos transdutores. O servidor ECDH tem como função fazer o acordo de chaves com os nós que se pretendem ligar à rede. É utilizado o *Diffie Hellman* para que os nós e o ECDH Server acordem uma chave comum. Essa chave é posteriormente utilziada para estabelecer uma ligação DTLS entre os nós e o *Bootstrap Server* no modo PSK. O ECDH Server é ainda responsável por configurar o *Boostrap Server* e o *Leshan* com as credenciais de acesso dos nós. As credenciais são compostas por *username* e *password*. O *Bootstrap Server* tem como função configurar os nós com as credenciais (username e passaword) e o endereço IP do servidor Leshan de uma forma dinâmica. O MGNT Server tem a lista de todos os nós autorizados na rede, tendo como função autenticar os nós na rede através de um token. O Leshan é a aplicação que permite interagir com a API dos nós.

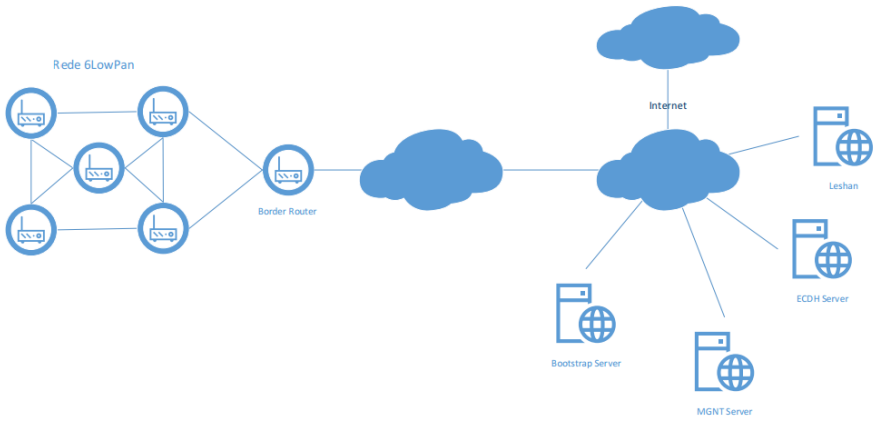


Figura 9 - Diagrama da solução proposta

O processo de admissão na rede pode ser dividido em cinco fases:

1. **Provisioning**

É no *provisioning* que os nós são configurados com o código desenvolvido onde existe um identificador único (ID) gerado aleatoriamente e secreto. ID é associado ao endereço MAC do dispositivo e é guardado no servidor de gestão para mais tarde autenticar os nós.

2. **Acordo de Chaves**

O nó liga-se ao ECDH Server e inica o processo acordo de chaves baseado no método de *Diffie Hellman* (DH). Este é necessário para que o nó e o ECDH Server cheguem a uma chave simétrica sobre um canal inseguro. O método de *Diffie Hellman* utilizado nesta solução baseia-se no uso de criptografia de curvas elípticas. A chave simétrica que resulta deste processo é posteriormente utilizada para que o nó estabeleça uma ligação segura, através de DTLS no modo PSK, entre o nó e *o Bootstrap Server*. No momento em que a chave é colocada no *Bootstrap Server* é ainda gerada a chave para a ligação entre o nó e o *Leshan* sendo esta colocada no *Boostrap Server* e no *Leshan*. Note-se que existe um canal de comunicação seguro entre o ECDH server, o *BootStrap Server* e o *Leshan*.

3. Autenticação

Nesta fase já existe um canal seguro de comunicação do nó com o *Bootstrap Server*. O nó envia um pedido ao *Bootstrap Server* para obter as configurações de acesso ao *Leshan*, nomeadamente o *username*, a *password* e o endereço IP do servidor. Nesta mensagem é enviado o endereço IP ou *endpoint* do servidor assim como o ID do nó para ser autenticado pelo MGNT server. Quando o *Bootstrap Server* recebe o pedido, envia para o MGNT Server o token, definido no provisioning, de forma a concluir a autenticação do nó. Se o ID enviado for válido então o nó vai receber as configurações para aceder à rede. Se o ID não for válido é enviada uma mensagem de erro para o nó, o endpoint é removido tanto do *Bootstrap Server* como do *Leshan* e o nó é retirado das tabelas de encaminhamento e vizinhança do *Border Router* de forma a impedir a comunicação com o nó.

4. Bootstraping

Esta fase apenas ocorre se o processo de autenticação for válido. O *Bootstrap Server* responde ao nó com as configurações de acesso ao *Leshan*, definidas e configuradas pelo ECDH Server. É enviado o endpoint do *Leshan* assim como as credenciais do nó. A chave definida de uma forma aleatória pode ser pode ser individual ou de grupo.

5. Comunicação

Nesta fase o nó já tem os dados de acesso ao *Leshan*. É iniciada uma ligação DTLS com as chaves enviadas pelo *Bootstrap Server*. Desta forma existe um canal de comunicação seguro entre os nós e o *Leshan* garantindo a integridade e confidencialidade das mensagens.

A exposição prolongada das chaves pode levar a que sejam comprometidas, logo existe a necessidade de uma troca de chaves de comunicação. Esta funcionalidade é conseguida através do envio de uma mensagem de reset ao nó.

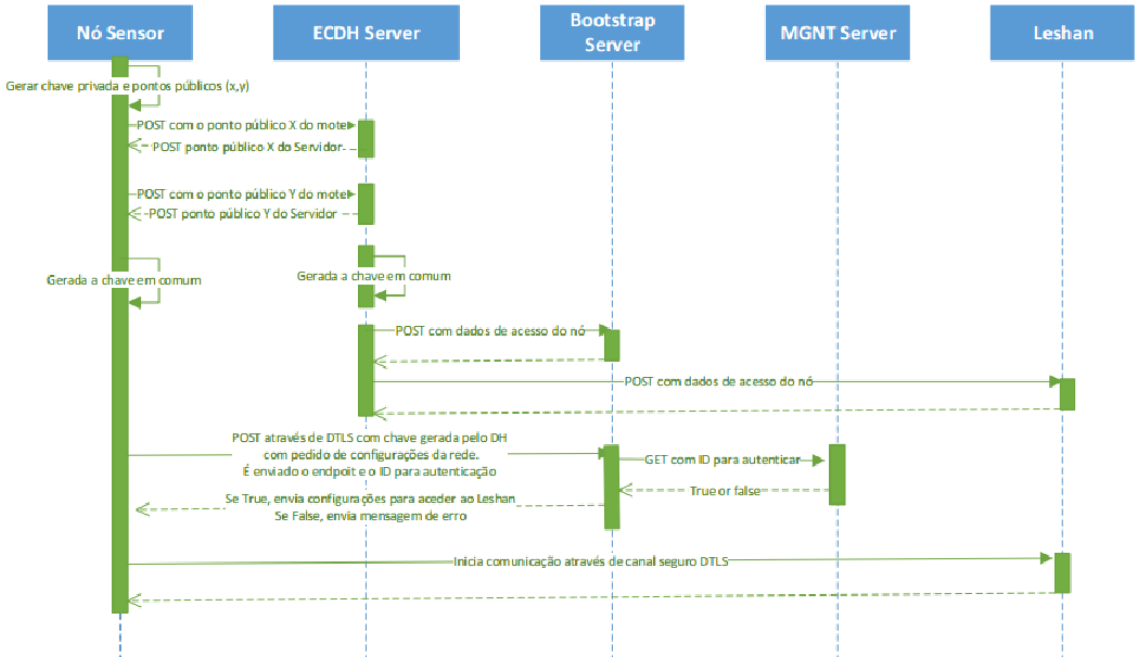


Figura 10 - Visão global das comunicações

3.5 Implementação

3.5.1 Rede 6LoWPAN

Todos os nós da rede usam o sistema operativo Contiki na versão Contiki-NG. Para o *Border Router* foi utilizado o rpl-border-router [41] , um exemplo presente na distribuição do Contiki-NG que foi adaptado. O rpl-border-router é composto por dois serviços, o de border-router que implementa a pilha protocolar de uma rede *6LowPan* e RPL, e o WebServer que disponibiliza uma página que contem informação sobre os nós ligados à rede. Por omissão a rede não suporta mecanismos de segurança na camada IEEE 802.15.4. O RPL é utilizado no modo lite [42], fazendo com que apenas o *Border Router* tenha conhecimento de toda a rede e os restantes nós apenas tenham uma rota para encaminhar os pacotes de forma a chegar ao *Border Router*.

```
#include "contiki.h"

/* Log configuration */
#include "sys/log.h"
#define LOG_MODULE "RPL BR"
#define LOG_LEVEL LOG_LEVEL_INFO

/* Declare and auto-start this file's process */
PROCESS(contiki_ng_br, "Contiki-NG Border Router");
AUTOSTART_PROCESSES(&contiki_ng_br);

/*---------------------------------------------------------------------------*/
PROCESS_THREAD(contiki_ng_br, ev, data)
{
  PROCESS_BEGIN();

#if BORDER_ROUTER_CONF_WEBSERVER
  PROCESS_NAME(webserver_nogui_process);
  process_start(&webserver_nogui_process, NULL);
#endif /* BORDER_ROUTER_CONF_WEBSERVER */

  LOG_INFO("Contiki-NG Border Router started\n");

  PROCESS_END();
}
```

