UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Igor Cesar Gonzalez Ribeiro

Implementação de um Sistema Seguro de Distribuição e Armazenamento de Chaves Criptográficas em uma Rede de Sensores Sem Fio

Niterói
2010

Igor Cesar Gonzalez Ribeiro

Implementação de um Sistema Seguro de Distribuição e Armazenamento de Chaves Criptográficas em uma Rede de Sensores Sem Fio

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Célio Vinicius Neves de Albuquerque
Co-orientador: Juliano Fontoura Kazienko

Niterói
2010

RESUMO

A segurança em Redes de Sensores Sem Fio (RSSF) consiste em uma importante área de pesquisa na atualidade. Com o foco nesta, vários trabalhos têm estudado a distribuição de chaves criptográficas entre sensores com a intenção de tornar mais segura a comunicação de dados. Nesse contexto, projetar mecanismos seguros para essas redes consiste em um grande desafio uma vez que os recursos são escassos em tais dispositivos. Neste sentido, este trabalho tem o objetivo de implementar e avaliar um sistema de distribuição e armazenamento seguros de chaves criptográficas para RSSF. Para a implementação, foi utilizada uma abordagem baseada em codificação de redes onde um nó móvel é responsável por distribuir as chaves par-a-par entre os nós que desejam se comunicar. Após a distribuição das chaves, elas são armazenadas de forma segura na memória principal ou no dispositivo de armazenamento secundário dos sensores. Com a finalidade de avaliar o sistema implementado, são analisadas a sobrecarga introduzida no tempo de processamento e o consumo total de memória flash.

Palavras Chave:
Distribuição de Chaves, Armazenamento de Chaves, Segurança em RSSF

ABSTRACT

Currently, security in Wireless Sensor Networks (WSNs) is an important research area. With this in mind, many works have studied key distribution between sensors in order to become data comunication more secure. In this context, the building of secure mechanisms for these networks consists in a great challenge given the constrained resources on those devices. In this sense, the aim of this work is to implement and evaluate a secure key storage and distribution system for WSNs. For the implementation, an approach based on network coding where a mobile node is resposable to accomplish a pairwise key distribution, was used . After the distribution, the keys are stored in a secure way inside the sensor's primary or secondary storage device. With the aim of evaluating the developed system, the overhead introduced in processing time and the total memory usage are analysed.

Keywords:
Key Distribution, Key Storage, Security in WSNs

LISTA DE ACRÔNIMOS

RSSF:	Rede de Sensores Sem Fio
WSN:	Wireless Sensor Network
TinyOS:	Tiny Operating System
TOSSIM:	”TinyOs”Simulator
RAM:	Random Acces Memory
ROM:	Read Only Memory
EEPROM:	Electrical Erasable Programmed Read Only Memory
MANET:	Multihop Ad Hoc Networks
ED:	Event Detection
SPE:	Spatial Process Estimation
IEEE:	Institute of Electrical and Electronics Engineers
VSLA:	VITORIA Simulation Log Analyser

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1 - INTRODUÇÃO

A tecnologia das Redes de Sensores Sem Fio (RSSF) tem alavancado a evolução de aplicações em diversas áreas, como detecção de fogo em florestas, monitoramento de condições ambientais, detecção de inundação, entre outros [2]. Muitas dessas aplicações necessitam de algum nível de segurança com a intenção de evitar que, por exemplo, dados sensíveis transmitidos por um sensor sejam interceptados e agreguem conhecimento para pessoas não autorizadas. Ainda, para evitar que um sensor desconhecido possa enviar dados à rede, gerando dados falsos que podem resultar em interpretações incorretas que por sua vez podem causar resultados desastrosos. Fica claro então que é necessário adotar medidas de segurança para evitar que os eventos mencionados anteriormente ocorram, entretanto nem todas as aplicações têm exigências iguais quanto à segurança e cada mecanismo de segurança implementado consome recursos como processamento, memória e energia que são extremamente escassos nos nós sensores das RSSF.

No contexto de uma aplicação que exija autenticação e confidencialidade, pode-se utilizar mecanismos de criptografia de chaves simétricas para prover ambos. Entretanto, como em qualquer sistema criptográfico de chaves simétricas, é necessário se estabelecer uma forma segura pela qual as partes que desejam se comunicar possam tomar conhecimento da chave secreta.

Em muitos cenários de RSSFs, os nós operam de forma não assistida. Isso significa que os mesmos ficam expostos a violações físicas que podem comprometer a segurança de toda a rede. Assim, torna-se necessário não só a distribuição segura, mas também o armazenamento seguro das chaves utilizadas por esses nós para que ainda que um atacante tenha acesso as chaves, estas em nada sejam úteis para que o mesmo possa comprometer a segurança da rede.

O objetivo deste trabalho é implementar um sistema que provê confidencialidade às RSSF utilizando para tanto o esquema de criptografia de chaves simétricas. Para efetuar a distribuição das chaves, foi utilizada a abordagem de Codificação de Redes proposta em [19] e para garantir que as chaves sejam armazenadas de forma segura, foi utilizada, com algumas simplificações, a abordagem proposta em [11]. Adicionalmente, é realizada uma avaliação desta implementação através de métricas como uso das memórias ROM e RAM e sobrecarga no tempo de processamento, com a intenção de verificar a viabilidade de seu uso em RSSF reais.

Este trabalho está organizado da seguinte forma. No Capítulo 2 é apresentada uma rápida introdução sobre as redes de sensores sem fio, assim como uma visão geral dos sensores que correspondem aos nós destas redes. Ainda no Capítulo 2 é feita uma análise sobre a necessidade e técnicas de segurança para as RSSF. No Capítulo 3 este trabalho analisa o protocolo de distribuição segura de chaves criptográficas proposto em [19] e também o protocolo que provê o armazenamento seguro de chaves criptográficas proposto em [11]. O Capítulo 4 descreve a metodologia utilizada neste trabalho para implementar e avaliar o sistema proposto. O Capítulo 5 contém uma descrição detalhada do sistema implementado neste trabalho. O Capítulo 6 apresenta a avaliação realizada sobre o sistema proposto e o capítulo 7 conclui este trabalho.

CAPÍTULO 2 - REDES DE SENSORES SEM FIO

2.1 VISÃO GERAL

Uma rede de sensores sem fio (RSSF) pode ser classificada como um tipo especial de rede *ad hoc* de múltiplos saltos *(MANETs - Multihop Ad Hoc Networks)* tendo em vista o grande número de similaridades entre ambas. Na maioria das aplicações onde a tecnologia de RSSF é utilizada, a finalidade é coletar dados através dos nós sensores e esses dados então devem ser enviados a um determinado destino para análise e processamento. Para tanto, os nós sensores de tais redes captam a informação requerida para a aplicação, a empacotam e a enviam através de ondas de rádio para um outro nó específico ou para todos os nós na rede dentro do alcance do seu rádio. Os dados são então repassados através de um ou mais saltos para um nó sorvedouro que pode utilizar a informação localmente ou envia-la para outras redes através de um *gateway*. A figura 1 mostra o esquema de uma RSSF como explicado anteriormente.

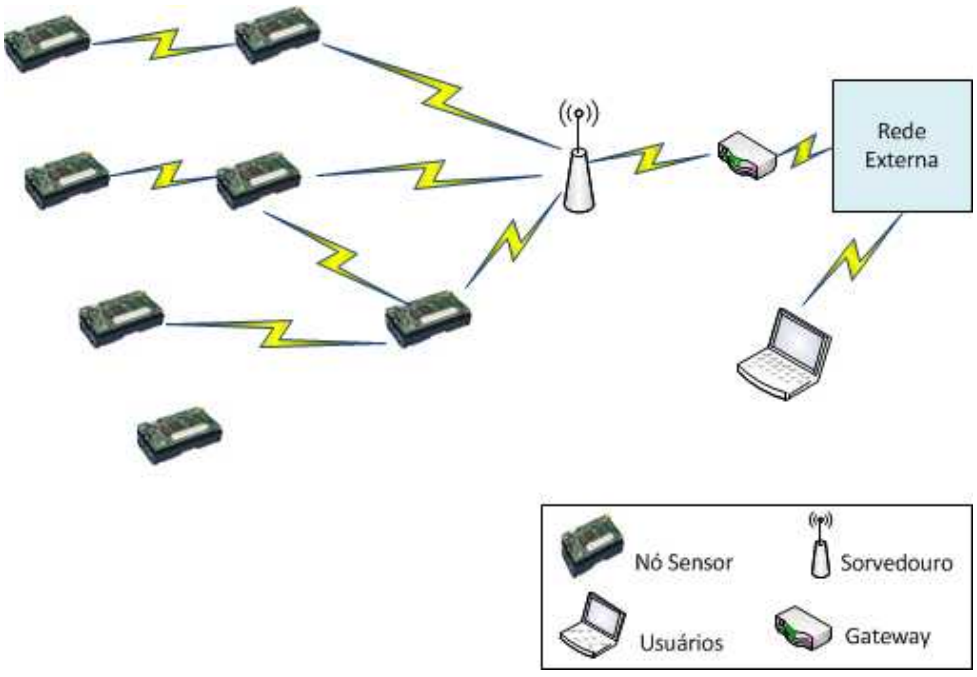

Figura 1: Roteamento de Pacotes com um Nó Sorvedouro. Os dados são roteados através de múltiplos saltos até o nó sorvedouro que então os envia para um *gateway* e este repassa para a rede externa.

Muitas aplicações desenvolvidas para as RSSF necessitam de configuração dinâmica dos

nós ou informações provenientes de uma localização remota. Assim, o nó sorvedouro também é utilizado para disseminar pacotes no sentido da rede externa para a RSSF.

Um problema que pode ser verificado com a utilização da estrutura mencionada acima é a baixa escalabilidade da RSSF. Com apenas um nó sorvedouro, conforme a rede cresce este deve atender a cada vez mais nós sensores que precisam enviar os dados coletados para a rede externa. Assim, a RSSF passa a ter um limite superior quanto ao número de nós sensores que a mesma pode conter. Além disso, se o nó sorvedouro falhar, nenhum pacote chegará ao seu destino. Com o objetivo de sanar tais problemas, pode-se utilizar mais de um nó sorvedouro como mostra a figura 2.

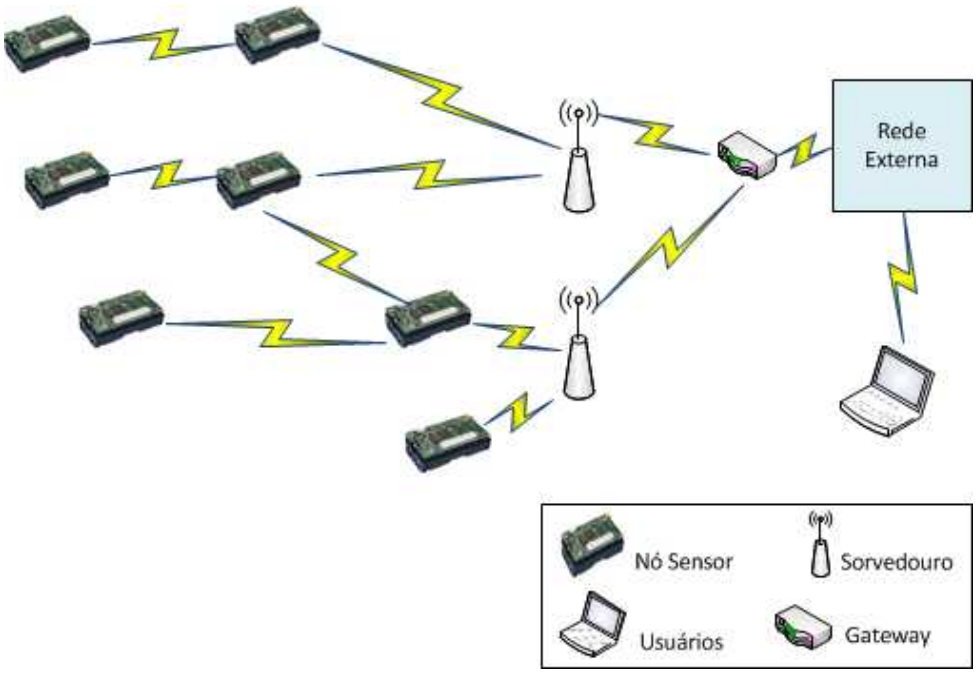

Figura 2: Roteamento de Pacotes com multiplos Nós Sorvedouros. É preciso definir no algoritmo de roteamento para qual nó sorvedouro rotear os pacotes.