Figura 11 - Border Router

Para os nós foi utilizado o exemplo lwm2m-ipso-objects adaptado. O protocolo OMA Lightweight M2M e IPSO Smart Objects (LWM2M) é um *standard* para IoT e dispositivos M2M que tem por base comunicações COAP e DTLS [43]. Este exemplo LWM2M foi escolhido pelos desenvolvimentos já efetuados pela comunidade como o *Bootstraping* inseguro e integração com aplicações *open source* como o *Leshan* e o *Bootstrap Server*. Contudo foram necessárias várias adaptações e alterações para implementar a solução apresentada.

O protocolo LWM2M implica um registo no servidor por parte dos nós, para que o servidor saiba quais os métodos que estão disponíveis. Para suportar este mecanismo o Contiki-NG tem uma máquina de estados, que também dá suporte ao processo de bootstraping (por um canal inseguro) se assim for o desejado, sendo necessária a ativação de uma flag de compilação Figura 12.

```
#ifndef REGISTER_WITH_LWM2M_BOOTSTRAP_SERVER
#define REGISTER_WITH_LWM2M_BOOTSTRAP_SERVER 1
#endif

#ifndef REGISTER_WITH_LWM2M_SERVER
#define REGISTER_WITH_LWM2M_SERVER 1
#endif
```

Figura 12 - Defines com a ativaçãao do processo de Bootstraping

O código disponibilizado está preparado para ligar o nó diretamente a um servidor DTLS como por exemplo o *Leshan* sendo assim necessário utilizar o método de obtenção de chaves *keystore simple* onde as chaves são definidas previamente no provisioning Figura 13.

```
#define COAP_DTLS_PSK_DEFAULT_IDENTITY "OurIdentity"
#define COAP_DTLS_PSK_DEFAULT_KEY      "OurSecret"
```

Figura 13 - Definição de chaves para utilizar o método keystore simple

Para suportar a arquitetura apresentada foi necessário, primeiro, adicionar estados à máquina de estados para suportar o acordo de chaves, segundo, alterar o código existente para permitir o bootstraping através de um canal seguro e configuração de um endereço seguro (COAPS), terceiro, adicionar os métodos para o servidor consumir.

- Novos estados para suportar o mecanismo de troca de chaves Diffie-Hellman (ECDH)

O Contiki-NG utiliza uma máquina de estados para controlar o bootstraping e o registo no servidor LWM2M. Para suportar o mecanismo era necessário adicionar alguns estados. Todos os estados são adicionados no início do processo de registo do nó, ou seja, é necessário a máquina começar no estado 20 ao invés do estado 0 Figura 14

```
#define INIT                0
#define WAIT_NETWORK        1
#define DO_BOOTSTRAP        3
#define BOOTSTRAP_SENT      4
#define BOOTSTRAP_DONE      5
#define DO_REGISTRATION     6
#define REGISTRATION_SENT   7
#define REGISTRATION_DONE   8
#define UPDATE_SENT         9
#define DEREGISTER         10
#define DEREGISTER_SENT    11
#define DEREGISTER_FAILED  12
#define DEREGISTERED       13
/*
* Estados ECDH
*/
#define INIT_ECDH                  20
#define SENDMESSAGE_ECDHX          21
#define SENDMESSAGE_ECDHY          22
#define VALIDATE_ECDH              23
#define WAITRESPONSE_ECDH          24
#define WAITCALCULUS_ECDH          25
```

Figura 14 - Estados Coniki-NG LWM2M

O estado inicial 20 - *INIT_ECDH* serve como porta de entrada na máquina de estados, e inicia o processo de criptografia Figura 15

```
case INIT_ECDH:
//Inicia logo o processo
process_start(&genSharedKey_ecdh, NULL);

LOG_DBG("RD Client started with endpoint '%s' and client lifetime %d\n", session_info.ep, session_info.lifetime);
rd_state = SENDMESSAGE_ECDHX;
break;
```

Figura 15 - Estado INIT_ECDH

O processo de criptografia foi adaptado de um exemplo já existente no Contiki-NG ecc-ecdh.c, que tem como objetivo demonstrar como no Contiki é possível utilizar criptografia de chave assimétrica na forma de curvas elípticas. Este exemplo gera duas chaves privadas aleatoriamente, uma entidade A e outra entidade B geram os seus pontos públicos, simulam a troca de chaves, e efetuam a derivação com base na sua chave privada e nas chaves públicas da outra entidade. No fim as duas chaves são comparadas. Este exemplo foi adaptado para gerar apenas uma chave privada e as chaves públicas correspondentes, com base na curva NIST-192. Depois é necessário esperar a recepção dos pontos públicos Figura 16.

```
PROCESS(genSharedKey_ecdh, "Generate ECDH");
PROCESS_THREAD(genSharedKey_ecdh, ev, data)
{
  PROCESS_BEGIN();
  LOG_DBG(" PROCESSO genSharedKey_ecdh \n");
  pka_init();

  static ecc_compare_state_t state = {
    .process = &genSharedKey_ecdh,
    .size    = 6,
  };

  memcpy(state.b, nist_p_192.n, sizeof(uint32_t) * 6);
  static uint32_t secret_a[6];
  do {
    ecc_set_random(secret_a);

    memcpy(state.a, secret_a, sizeof(uint32_t) * 6);
    PT_SPAWN(&(genSharedKey_ecdh.pt), &(state.pt), ecc_compare(&state));
  } while(state.result != PKA_STATUS_A_LT_B);


  static ecc_multiply_state_t side_a = {
    .process    = &genSharedKey_ecdh,
    .curve_info = &nist_p_192,
  };
  memcpy(side_a.point_in.x, nist_p_192.x, sizeof(uint32_t) * 6);
  memcpy(side_a.point_in.y, nist_p_192.y, sizeof(uint32_t) * 6);
  memcpy(side_a.secret, secret_a, sizeof(secret_a));

  //Gera chaves publicas
  PT_SPAWN(&(genSharedKey_ecdh.pt), &(side_a.pt), ecc_multiply(&side_a));

  // //define a pontos da chave publica para enviar
  memcpy(MyPubKeyX, side_a.point_out.x, sizeof(uint32_t) * 6);
  memcpy(MyPubKeyY, side_a.point_out.y, sizeof(uint32_t) * 6);

  //Espera pela resposta do ecdhServer
  PROCESS_WAIT_EVENT();
```

Figura 16 - Inicio do processo criptografico.

A máquina de estados avançou para o estado *SENDMESSAGE_ECDHX* para enviar o ponto X para o ECDH Server. Os pontos X e Y são enviados separadamente devido ao tamanho máximo dos pacotes definido para mensagens deste tipo. O ponto X do nó é enviado através de uma mensagem COAP para o ECDH Server com *endpoint* /ecdh/pubx para que o server saiba que está a receber o ponto X e envia na querystring o seu endereço MAC. Como se espera uma resposta é definido um callback (ECDHGetSetX_callback) que tem como objetivo receber o ponto X da outra entidade e fazer o seu tratamento para ficar no formato das estruturas utilizadas para o processo criptográfico.

```
static bool
ECDHGetSetX()
{
  LOG_DBG(" INIT ECDHGetSetX \n");

  coap_endpoint_t ECDHserver_ep;

  if(coap_endpoint_parse(ECDH_SERVER_ADDRESS, strlen(ECDH_SERVER_ADDRESS), &ECDHserver_ep) != 0)
  {
    coap_init_message(request, COAP_TYPE_CON, COAP_POST, 0);
    coap_set_header_uri_path(request, "/ecdh/pubx");
    //Envia Chaves
    char PubX[64];
    getMyPubKeyX(PubX);
    snprintf(query_data, sizeof(query_data) - 1, "?ep=%s", session_info.ep);
    //O payload tem o Ponto X
    coap_set_payload(request, (uint8_t *)PubX, sizeof(PubX) - 1);

     coap_set_header_uri_query(request, query_data);
     LOG_INFO("Send Pub X - ECDH[");
     LOG_INFO_COAP_EP(&ECDHserver_ep);
     LOG_INFO_("] as '%s'\n", query_data);

    if(coap_send_request(&rd_request_state, &ECDHserver_ep, request, ECDHGetSetX_callback)) {
        rd_state = WAITRESPONSE_ECDH;;
        return true;
    }
  }
  return false;
}
```

Figura 17 - Envio do ponto X do nó

```
static void
ECDHGetSetX_callback(coap_callback_request_state_t *callback_state)
{
  const char *Xpoint = NULL;
  size_t len = 0;
  coap_request_state_t *state = &callback_state->state;
  LOG_DBG("ECDH callback Response: %d, \n", state->response != NULL);
  if(state->status == COAP_REQUEST_STATUS_RESPONSE) {

    coap_message_t *request = state->response;

    if((len = coap_get_post_variable(request, "Xpoint", &Xpoint))) {
      LOG_DBG("Xpoint %.*s\n", len, Xpoint);
    }

    char auxx[strlen(Xpoint)];
    strcpy(auxx, Xpoint);
    ParseStringToUInt32X(auxx);

    rd_state = VALIDATE_ECDH;

  }else if(state->status == COAP_REQUEST_STATUS_TIMEOUT) {
    LOG_DBG_("Server not responding! Retry?");
    rd_state = VALIDATE_ECDH;
  }
}
```

Figura 18 - Callback para receber ponto X do ECDH Server

Quando é recebida a resposta do servidor passa para o estado *VALIDATE_ECDH*, para fazer o mesmo processo relativamente o ponto Y. Este estado foi criado para validar o processo, desta forma o nó avança de estado quando já tem os dois pontos públicos do

servidor. Quando as chaves já estão configuradas é feito um POST ao processo criptográfico para que este calcule a chave em comum, ficando no estado *WAITCALCULUS_ECDH* Figura 19.

```
case VALIDATE_ECDH:
  LOG_DBG("VALIDATE_ECDH \n");
  printKey192(brPubKeyX);
  printKey192(brPubKeyY);

  if(brPubKeyX[0] == 0){
    rd_state = SENDMESSAGE_ECDHX;
  }else if (brPubKeyY[0] == 0){
     rd_state = SENDMESSAGE_ECDHY;
  }else{
    //Contitua o processo
    process_post_synch(&genSharedKey_ecdh, PROCESS_EVENT_CONTINUE, NULL);
    rd_state=WAITCALCULUS_ECDH;
    //Agora sim, está tudo validado vais fazer o bootstrap;
    //rd_state = DO_BOOTSTRAP;
  }
  break;
```

Figura 19 - Estado de validação de receção dos pontos do servidor

Já com as chaves trocadas, o processo faz a derivação para a chave comum, e já com a chave partilhada encontrada é gerada uma chave mais pequena com base na chave partilhada. Esta chave vai ser utilizada como chave na ligação com o *Bootrstap Server*, e finalmente segue para os estados que fazem *bootstraping* Figura 20.

```
//Espera pela resposta do ecdhServer
PROCESS_WAIT_EVENT();

//Já quando as cahves estão trocadas
memcpy(side_a.point_in.x, brPubKeyX, sizeof(uint32_t) * 6);
memcpy(side_a.point_in.y, brPubKeyY, sizeof(uint32_t) * 6);

//Generates Shared Key
PT_SPAWN(&(genSharedKey_ecdh.pt), &(side_a.pt), ecc_multiply(&side_a));

printf("Shared Key  \n");
printKey192(side_a.point_out.x);
printKey192(side_a.point_out.y);

uint8_t key[8];
//GRAVA A CHAVE SIMETRICA
memcpy(key , &side_a.point_out.x, sizeof(key));
//Key to Connect with BS server
GetKeyuint8(MyBSKey,key);
memcpy(MyBSKeyPUB , &MyBSKey, sizeof(MyBSKey));
//inicia o processo de Bootstrap
rd_state=DO_BOOTSTRAP;

pka_disable();

PROCESS_END();
```

Figura 20 - Final do processo criptográfico e geração da chave para comunicar com o BSServer.

- Alterações para permitir o bootstraping seguro

Para permitir um *bootstraping seguro* é necessário garantir uma ligação DTLS com o *Bootstrap Server*. A chave utilizada é a resultante do processo anterior de *Diffie-Hellman*. O Contiki suporta dois modos distintos, que funcionam de uma forma exclusiva (funciona de um modo ou de outro) para armazenar chaves para ligações DTLS no modo PSK. O armazenamento simples, onde a chave é definida no *provisioning*, ou com *bootstrap* onde a chave é recebida no processo de *bootstraping* inseguro e é criada uma instância de segurança de LWM2M onde as chaves ficam armazenadas.

De modo a implementar a solução apresentada é necessário que os dois modos de armazenamento funcionem simultaneamente, onde um deles, neste caso do armazenamento simples, seja adaptado de forma a suportar alterações em runtime para que a chave a ser utilizada seja a resultante do *Diffie-Hellman* Figura 21.

```
case DTLS_PSK_KEY:
  if(dtls_keystore->coap_get_psk_info) {
    ks.identity = id;
    ks.identity_len = id_len;
    /* we know that session is a coap endpoint */
    dtls_keystore->coap_get_psk_info((coap_endpoint_t *)session, &ks);
  }

  if(ks.key == NULL || ks.key_len == 0) {
    char MyBSKeyPUB1[16];

    memcpy(MyBSKeyPUB1 , &MyBSKeyPUB, sizeof(MyBSKeyPUB));

    LOG_DBG("pkChar : %s\n", MyBSKeyPUB);
    LOG_DBG("pk : %s\n", MyBSKeyPUB1);

    ks.key = (uint8_t *)MyBSKeyPUB1;
    ks.key_len = strlen(MyBSKeyPUB1);
  }

  if(result_length < ks.key_len) {
    LOG_DBG("cannot return psk -- buffer too small\n");
    return dtls_alert_fatal_create(DTLS_ALERT_INTERNAL_ERROR);
  }
  memcpy(result, ks.key, ks.key_len);
```

Figura 21 - Método de utilização da chave

Como o processo de *bootstrap* seguro não era suportado existia uma validação ao receber as configurações para validar se o endereço enviado era COAPS, se sim retornava um erro e abortava o processo de configuração, logo era necessário remove-la.

```
LOG_DBG("**** Found security instance using: ");
LOG_DBG_COAP_STRING((const char *)security->server_uri,
                    security->server_uri_len);
LOG_DBG_(" (len %d) \n", security->server_uri_len);
/* TODO Should verify it is a URI */
/* Check if secure */
//PF - this way we can connect to BS with DTLS.
secure = 0;// strncmp((const char *)security->server_uri,
    //        "coaps:", 6) == 0;

if(!coap_endpoint_parse((const char *)security->server_uri,
                        security->server_uri_len,
                        &session_info.server_ep)) {
  LOG_DBG("Failed to parse server URI!\n");
} else {
  LOG_DBG("Server address:");
  LOG_DBG_COAP_EP(&session_info.server_ep);
  LOG_DBG_("\n");
  if(secure) {
    LOG_DBG("Secure CoAP requested but not supported - can not bootstrap\n");
  } else {
    lwm2m_rd_client_register_with_server(&session_info.server_ep);
    session_info.bootstrapped++;
  }
}
```

Figura 22 - Remoção da validação para endereços seguros.