A utilização de mais de um nó sorvedouro por um lado minimiza os problemas de escalabilidade e ponto único de falhas, mas por outro lado introduz uma complexidade extra no protocolo utilizado pelos nós sensores, pois agora é necessário fazer uma escolha sobre para qual nó sorvedouro rotear os dados. Essa escolha pode depender de vários fatores como: sobrecarga de um nó sorvedouro, número de saltos entre o sensor e o sorvedouro, etc. [4].

Como na maioria das aplicações as RSSF devem funcionar de forma não assistida por um longo período de tempo e a fonte de energia presente nos sensores é limitada, é muito importante que se tenha um cuidado especial no desenvolvimento de protocolos que otimizem o consumo de energia. Para tanto, um método comumente utilizado é o dos ciclos de operação [17], onde os sensores alternam entre os períodos de atividade, inatividade(*idle*) e baixo consumo (*sleep*).

2.2 OS NÓS SENSORES

Os nós sensores de uma RSSF são dispositivos com 4 funções principais: sensoriamento, processamento, comunicação e armazenamento [2]. Como mostrado na Figura 3, um sensor é composto basicamente de um micro-controlador responsável por realizar um pequeno processamento de dados, uma memória RAM, uma memória não volátil utilizada para armazenar dados referentes a inicialização do sistema que pode ser do tipo EEPROM(*Electric Erasable PROM*), um dispositivo de armazenamento não volátil utilizado para armazenamento de dados e do programa a ser executado pelo sensor, um dispositivo receptor/transmissor de rádio-frequência utilizado para enviar e receber dados e os sensores propriamente ditos que podem ser de luminosidade, umidade, temperatura, dentre outros.

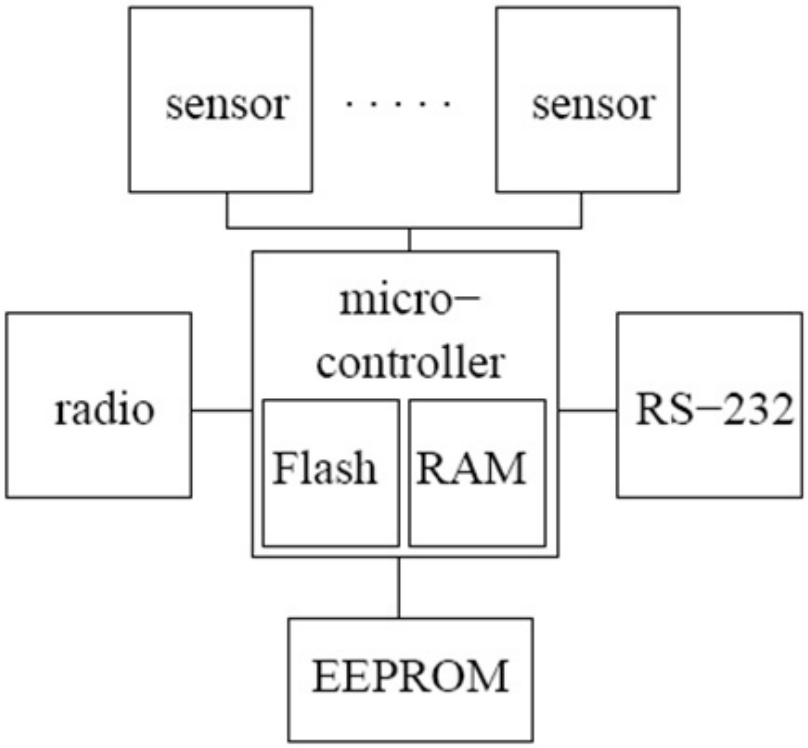

Figura 3: Esquema geral de um nó sensor [3].

Em aplicações reais de redes de sensores sem fio, são utilizados milhares de sensores com a finalidade de cobrir a maior área possível e assim obter medições mais realísticas. Dessa forma, se cada sensor fosse um dispositivo caro, o custo de uma RSSF poderia tornar a maior parte das aplicações inviável financeiramente. Percebe-se então que o custo tem uma grande importância no projeto de tais dispositivos e, por isso, a capacidade dos recursos como: processador, memórias e largura de banda deve ser equilibrada com seus custos associados. Além disso, *hardware* mais poderoso também consome mais energia, o que poderia degradar o tempo

de operação destes dispositivos. A Tabela 1 mostra a capacidade do hardware para alguns sensores comerciais.

Tabela 1: Capacidade de hardware de alguns sensores comerciais [3].

Modelo	**Micro-Controlador**	**RAM**	**Memória Flash**	**Largur**
Mica2	8bit Atmel ATmega	4KB	128KB	76
ESB (Embedded Sensor Borads)	Texas Inst. MSP430 F1611	2KB	60KB	19
Telos	16bit Texas Inst. MSP430 F1611	10KB	48KB	25

2.3 APLICAÇÕES

A flexibilidade da tecnologia das RSSF tem possibilitado um número muito grande de aplicações em diversas áreas do conhecimento. Muitas são as situações em que uma determinada medição ou monitoramento devem ser feitas em uma área onde os seres humanos não podem chegar por causa de condições ambientais impróprias para a vida. Ainda, a simples presença de pessoas no ambiente poderia alterar a acuracidade dos dados obtidos, ou por alguma outra restrição imposta pela aplicação.