Para que a ligação com o *Leshan* fosse conseguida era necessário desconectar a ligação existente com o *Bootstrap Server*. Desta forma no estado onde é feito o registo no *Leshan* é desconectada a ligação com o *Bootstrap Server* e efetuada a ligação com o *Leshan*.

```
case DO_REGISTRATION:
  // PF -Disconect do Bs server
  coap_endpoint_disconnect(&session_info.bs_server_ep);


  if(!coap_endpoint_is_connected(&session_info.server_ep)) {
    /* Not connected... wait a bit... and retry connection */
    coap_endpoint_connect(&session_info.server_ep);
    LOG_DBG("Wait until connected... \n");
    return;
  }

  if(session_info.use_registration && !session_info.registered &&
     update_registration_server()) {
    int len;
```

Figura 23 - Ligação DTLS com Leshan.

3.5.2 Provisioning dos nós

O provisioning é feito manualmente, sendo necessário gerar os tokens e colocalos tanto no nó como no MGNT Server associados ao endereço MAC.

3.5.3 ECDH Server

O ECDH Server foi criado em Node.JS e tem como objectivo dar suporte ao mecanismo de ECDH. A troca de mensagens é feita através de COAP logo é necessário implementar um servidor para responder às mensagens dos nós. São utilizadas as bibliotecas de COAP para comunicação e elliptic para o mecanismo criptográfico.

Quando o servidor é iniciado são gerados os pontos públicos Figura 24, é também criado um array associativo para guardar as informações relativas à troca de chaves.

```
var OwnKeyPair = ec.genKeyPair();
var OwnX = OwnKeyPair.getPublic().getX().toString(16).toUpperCase();
var OwnY = OwnKeyPair.getPublic().getY().toString(16).toUpperCase();
```

Figura 24 - Gerar chaves ECDH Server.

Ao receber um POST do nó com o seu ponto público X, o ECDH Server envia o seu ponto X, o mesmo acontece com o ponto Y Figura 25 - Resposta com os pontos públicos do servidor.

```
if(path == '/ecdh/puby'){
    item.pointY=response.split(';')[0];
    res.end('Ypoint='+OwnY+'\n');
    res.reset();
}else if(path == '/ecdh/pubx'){
    item.pointX=response.split(';')[0];
    res.end('Xpoint='+OwnX+'\n');
    res.reset();
}else{
```

Figura 25 - Resposta com os pontos públicos do servidor

No final do pedido, existe uma validação para determinar se o servidor já tem os dois pontos que correspondem ao nó Figura 27. Quando o servidor já tem os dois pontos, então gera a chave partilhada para a ligação com o *Bootstrap Server*, gera outra chave (esta chave pode ser de grupo ou individual) para o comunicar com o *Leshan* Figura

26. É ainda feita a configuração do *Boostrap Server* e do *Leshan* com o novo nó e seus dados de acesso Figura 27.

```
function SetSharedKey(item) {

    var pubB = {
        x: item.pointX.toString('hex'),
        y: item.pointY.toString('hex')
    }; // case 2
    var key2 = ec.keyFromPublic(pubB);

    var shared1 = OwnKeyPair.derive(key2.getPublic());

    //gera chave partilhada
    var shared1 = OwnKeyPair.derive(key2.getPublic());
    console.log("Partilhada " + shared1.toString(16));

    shared1 = pad(shared1.toString(16), 48, '0')

    var res = shared1.toString(16).match(/.{2}/g);
    var final = ""
    res.reverse().forEach(function (entry, idx, array) {
        if (idx < 8)
            final += entry;
    });
    return final;
}
```

Figura 26 - Derivação da chave partilhada.

```
//actuliza hash
hash.setItem(ep, item);

if (item.pointX != '' && item.pointY != '') {
    console.log('Configura  ');
    let BSServerSharedkey = SetSharedKey(item).toUpperCase();;
    let LeshanSharedkey = makeid();
    httpHandler.DeleteConfigurationBSServer(ep, BSServerSharedkey, LeshanSharedkey);
    httpHandler.DeleteConfigurationLeshanServer(ep, LeshanSharedkey);
    hash.removeItem(ep);
}
```

Figura 27 - Validações e Configuração do Bootstrap Server e Leshan

3.5.4 Bootstrap Server

O *Bootstrap Server* é um servidor/cliente LWM2M e DTLS que faz parte do projeto Eclipse Leshan, uma framework em Java *opensource*. A solução está implementada numa arquitetura cliente-servidor onde existe uma API com métodos expostos para

consumir e uma página Web que consome alguns dos métodos da API. A Figura 28 mostra os clientes configurados para bootstraping, onde na coluna LWM2M ***Bootstrap Server*** estão as configurações para o nó se ligar ao *Bootstrap Server* e na coluna ***LWM2M Server*** estão as configurações que são enviadas para o nó por forma a ligar-se ao servidor LWM2M.

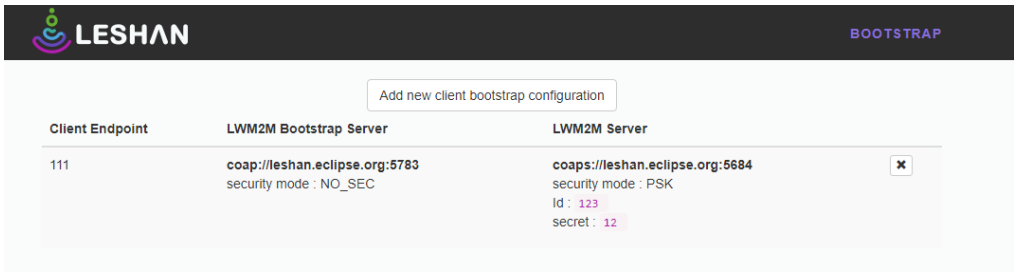


Figura 28 - Boostrap Server - Lista de clientes configurados.

Para suportar a solução apresentada anteriormente foram necessárias alterações na aplicação. A solução está implementada em Java que utiliza Maven para a gestão do projeto. Um nó para ser autenticado tem de enviar o seu identificador único, definido no provisioning, no pedido ao *Bootstrap Server* Figura 29, este, por sua vez, comunica com o MNGM Server para autenticar o nó Figura 30 - Quando é recebido um pedido para configurar um nó.. Se a resposta for positiva então é feito o processo de *Bootstrap* normal ou seja as configurações são enviadas para o nó. Se a resposta for negativa então é enviada uma mensagem de erro ao nó, e é removido das rotas do *Border Router*.

```
private boolean sendGet(String endpoint, String secAuth) throws Exception {

    String url = "http://localhost:3000/mngm?ep="+endpoint+"&secId=" + secAuth;

    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    con.setRequestMethod("GET");

    int responseCode = con.getResponseCode();
    System.out.println("\nSending 'GET' request to URL : " + url);
    System.out.println("Response Code : " + responseCode);

    BufferedReader in = new BufferedReader(
            new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();

    //print result
    System.out.println(response.toString());
    if(response.toString().equals("true") ) {
        return true;
    }else {
        return false;
    }
}
```

Figura 29 - Get ao MNGM Server

```
public BootstrapResponse bootstrap(Identity sender, BootstrapRequest request) {
    String endpoint = request.getEndpointName();
    String secAuth = request.getSecAuthName();

    // Start session, checking the BS credentials
    final BootstrapSession session = this.sessionManager.begin(endpoint,secAuth, sender);

    if (!session.isAuthorized()) {
        this.sessionManager.failed(session, UNAUTHORIZED, null);
        return BootstrapResponse.badRequest("Unauthorized");
    }

    boolean isAuth = false;
    try {
            isAuth = sendGet(endpoint,secAuth);
            } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }

    if(!isAuth) {
            LOG.debug("No Auth for {}", endpoint, secAuth);
        this.sessionManager.failed(session, UNAUTHORIZED, null);
            return BootstrapResponse.badRequest("Unauthorized SecAuth");
    }
```

Figura 30 - Quando é recebido um pedido para configurar um nó.