Nesses casos, utilizar uma RSSF é a solução ideal já que como foi visto anteriormente, os sensores podem coletar dados e enviar ao destino, assim como serem configurados sem que seja necessária a presença de qualquer ser humano.

Segundo [4], podemos classificar as aplicações em RSSF, quanto ao tipo de dados manipulado, em duas categorias: detecção de eventos (*ED - Event Detection*) e estimação de processos espaciais (*SPE - Spatial Process Estimation*).

No primeiro caso, a finalidade da aplicação é detectar a ocorrência de eventos e alertar as pessoas ou dispositivos interessados no respectivo evento. Para tanto, é necessário que os sensores monitorem uma determinada características em períodos de tempo determinados (segundo a política de ciclos de operação discutida anteriormente) e compare o valor obtido com um valor de limiar (*Threshold*). Caso o valor exceda o limiar então significa que o evento ocorreu e deve-se então adotar as medidas estabelecidas de acordo com o protocolo usado pela RSSF.

Uma vez que as redes de sensores sem fio, como o próprio nome diz, funcionam com enlaces sem fio, elas sofrem de problemas de interferência oque causam um número maior de perdas e atrasos de pacote do que em uma rede cabeada. Em aplicações comuns, a perda ou atraso ex-

cessivo de um pacote não representa graves problemas, mas para as aplicações de detecção de eventos que funcionam em RSSF, podem gerar grandes incidentes como alarmes falsos sobre desastres ambientais ou a ausência de alarmes verdadeiros sobre as condições de saúde de um paciente em observação. Com a intenção de se aprimorar a confiabilidade da RSSF quanto a emissão correta de alarmes de ocorrência de eventos, é de fundamental importância escolher um número grande o suficiente de sensores para compor a rede de forma que a informação a ser enviada seja redundante o suficiente para que perdas ou atrasos de pacotes não impeçam a informação de chegar ao seu destino.

Como exemplo dessa categoria de aplicações pode-se citar a detecção de erupções vulcânicas como mostrado na Figura 4, de superaquecimento em uma usina e de um determinado parâmetro clínico de um paciente em um hospital.

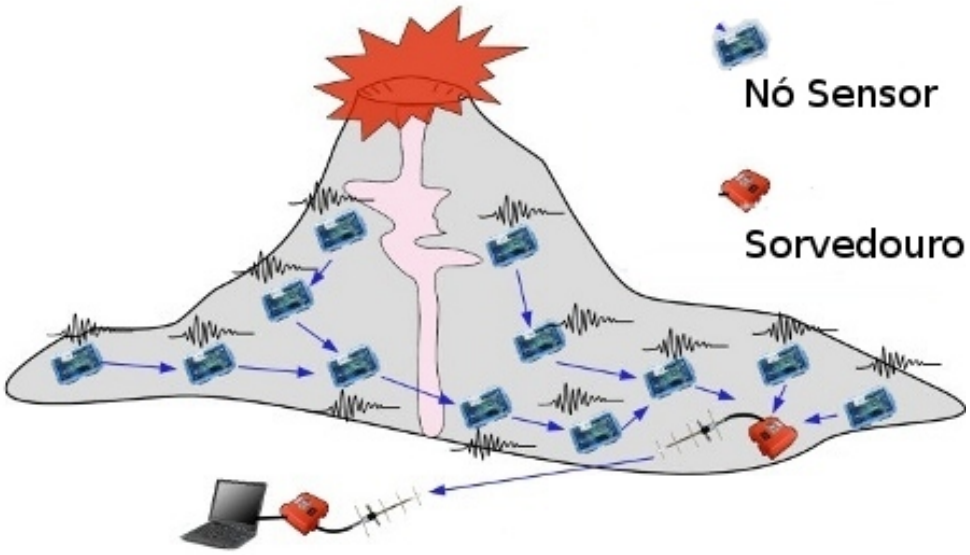

Figura 4: Aplicação de RSSF para detecção de erupções vulcânicas. Os sensores são dispostos sobre a área do vulcão e analisam informações como temperatura e pressão e caso os valores medidos estejam acima ou abaixo de um valor de patamar, um alerta é emitido.

Em uma aplicação SPE a finalidade da rede é coletar dados para estimar um determinado fenômeno como por exemplo a variação de temperatura na floresta amazônica ou a pressão atmosférica em uma grande área. Uma vez que em uma grande área um fenômeno pode apresentar medições diferentes em regiões diferentes, é necessária que a RSSF seja densa o suficiente para que o modelo gerado pelo conjunto de dados coletados pela mesma seja suficientemente próximo da situação real do ambiente analisado. Entretanto, quanto mais densa a rede maior o seu custo e dessa forma torna-se necessário um balanço entre o nível de realismo do modelo desejado e o custo com o qual é possível arcar.

Em [4] é descrita, como estudo de caso, uma aplicação real utilizada na agricultura. Uma

RSSF foi montada com *hardware* e *software* de baixo custo e tem a função de periodicamente monitorar alguns parâmetros físicos como temperatura, pressão do ar, umidade, etc. Com a informação obtida através desta aplicação é possível aumentar a quantidade e a qualidade da produção e ainda reduzir os custos.

2.4 SEGURANÇA EM RSSF

Como visto anteriormente, existem inúmeras aplicações possíveis para as RSSF e cada uma delas pode exigir serviços diferentes de segurança. Um sistema que monitora a ocorrência de incêndios em uma floresta, por exemplo, precisa garantir que quando a temperatura ultrapassar o limiar, o alerta emitido deve chegar ao seu destino, pois caso isso não aconteça, um desastre ambiental pode ocorrer. É possível perceber nesse caso que é muito importante que o sistema opere sem interrupções ou seja, é preciso garantir a disponibilidade da rede. Além disso, quando o destino receber o alerta, é necessário que o mesmo confie na informação que recebeu ou seja, é necessário que o destino tenha certeza que o nó ou os nós que enviaram este alerta são de fato nós autênticos, que os dados recebidos são íntegros (a informação enviada e a recebida são idênticas) e que a informação recebida não é uma reprodução de uma transmissão antiga (*Data Freshness*).

Em uma aplicação militar onde a RSSF é responsável por monitorar as atividades de um exército inimigo, além dos serviços de segurança mencionados no exemplo anterior, é necessário que a informação transmitida assim como as informações de roteamento sejam confidenciais.

Apesar de muito importantes, os serviços de segurança impõem algum nível de dificuldade quanto à sua implantação nas RSSF já que tais serviços exigem mais dos recursos (processamento, memória e energia) dos nós sensores, que os possuem em quantidade bastante limitada e por isso podem gerar um gargalo na rede. A seguir são discutidos os desafios encontrados no desenvolvimento de protocolos que implementem os serviços de segurança mencionados [9, 17]:

- **Confidencialidade:** com o objetivo de prover confidencialidade, é necessário a utilização de algoritmos de criptografia que convertem um texto claro em uma saída que não agrega conhecimento a um possível atacante. Cada nó que precise conhecer o conteúdo da mensagem cifrada, deve utilizar o algoritmo para decriptar a mensagem e caso haja necessidade do nó encaminhar a mensagem novamente para a rede, este deve cifrar a mensagem novamente. Este processo pode tornar o uso excessivo do processador a ponto de gerar um

atraso tão grande que torne a aplicação inviável. Além disso, a utilização maior do processador também gasta mais energia que pode levar a uma minimização do tempo de vida da rede. Ainda, a utilização de cifras simétricas requerem um meio pelo qual os nós tomem conhecimento das chaves a serem utilizadas. Assim, é necessário que se estabeleça um sistema de distribuição de chaves que, com exceção do esquema de pré-distribuição, aumentará o número de mensagens trocadas consumindo mais banda e também bateria.

- **Autenticação:** para garantir a autenticidade de um dado em uma RSSF é necessário incluir no pacote uma etiqueta de autenticação. Apesar de pequeno, o aumento do tamanho do pacote de dados poderia, além de causar um consumo maior de energia em cada nó no caminho que leva o dado da origem ao destino, também torna o pacote mais sujeito a erros de transmissão causados principalmente por interferências.

- **Integridade:** o serviço de integridade também requer o uso de criptografia e está sujeito aos mesmos problemas de implementação que o serviço de confidencialidade.

- **Disponibilidade:** uma vez que as RSSF normalmente operam sem o monitoramento dos nós sensores e muitas vezes em ambientes hostis, estes podem facilmente ser capturados e modificados para que operem de forma inapropriada ou ainda um grande número de sensores podem ser substituídos por sensores falsos que contribuam para o mal funcionamento da rede causando sua indisponibilidade.

A utilização de enlaces de comunicação sem fio e a forma não monitorada de operação dos nó sensores em uma RSSF introduzem brechas de segurança que podem ser exploradas de muitas formas diferentes. Um atacante que pretendesse causar o mal funcionamento da rede, evitando assim que a mesma pudesse cumprir a sua função, poderia simplesmente introduzir um dispositivo (que poderia ser também um sensor) que emitisse sinas aleatórios na mesma frequência de forma a interferir na comunicação normal da rede. Para evitar tal problema, poderia ser usado uma forma de codificação de sinais conhecida como espalhamento espectral. Entretanto, rádios que utilizam este tipo de codificação são mais complexos e por isso mais caros além de consumir mais energia [9, 17].

Um outro tipo de ataque é o da captura de nós sensores. Uma vez que esses dispositivos não são monitorados, torna-se fácil captura-los e extrair os dados contidos nele. Dessa forma, é necessário que dados confidenciais que necessitam ser mantidos nos sensores estejam na forma encriptada. Porém, as chaves utilizadas também devem ser mantidas nos sensores e caso estas estivessem na forma de texto claro, de nada adiantaria codificar os dados. Como solução para este problema, pode-se adotar duas alternativas: a primeira é a utilização de chaves variáveis

para evitar que chaves obtidas através de sensores capturados pudessem ser utilizadas posteriormente. A segunda alternativa é a utilização de dispositivos *tamper proof* para armazenamento de dados sigilosos. O problema em utilizar tais dispositivos é que eles poderiam elevar o custo dos sensores. Estas duas alternativas poderiam ser utilizadas em conjunto para aumentar o nível de segurança. No Capítulo 3 será discutida uma abordagem que utiliza a tecnologia *tamper proof* para proteger um segredo utilizado para encriptar chaves.

As RSSF operam em modo *Ad Hoc* o que significa que todos os seus nós são responsáveis por rotear pacotes da origem até o destino. Assim, é possível manipular os pacotes de roteamento de forma a causar diversos tipos de problemas na rede como: partição da rede, aumento da latência fim-a-fim, aumento ou diminuição de rotas, bloqueio de nós, etc. Dentre os possíveis ataques pode-se citar [10]:

- **Falsificação, alteração ou reprodução de informações de roteamento:** através da manipulação de informações de roteamento, um atacante poderia, por exemplo, criar rotas muito grandes o que elevaria o consumo de energia causando a "'morte"' de alguns sensores, degradando os serviços da rede. Uma outra situação é a potencial geração de *loops* nas rotas já que o atacante poderia repassar os pacotes para o sensor que os enviou que por sua vez enviaria novamente para o primeiro.