3.5.5 Leshan

O Leshan é um servidor LWM2M e DTLS que faz parte do projeto Eclipse Leshan uma framework Java *opensource*. A solução está implementada numa arquitetura cliente-servidor onde existe uma API com métodos expostos para consumir, e uma página Web que consome alguns dos métodos da API. Na Figura 31 é possível ver os clientes ligados ao servidor, e na Figura 32 os métodos disponíveis de um nó. Para suportar a solução apresentada anteriormente não foram necessárias alterações na aplicação. É através do *Leshan* que é possivel interagir com os nós.

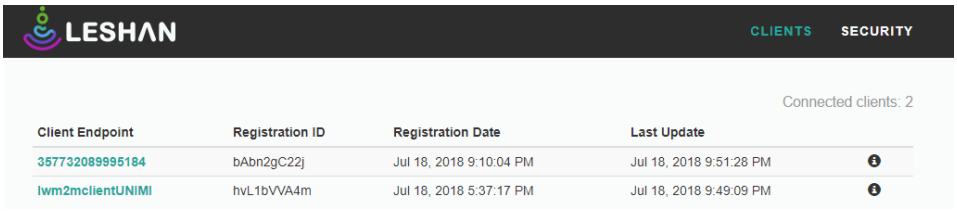


Figura 31 - Leshan - Lista de clientes

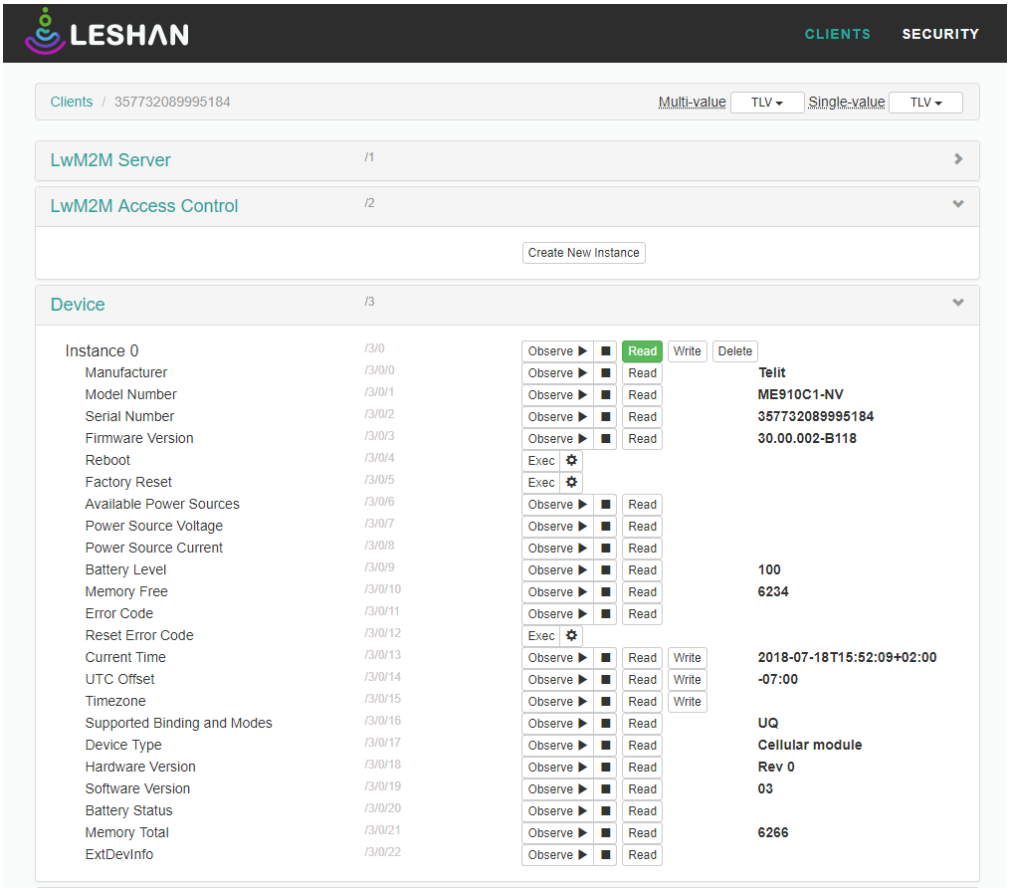


Figura 32 - Leshan - Métodos disponíveis de um nó.

4 Resultados

De forma validar a solução proposta foi configurada uma testbed que consiste num raspberry onde são suportados o ECDH Server, o Bootstrap Server, o MGNT Server e o Leshan, e o Border Router. Para os nós da rede foram utilizados os dispositivos Zolertia RE-Mote com o sistema operativo Contiki-NG v4. Foram utilizados três tipos de *software* para os nós, um para *Border Router* que serve de interface para a rede 802.15.4, um *Sniffer* para capturar os pacotes 802.15.4 e outro com a solução desenvolvida para os nós sensores da rede. Existem ainda nós válidos e não válidos, os nós válidos são os que têm um token válido no MGNT Server para se conectarem à rede, os nós inválidos não têm um token válido no MGNT Server, logo não se podem ligar à rede.

Foi definida uma bateria de testes com o objectivo de validar a solução proposta. Os testes têm como base a adição consecutiva de três nós diferentes:

i) Adicionar à rede um nó desconhecido e válido;
ii) Adicionar à rede um nó conhecido inválido;
iii) Adicionar à rede um nó conhecido e válido.

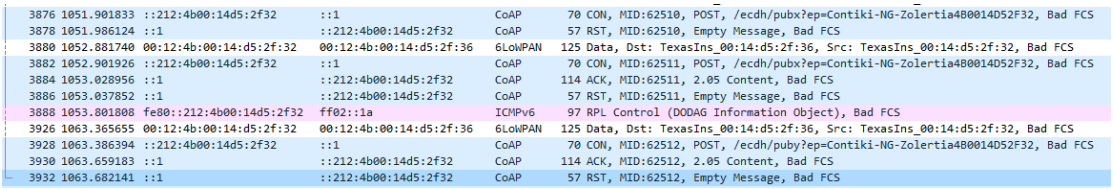
Na fase inicial todos os nós, conhecidos ou desconhecidos, válidos ou não válidos seguem o mesmo fluxo, desta forma todos os cenários de teste descritos abaixo têm em comum a fase inicial. Quando um nó se junta à rede comunica com o ECDH Server para iniciar o processo de acordo de chaves. Este processo é seguido pelo handshake necessário para se ligar por DTLS ao *Bootstrap Server*. Apenas nesta fase do processo é visível a informação transportada nos pacotes capturados. As comunicações subsequentes são cifradas. É depois da comunicação com o *Bootstrap Server* que um nó é rejeitado ou aceite.

A figura Figura 33 representa a troca de pacotes realizada entre o nó *"00:12:4b:00:06:0d:61:0d"* e o *Border Router*.

No.	Time	Source	Destination	Protocol	Lengt	Info
3747	1023.712632	fe80::212:4b00:14d5:2f32	ff02::1a	ICMPv6	27	RPL Control (DODAG Information Solicitation), Bad FCS
3767	1030.764092	fe80::212:4b00:14d5:2f32	fe80::212:4b00:14d5:2f36	ICMPv6	102	RPL Control (DODAG Information Object), Bad FCS
3787	1032.100202	fe80::212:4b00:14d5:2f32	fe80::212:4b00:14d5:2f36	ICMPv6	102	RPL Control (DODAG Information Object), Bad FCS
3789	1034.419177	fe80::212:4b00:14d5:2f32	fe80::212:4b00:14d5:2f36	ICMPv6	102	RPL Control (DODAG Information Object), Bad FCS
3810	1038.419126	fe80::212:4b00:14d5:2f32	fe80::212:4b00:14d5:2f36	ICMPv6	102	RPL Control (DODAG Information Object), Bad FCS
3812	1039.249572	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3814	1039.273621	::212:4b00:14d5:2f32	::1	CoAP	70	CON, MID:62510, POST, /ecdh/pubx?ep=Contiki-NG-Zolertia4B0014D52F32, Bad FCS
3833	1043.074708	::212:4b00:14d5:2f32	::212:4b00:14d5:2f36	ICMPv6	85	RPL Control (Destination Advertisement Object), Bad FCS
3835	1043.190444	::212:4b00:14d5:2f36	::212:4b00:14d5:2f32	ICMPv6	43	RPL Control (Destination Advertisement Object Acknowledgment), Bad FCS
3846	1046.480847	fe80::212:4b00:14d5:2f32	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object), Bad FCS
3874	1051.874080	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS

Figura 33 - Captura de pacotes Wireshark com ligação do nó à rede.