- **Repasse Seletivo:** redes *Multi-hop* frequentemente assumem que seus nós irão repassar prontamente os pacotes que receberam de forma que os dados cheguem até o nó sorvedouro. O ataque de repasse seletivo explora esta característica fazendo com que um nó malicioso repasse apenas os pacotes que lhe forem interessantes. A forma mais fácil de realizar este tipo de ataque é quando um atacante, através da falsificação de pacotes de roteamento, consegue inserir um nó malicioso no fluxo de dados entre a origem e o destino. Uma vez fazendo parte do fluxo de dados, todos os pacotes daquele fluxo passarão por ele e será sua decisão repassa-los ou não. Uma outra forma de se realizar esse ataque é quando o nó malicioso não faz parte do fluxo de dados, mas consegue ouvir a comunicação dos nós vizinhos. Neste caso, o atacante poderia gerar uma interferência na comunicação alvo para que determinados pacotes não fossem recebidos ou fossem recebidos com erro.

- **Ataques de escoamento:** o objetivo principal deste ataque é fazer com que todos os fluxos de pacotes passem pelo nó malicioso. Os protocolos de roteamento implementado nos sensores deve escolher o nó vizinho para o qual repassar um determinado pacote. Essa escolha pode depender de vários fatores como: menor número de saltos até a estação base,

menor consumo de energia, melhor qualidade de sinal, etc. Dessa forma, um atacante poderia enviar pacotes de roteamento oferecendo a melhor rota em termos da preferência do protocolo e dessa forma, todos os outros nós da rede escolheriam uma rota que passasse pelo nó malicioso. Uma vez alcançado esse objetivo, o nó malicioso poderia descartar todos os pacotes da rede, funcionando assim como um ”buraco negro”ou gerar um ataque de repasse seletivo.

O assunto de segurança em RSSF é extremamente amplo e inúmeros trabalhos já foram escritos abordando-o. Dessa forma, não é o objetivo deste trabalho esgotar todo este assunto, ao contrário, foi apresentada aqui apenas uma visão geral sobre alguns temas que estão mais relacionados com o objetivo real deste trabalho que é a distribuição e armazenamento seguros de chaves criptográficas. Muito mais informações podem ser encontradas em [17, 10, 1, 22].

CAPÍTULO 3 - TRABALHOS RELACIONADOS

Como visto no Capítulo 2, o modo de operação dos nós sensores de uma RSSF criam inúmeras brechas de segurança que se não corrigidas podem gerar resultados catastróficos. Muitos dos serviços de segurança necessários são implementados através de algoritmos criptográficos.

À um sistema criptográfico, quando uma mesma chave é utilizada tanto no processo de encriptação quanto no processo de decriptação de um determinado conjunto de bits, da-se o nome de **Cifra de Chaves Simétricas**. Por outro lado, se uma chave é utilizada no processo de encriptação e uma outra, diferente da primeira, é utilizada no processo de decriptação de um conjunto de bits, da-se o nome de **Cifra de Chaves Assimétricas** ou **Cifra de Chave Pública**.

Um problema fundamental relacionado às cifras de chaves simétricas é a necessidade de que todas as partes que desejam se comunicar conheçam a chave. Assim, a maneira pela qual as partes tomam conhecimento da chave é de fundamental importância para a segurança do criptosistema.

Com o objetivo de resolver o problema inerente ao criptosistema de chaves simétricas descrito, foi desenvolvido o criptosistema de chaves assimétricas, onde cada parte que necessite se comunicar de forma segura tem duas chaves. Uma dessas chaves, conhecida como chave pública, é disponibilizada publicamente, por exemplo através da Internet. A outra chave, conhecida como chave privada, deve ser mantida em segredo. Se uma fonte de dados A precisa enviar uma mensagem confidencial m para B, então A encripta a mensagem m utilizando a chave pública de B, K_B^+. Quando B receber a mensagem ele então deve utilizar sua chave privada, K_B^- para decriptar a mensagem. As chaves são geradas em conjunto de tal forma que uma mensagem encriptada com K_B^+ só possa ser decriptada utilizando-se K_B^-.

A utilização de cifras simétricas, como o RC4, DES e AES [21], é muitas vezes considerada mais apropriada para uso em sistemas embarcados como os sensores, por dependerem de algoritmos que contém instruções bit-a-bit como ou-exclusivo e deslocamentos, que são executadas no processador com um número menor de ciclos de relógio e utilizam menos memória e energia do que funções matemáticas mais complexas presentes em algoritmos de cifras assimétricas como o RSA [21] e o ECC [18].

Apesar dos benefícios quanto ao desempenho, como mencionado anteriormente, a segurança das cifras simétricas dependem da segurança do sistema de distribuição de chaves utilizado. Além disso, uma vez que a captura de nós sensores das RSSF não constituem uma atividade muito complexa, a forma como os sensores armazenam as chaves e outros dados sigilosos também têm importância fundamental para a segurança.

Com o objetivo de solucionar o problema da distribuição segura de chaves criptográficas pelos nós de uma RSSF, muitos trabalhos foram desenvolvidos adotando diferentes esquemas como: pré-distribuição de chaves [6], distribuição através de infraestrutura de chave pública [16] e a distribuição através de uma terceira parte confiável [20].

- **Pré-distribuição de chaves:** apesar da pouca necessidade de poder computacional, cada sensor necessita armazenar todas as chaves dos nós com os quais deseja se comunicar. Dessa forma, a quantidade de memória que os nós sensores dispõem acaba limitando a densidade da rede. Além disso, uma vez que as chaves foram armazenadas e a rede entrou em utilização, adicionar um novo sensor à esta não é trivial, tornando esse sistema não escalável.

- **Distribuição através de infraestrutura de chave pública:** a utilização de sistemas de chave pública garante a autenticação e a distribuição de chaves secretas sem a necessidade de um segredo compartilhado já que os nós utilizam as chaves públicas dos seus vizinhos para criptografar a chave secreta que este deseja compartilhar com cada nó. Apesar dos benefícios, um sistema de chave pública requer uma capacidade de processamento e memória maior que a disponível em muitos sensores comerciais [19].

- **Distribuição através de uma terceira parte confiável:** introduz um super nó que possui comunicação segura com os demais nós da rede e é responsável por distribuir as chaves utilizadas pelos mesmos. Esse esquema tem a vantagem de requerer pouco poder computacional, entretanto introduz um ponto único de falha o que representa um grande risco à segurança.

3.1 SISTEMA PROPOSTO POR OLIVEIRA E BARROS

Além dos sistemas abordados anteriormente, Oliveira e Barros propuseram em [19] um esquema de distribuição de chaves baseado nos modelos de pré-distribuição, codificação de redes (*Network Coding*) [5] e distribuição através de uma terceira parte confiável. Nesse esquema, um nó móvel é utilizado para estabelecer conexões seguras entre os nós sensores da rede. A figura 5 ilustra uma RSSF com a presença de um nó móvel.

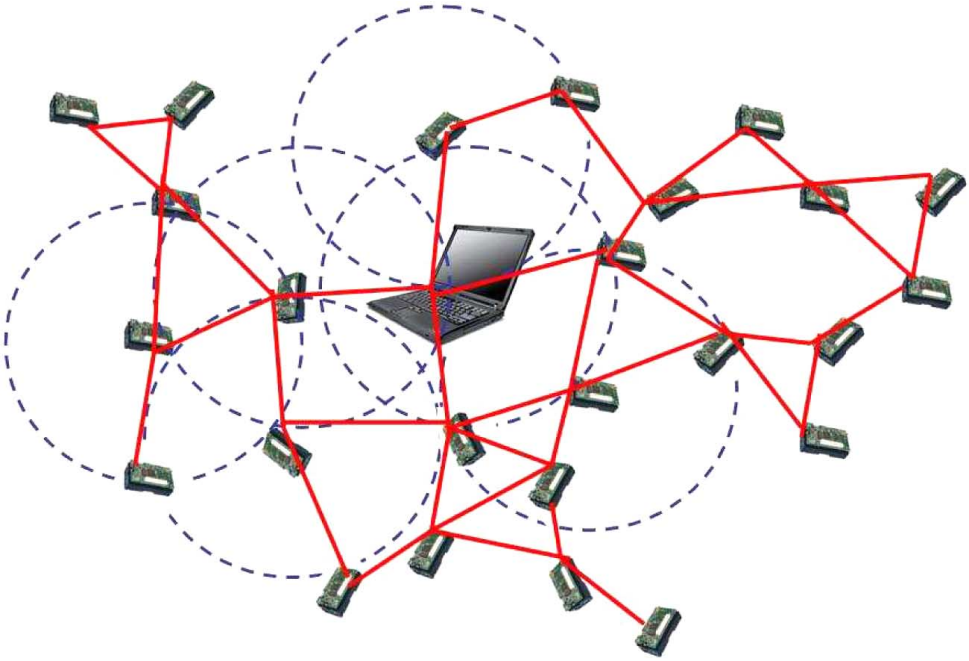

Figura 5: Modelo de uma rede de sensores sem fio com a presença de um nó móvel representado por um laptop. [19].

A seguir o protocolo de distribuição de chaves em uma RSSF entre apenas dois nós, proposto em [19], é descrito:

1. **Antes da implantação dos nós sensores na rede:**

   - É gerado um grande conjunto ρ de chaves K_i estatisticamente independentes e seus identificadores i onde $i \in \{0, \ldots, |\rho| - 1\}$;
   - É gerada uma sequência binária R de números aleatórios com o mesmo tamanho das chaves K_i em bits;
   - É armazenada na memória do no móvel uma tabela contendo os índices i de cada chave e a versão encriptada $K_i \oplus R$ da mesma;
   - Faça $n \ll |\rho|$ ser o número de *links* que cada nó pretende utilizar durante seu tempo de vida na rede (esse número pode ser aumentado posteriormente). Para cada nó j da rede, carregue em sua memória um subconjunto S_j de ρ contendo n chaves tal que $(S_1 \cap S_2 \cap \ldots \cap S_n) = 0$, juntamente com seus respectivos identificadores.

2. **Depois da implantação dos nós sensores na rede**

   - O nó móvel envia mensagens de HELLO em modo *broadcast* para a RSSF;
   - Cada nó sensor dentro da área de alcance do nó móvel recebe a mensagem de HELLO e responde com o identificador de uma das chaves de seu conjunto S;

- Após o nó móvel receber o identificador $i(A)$ do nó A e $i(B)$ do nó B ele os utiliza para indexar sua tabela de chaves e recuperar $K_i(A) \oplus R$ e $K_i(B) \oplus R$ respectivamente. Logo após, é executada a operação de XOR de codificação de redes sobre as chaves recuperadas $K_i(A) \oplus R \oplus K_i(B) \oplus R$. O fator R é cancelado na operação, resultando em $K_x = K_i(A) \oplus K_i(B)$. E assim, K_x é enviada aos nós A e B;
- Uma vez que os nós A e B conhecem K_x e suas respectivas chaves $K_i(A)$ e $K_i(B)$, A pode recuperar $K_i(B)$ fazendo $K_i(B) = K_x \oplus K_i(A)$ e B pode recuperar $K_i(A)$ fazendo $K_i(A) = K_x \oplus K_i(B)$;

Uma vez concluído o processo de distribuição, o nó A então pode enviar mensagens $mA \rightarrow B$ encriptadas com $K_i(A)$, ou seja $E_{Ki}(A)(mA \rightarrow B)$ para B e B por sua vez pode enviar mensagens $mB \rightarrow A$ encriptadas com $K_i(B)$, ou seja $E_{Ki}(B)(mB \rightarrow A)$ para A. A Figura 6 ilustra a etapa 2 do protocolo descrito acima e a posterior troca de mensagens encriptadas entre os dois nós, A e B da rede.