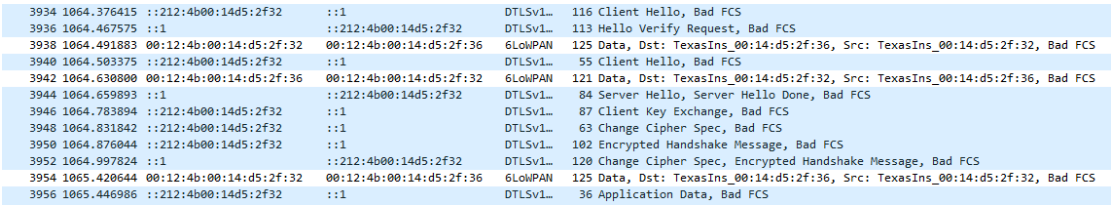
Nesta fase é possivel ver a troca de chaves entre o nó e o ECDH Server. Esta é iniciada no pacote 3882 Figura 34, onde o nó faz um POST com o ponto X da sua chave pública para o método "/ecdh/pubx", indicando na querystring "?ep" o seu identificador único ("Nome do mote" + MAC Address). O ECDH Server responde no pacote 3884 Figura 34, com o ponto X da sua chave pública. É possível observar o mesmo comportamento para o envio do ponto Y para o método "/ecdh/pub" no pacote 3928 e a resposta do ECDH Server no 3930. Os pontos públicos são uma string com 64 caracteres hexadecimais em maiúsculas .



3876	1051.901833	::212:4b00:14d5:2f32	::1	CoAP	70	CON, MID:62510, POST, /ecdh/pubx?ep=Contiki-NG-Zolertia4B0014D52F32, Bad FCS
3878	1051.986124	::1	::212:4b00:14d5:2f32	CoAP	57	RST, MID:62510, Empty Message, Bad FCS
3880	1052.881740	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3882	1052.901926	::212:4b00:14d5:2f32	::1	CoAP	70	CON, MID:62511, POST, /ecdh/pubx?ep=Contiki-NG-Zolertia4B0014D52F32, Bad FCS
3884	1053.028956	::1	::212:4b00:14d5:2f32	CoAP	114	ACK, MID:62511, 2.05 Content, Bad FCS
3886	1053.037852	::1	::212:4b00:14d5:2f32	CoAP	57	RST, MID:62511, Empty Message, Bad FCS
3888	1053.801808	fe80::212:4b00:14d5:2f32	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object), Bad FCS
3926	1063.365655	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3928	1063.386394	::212:4b00:14d5:2f32	::1	CoAP	70	CON, MID:62512, POST, /ecdh/puby?ep=Contiki-NG-Zolertia4B0014D52F32, Bad FCS
3930	1063.659183	::1	::212:4b00:14d5:2f32	CoAP	114	ACK, MID:62512, 2.05 Content, Bad FCS
3932	1063.682141	::1	::212:4b00:14d5:2f32	CoAP	57	RST, MID:62512, Empty Message, Bad FCS

Figura 34 - Captura de pacotes Wireshark processo de ligação ao ECDH Server

Nesta fase é possivel ver a troca de mensagens entre o nó e o Bootstrap Server de forma a efetuar o Handshake, para estabelecer a ligação DTLS. As mensagens são descritas na secção 2.4



3934	1064.376415	::212:4b00:14d5:2f32	::1	DTLSv1…	116	Client Hello, Bad FCS
3936	1064.467575	::1	::212:4b00:14d5:2f32	DTLSv1…	113	Hello Verify Request, Bad FCS
3938	1064.491883	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3940	1064.503375	::212:4b00:14d5:2f32	::1	DTLSv1…	55	Client Hello, Bad FCS
3942	1064.630800	00:12:4b:00:14:d5:2f:36	00:12:4b:00:14:d5:2f:32	6LoWPAN	121	Data, Dst: TexasIns_00:14:d5:2f:32, Src: TexasIns_00:14:d5:2f:36, Bad FCS
3944	1064.659893	::1	::212:4b00:14d5:2f32	DTLSv1…	84	Server Hello, Server Hello Done, Bad FCS
3946	1064.783894	::212:4b00:14d5:2f32	::1	DTLSv1…	87	Client Key Exchange, Bad FCS
3948	1064.831842	::212:4b00:14d5:2f32	::1	DTLSv1…	63	Change Cipher Spec, Bad FCS
3950	1064.876044	::212:4b00:14d5:2f32	::1	DTLSv1…	102	Encrypted Handshake Message, Bad FCS
3952	1064.997824	::1	::212:4b00:14d5:2f32	DTLSv1…	120	Change Cipher Spec, Encrypted Handshake Message, Bad FCS
3954	1065.420644	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125	Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3956	1065.446986	::212:4b00:14d5:2f32	::1	DTLSv1…	36	Application Data, Bad FCS

Figura 35- Captura de pacotes Wireshark processo de handshake com o Bootstrap Server

Todas as mensagens seguintes são cifradas e não é possivel ver o seu conteúdo no wireshark.

Cenários de teste

Depois de ser estabelecida uma ligação segura entre o nó e o *Bootstrap Server* é então realizada a autenticação. É nesta fase que o nó envia o seu **token** para ser autenticado, é também nesta fase que o processo pode seguir dois caminhos, pois o nó pode ser recusado e excluído da rede ou ser aceite e continuar na rede.

1. Adicionar à rede um nó desconhecido e não válido

Para a concretização deste cenário foi utilizado o nó com o endereço MAC *"00:12:4b:00:06:0d:61:0d"*, é necessário que a chave de autenticação do nó **não** estar presente no servidor de gestão, de modo a que seja rejeitado. O processo segue o fluxo descrito na Figura 36 **,** onde no final é retornado "False".

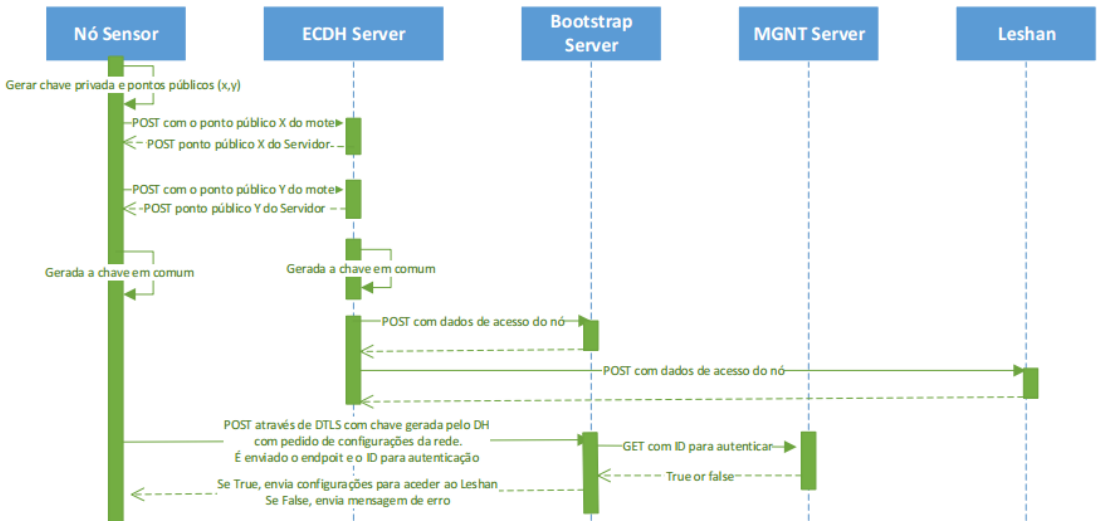


Figura 36 - Esquema de comunicações de um nó inválido (O retorno do MGNT Server é false).

Na captura wireshark com o filtro *"wpan.src64 == 00:12:4b:00:06:0d:61:0d || wpan.dst64 == 00:12:4b:00:06:0d:61:0d"*, é possível comprovar que apenas existe um pacote com a descrição "Application Data" Figura 37, o que significa que o nó não se juntou à rede por ter sido rejeitado.

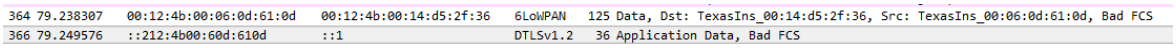


Figura 37 – Captura de pacotes Wireshark com pacotes cifrados.

2. Adicionar à rede um nó conhecido inválido

Para a concretização deste cenário foi utilizado o nó com o endereço MAC *"00:12:4b:00:11:f4:eb:50"* e, à semelhança do cenário acima descrito acima, a chave do nó **não** pode estar no servidor de gestão. O processo segue o fluxo descrito na Figura 36**,** onde no final é retornado "False".

Na captura wireshark foi utilizado o filtro *"wpan.src64 == 00:12:4b:00:11:f4:eb:50 || wpan.dst64 == 00:12:4b:00:11:f4:eb:50"*, é possível ver que apenas existem quatro

pacotes com a descrição “Application Data”, isto quer dizer que o nó não se juntou à rede e foi rejeitado Figura 38.

3398	934.794819	::212:4b00:11f4:eb50	::1	DTLSv1.2	36 Application Data, Bad FCS
3416	939.088444	00:12:4b:00:11:f4:eb:50	00:12:4b:00:14:d5:2f:36	6LoWPAN	125 Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:11:f4:eb:50, Bad FCS
3418	939.095391	::212:4b00:11f4:eb50	::1	DTLSv1.2	36 Application Data, Bad FCS
3438	944.508542	::212:4b00:11f4:eb50	::212:4b00:14d5:2f36	ICMPv6	85 RPL Control (Destination Advertisement Object), Bad FCS
3459	947.720984	00:12:4b:00:11:f4:eb:50	00:12:4b:00:14:d5:2f:36	6LoWPAN	125 Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:11:f4:eb:50, Bad FCS
3461	947.728404	::212:4b00:11f4:eb50	::1	DTLSv1.2	36 Application Data, Bad FCS
3463	949.054678	::212:4b00:11f4:eb50	::212:4b00:14d5:2f36	ICMPv6	85 RPL Control (Destination Advertisement Object), Bad FCS
3491	956.156560	::212:4b00:11f4:eb50	::212:4b00:14d5:2f36	ICMPv6	85 RPL Control (Destination Advertisement Object), Bad FCS
3523	961.936525	::212:4b00:11f4:eb50	::212:4b00:14d5:2f36	ICMPv6	85 RPL Control (Destination Advertisement Object), Bad FCS
3526	964.971678	00:12:4b:00:11:f4:eb:50	00:12:4b:00:14:d5:2f:36	6LoWPAN	125 Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:11:f4:eb:50, Bad FCS
3528	964.989397	::212:4b00:11f4:eb50	::1	DTLSv1.2	36 Application Data, Bad FCS
3548	968.765000	::212:4b00:11f4:eb50	::212:4b00:14d5:2f36	ICMPv6	85 RPL Control (Destination Advertisement Object), Bad FCS

Figura 38 - Captura de pacotes Wireshark de um nó rejeitado

3. Adicionr à rede um nó conhecido e válido

Para a concretização deste cenário foi utilizado o nó com o endereço MAC *“00:12:4b:00:14:d5:2f:32”* sendo necessário que a chave de autenticação do nó **estar** presente no servidor de gestão. O processo segue o fluxo normal descrito na Figura 39.

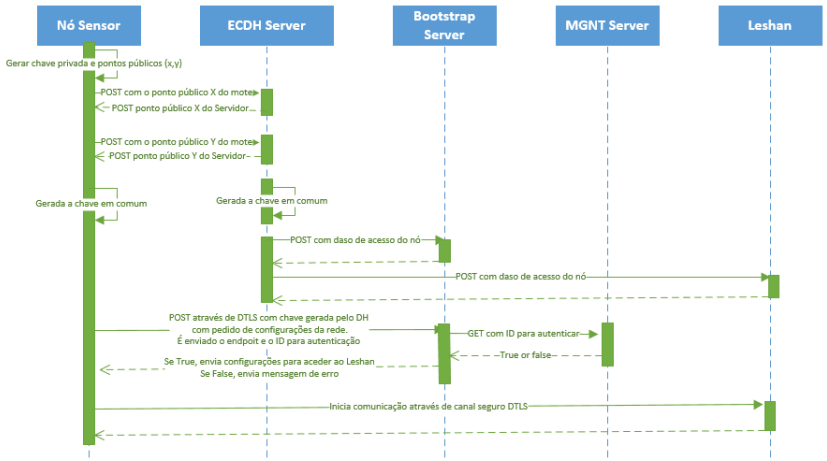


Figura 39 - Esquema de comunicações de um nó válido (O retorno do MGNT Server é true).

É ainda possível ver na captura wireshark, utilizando o filtro *“wpan.src64 == 00:12:4b:00:14:d5:2f:32 || wpan.dst64 == 00:12:4b:00:14:d5:2f:32”* que existem múltiplos pacotes com a descrição “Application Data”, isto quer dizer que o nó não foi rejeitado e se juntou à rede ao contrário dos anteriores Figura 40.

3952	1064.997824	::1	::212:4b00:14d5:2f32	DTLSv1.2	120 Change Cipher Spec, Encrypted Handshake Message, Bad FCS
3954	1065.420644	00:12:4b:00:14:d5:2f:32	00:12:4b:00:14:d5:2f:36	6LoWPAN	125 Data, Dst: TexasIns_00:14:d5:2f:36, Src: TexasIns_00:14:d5:2f:32, Bad FCS
3956	1065.446986	::212:4b00:14d5:2f32	::1	DTLSv1.2	36 Application Data, Bad FCS
3958	1065.582437	::1	::212:4b00:14d5:2f32	DTLSv1.2	86 Application Data, Bad FCS
3960	1065.652555	::1	::212:4b00:14d5:2f32	DTLSv1.2	94 Application Data, Bad FCS
3962	1065.690912	::212:4b00:14d5:2f32	::1	DTLSv1.2	90 Application Data, Bad FCS
3964	1065.880906	00:12:4b:00:14:d5:2f:36	00:12:4b:00:14:d5:2f:32	6LoWPAN	121 Data, Dst: TexasIns_00:14:d5:2f:32, Src: TexasIns_00:14:d5:2f:36, Bad FCS
3966	1065.891154	::1	::212:4b00:14d5:2f32	DTLSv1.2	92 Application Data, Bad FCS
3968	1066.020035	::212:4b00:14d5:2f32	::1	DTLSv1.2	90 Application Data, Bad FCS
3970	1066.130827	00:12:4b:00:14:d5:2f:36	00:12:4b:00:14:d5:2f:32	6LoWPAN	121 Data, Dst: TexasIns_00:14:d5:2f:32, Src: TexasIns_00:14:d5:2f:36, Bad FCS

Figura 40 - Captura de pacotes Wireshark de um nó aceite

5 Conclusões

Os mecanismos de auto-configuração são fundamentais em redes com grande densidade de nós em que a grande maioria das comunicações são do tipo M2M e onde a infraestrutura de rede não é estável. Este é o caso das redes de sensores, onde os nós têm a capacidade de detectar e de se juntar a uma infraestrutura de rede existente. Este comportamento não é desejável quando se trata de infraestruturas que são utilizadas para suportar serviços intolerantes a falhas e onde a segurança é um requisito. Nestas situações, os mecanismos de auto-configuração continuam a ser necessários, no entanto devem ser complementados por outros que permitam ao gestor da rede o controlo da infraestrutura. É precisamente o caso dos mecanismos de controlo de acessos. Por um lado, podem ser utilizados como ferramenta de gestão porque proporcionam a visibilidade da rede. Por outro, permitem controlar quais os nós que podem aceder à rede, mitigando os efeitos de um grande número de ataques de segurança. Note-se que os mecanismos de controlo de acessos permitem que apenas os nós cujo comportamento é conhecido e previsível usem a infraestrutura de rede para comunicar com a Internet e com outros nós da rede. Os mecanimos de controlo de acessos são compostos por dois sub mecanismos: i) autenticação e identificação dos nós e ii) autorização dos nós. Neste trabalho apenas foi considerada a proposta do sub mecanismo de autenticação e identificação dos nós, que depois de implementado foi integrado e testado numa solução completa de controlo de acessos.

Tradicionalmente, os mecanismos criptográficos estão na base da grande maioria dos mecanismos de segurança que garantem a integridade e confidencialidade dos dados e a autenticidade dos nós. Nos mecanismos criptográficos modernos os algoritmos são públicos e apenas a chave é secreta, sendo por este motivo a gestão das chaves uma tarefa crítica. A troca de chaves é a ação mais complexa da gestão de chaves, motivo pelo qual as chaves têm um tempo de exposição longo por não serem refrescadas frquentemente. A solução proposta usa um modelo para gestão de chaves, com *bootstraping*, que permite que as chaves sejam refrescadas cada vez de um nó é iniciado, diminuindo assim o tempo de vida e a exposição das chaves. O processo de *bootstraping* também é adequado a redes mais dinâmicas pois adapta-se a mudanças frequentes de endereços IP. A solução proposta recorre ao uso de duas chaves, a primeira é utilizada para autenticar o nó e a segunda para cifrar os dados gerados pelo

nó. Para validar a solução proposta foi necessário configurar uma testbed e os respectivos testes.

A solução tem como principais vantagens a identificação, autenticação e configuração dos nós de uma forma segura e dinâmica. As chaves são estabelecidas dinâmicamente através do processo de Diffie-Hellman, para serem utilizadas nas ligações DTLS no modo PSK. O facto das chaves públicas dos nós serem refrescadas a cada vez que os nós são ligados faz com o tempo de exposição das chaves seja reduzido. A configuração dinâmica do nós através do *Bootsrap Server* faz com que seja possível os nós utilizarem uma chave definida pelo ECDH Server, tornando assim o *Bootsrap Server* numa entidade fundamental no processo

5.1 Limitações & trabalho futuro

Atendendo à evolução das redes de sensores sem fios existe a necessidade de proteger a rede de todos os tipos de ataques, desenvolvendo assim um ou vários mecanismos de acesso à rede para proteger contra todos os tipos de ataques possíveis.

A solução proposta pode ser complementada com a validação do firmware nos nós, para que apenas firmwares conhecidos sejam permitdos na rede, e com a validação da clonagem dos nós.

6 Bibliografia

[1] J. S. Silva, R. M. Silva e F. Boavista, Redes de Sensores Sem Fios.

[2] J. Gubbi, R. Buyya, S. Marusic e M. Palaniswami, *Internet of Things (IoT): A vision, architectural elements, and future directions,* 2013.

[3] M. Khanafer, H. T. Mouftah e M. Guennoun, "A survey of beacon-enabled IEEE 802.15.4 MAC protocols in wireless sensor networks".

[4] G. Mulligan, "The 6LoWPAN architecture," 2007.

[5] L. Oliveira, J. Rodrigues, A. Sousa e J. Lloret, *Denial of Service Mitigation Approach for IPv6-enabled Smart,* 2013.

[6] N. K. G. M. e C. S. , "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," 2007. [Online]. Available: https://tools.ietf.org/html/rfc4919.

[7] J. Olsson, *6LoWPAN demystified,* 2014.

[8] X. Chen, K. Makki, K. Yen e N. Pissinou, "Sensor Network Security: A Survey".

[9] W. L. Lee, A. Datta e R. Cardell-Oliver, *Network Management in Wireless Sensor Networks.*

[10] C. Nuangjamnong, S. P. Maj e D. Veal, "The OSI Network Management Model - Capacity," 2011.

[11] A. K. Rajpoot, M. Varshney e A. Nailwal, "Security and Privacy Challenges in the Internet of Things".

[12] D. Mendez, I. Papapanagiotou e B. Yang, "Internet of Things: Survey on Security and Privacy," 2017.

[13] R. Acharya e K. Asha, "Data integrity and intrusion detection in Wireless Sensor Networks".

[14] X. He, M. Niedermeier e H. d. Meer, Dynamic key management in wireless sensor networks: A survey.

[15] A. RGHIOUI, S. BOUCHKAREN, A. KHANNOUS e M. BOUHORMA, *Securing private wireless sensors in a sharedenvironment in the internet of things context,* 2014.

[16] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig e G. Carle, DTLS based Security and Two-Way Authentication for the Internet of Things, 2013.

[17] "FAQ: Node Joining Process in 6LoWPAN - ND, RPL," 2015. [Online]. Available: https://ez.analog.com/docs/DOC-12488.

[18] O. Gaddour e A. Koubâa, *RPL in a nutshell: A surve,* 2012.

[19] S. C. E. N. e C. B. , *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs),* 2012.

[20] A. H. Chowdhury, M. Ikram, H.-S. Cha, H. Redwan, S. M. S. Shams, K.-H. Kim e S.-W. Yoo, *Route-over vs mesh-under routing in 6LoWPAN,* 2009.

[21] M. Richardson e I. Robles, *RPL- Routing over Low Power and Lossy Networks,* 2011.

[22] E. P. Thubert, C. Systems, A. Brandt, S. Designs, J. Hui, A. R. Corporation, R. Kelsey, E. Corporation, P. Levis, S. University, K. Pister, D. Networks, R. Struik, S. S. Consultancy, J. Vasseur e C. Systems, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,* 2012.

[23] S. K. Rao, D. Mahto e D. D. A. Khan, *A Survey on Advanced Encryption Standard,* 2015.

[24] J. Muthukuru e P. B. Sathyanaryana , *A Survey of Elliptic Curve Cryptografy Implementation Approaches for Efficient Smart Card Processing.*

[25] N. Gura, A. Patel, A. Wander, H. Eberle e S. C. Shantz, *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs,* 2004.

[26] K. Jaewoo, J. HahnEarl e L. Jaiyong, *Network Management Framework for Wireless Sensor Networks.*

[27] S. Bonnet, "Smart M2M Gateway based Architecture for M2M Device and Endpoint".

[28] D. Bendouda, L. Mokdad e H. Haffaf, "Method For Fault Management With RPL Protocol in WSNs," 2015.

[29] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H. Schmidtke e M. Beigl, "Using Web Service Gateways and Code Generation for Sustainable IoT System Development".

[30] Q. Zhu, R. Wang, Q. Chen, Y. Liu e W. Qin, "IOT Gateway: Bridging Wireless Sensor Networks into Internet of Things".

[31] S. Guoqiang, C. Yanming, Z. Chao e Z. Yanxu, "Design and Implementation of a Smart IoT Gateway".

[32] S. K. Datta e C. Bonnet, "Smart M2M Gateway based Architecture for M2M Device and Endpoint".

[33] J. Sá Silva, E. Monteiro e J. Granjal, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," 2015.

[34] N. Sastry e D. Wagner, *Security Considerations for IEEE 802.15.4 Networks.*

[35] I. Mansour, G. Chalhoub e P. Lafourcade, "Key Management in Wireless Sensor Networks," 2015.

[36] R. Roepke, T. Thraem, J. Wagener e A. Wiesmaier, *A Survey on Protocols securing the Internet of Things: DTLS, IPSec and IEEE 802.11i.*

[37] N. Modadugu e E. Rescorla, "The Design and Implementation of Datagram TLS".

[38] L. Oliveira, J. Rodrigues, A. Sousa e J. Lloret, *A Network Access Control Framework for 6LoWPAN Networks,* 2013.

[39] Y. B. Zikria , M. K. Afzal, F. Ishmanov , S. W. Kim e H. Yu , *A survey on routing protocols supported by the Contiki Internet of things operating system,* 2018.

[40] "A deployment-ready 6LoWPAN Border Router solution based on Contiki," [Online]. Available: https://github.com/cetic/6lbr. [Acedido em 1 11 2018].

[41] Contiki, "RPL-border-router," [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-RPL-border-router. [Acedido em 11 11 2018].

[42] Contiki, "Documentation:-RPL," [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-RPL. [Acedido em 11 11 2018].

[43] Contiki, "Documentation:-LWM2M," [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-LWM2M. [Acedido em 11 11 2018].

[44] P. L. R. S. e D. C. , *TinyOS: An Operating System for Sensor Networks.*