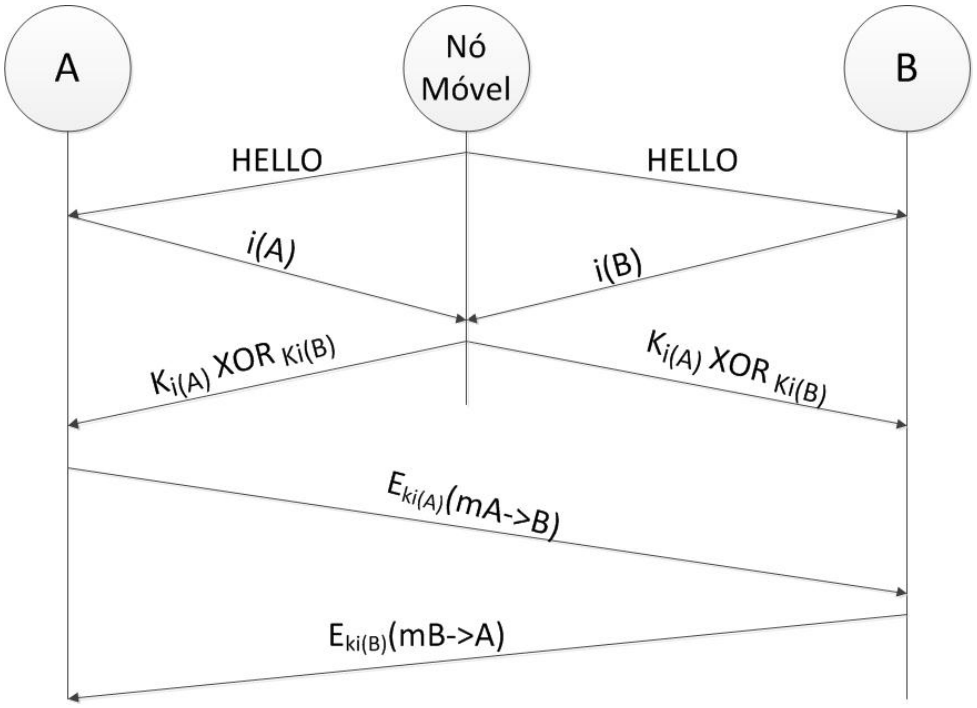

Figura 6: Protocolo de distribuição de chaves para uma RSSF com apenas dois nós [19].

Oliveira e Barros mostram em [19] que esse protocolo pode ser expandido para acomodar redes com um número arbitrário de nós de uma maneira simples e direta. Além disso, no mesmo trabalho são apresentadas possíveis extensões para prover serviços como: autenticação do nó móvel, requisição de chaves extras, chaves de *cluster* e revogação de chaves.

O sistema proposto por Oliveira e Barros funciona mesmo diante de *hardwares* extremamente limitados em processamento, memória e comunicação [19]. Além disso, provê segurança

extra já que além de ser possível implementar a autenticação do nó móvel, este guarda as chaves de forma codificada o que elimina a possibilidade de a partir de um ataque direcionado ao nó móvel, ter a segurança de toda a rede comprometida.

Apesar do grande número de vantagens, é possível identificar dois fatores no sistema de Oliveira e Barros que podem gerar preocupação. Primeiramente, para a implementação de tal sistema é necessário que além dos nós sensores tradicionais, haja um nó móvel com capacidade de processamento, armazenamento e comunicação maior do que a dos outros nós da rede. A necessidade do nó móvel pode, em algumas situações, inviabilizar sua utilização.

Em segundo lugar, uma falha mais grave pode ser observada quanto a segurança. Uma vez que as chaves de cada nós são guardadas em forma de texto claro na memória dos sensores, a captura dos mesmos pode resultar na descoberta de todas as chaves neste armazenadas. Em uma situação ainda pior, um nó sensor poderia ser atacado em conjunto com ó nó móvel. Nesse caso, a descoberta da chave do nó sensor $K_i(A)$ poderia ser utilizada para fazer um XOR com a sequência $K_i(A) \oplus R$, obtida através do ataque ao nó móvel. O resultado de tal operação seria o conjunto de bits R usado para codificar todas as chaves armazenadas no nó móvel. O resultado de tal ataque seria a descoberta de todas as chaves do conjunto ρ acabando por completo com a segurança da rede.

Com o objetivo de contornar o segundo problema, pode-se utilizar nós sensores com tecnologia *tamper proof* para assegurar que um ataque a um nó não revelaria o conteúdo de sua memória. Essa solução funciona bem, mas pode representar um elevado aumento no custo final da RSSF, já que é necessário prover a resistência a ataques (*tamper proof*) a todos os bits da memória, o que pode ser caro.

3.2 SISTEMA PROPOSTO POR KAZIENKO E ALBUQUERQUE

Com o objetivo de solucionar o problema do armazenamento seguro de chaves em nós sensores, encontrado no trabalho de Oliveira e Barros [19], foi proposto por Kazienko e Albuquerque em [11], um sistema que mantém as chaves criptográficas armazenadas na memória dos nós sensores na forma encriptada. Para tanto, uma sequência de bits Z com o mesmo tamanho das chaves K_i deve ser gerada por cada sensor e então armazenada em um módulo criptográfico dotado de tecnologia *tamper proof*.

A Figura 7 ilustra o sistema proposto por Kazienko e Albuquerque. O cadeado presente nos sensores A e B representam o fato de que as chaves criptográficas são armazenadas de forma segura nos mesmos. A distribuição das chaves é realizada como no sistema de Oliveira

e Barros, uma operação de ou-exclusivo é executada entre as chaves que se deseja distribuir (1) e em seguida o resultado é enviado em modo *broadcast* para a rede (2). Quando o nó A recebe a mensagem de distribuição, este realiza uma operação de ou-exclusivo entre a sua chave (codificada) e o conteúdo da mensagem de distribuição ou seja, $K_i(A) \oplus Z_A \oplus K_i(A) \oplus K_i(B)$ o que resulta em $K_i(B) \oplus Z_A$ (4). O mesmo processo é realizado no sensor B (3). Para o nó A recuperar uma chave qualquer $K_i(A)$, este deve utilizar a versão da chave codificada armazenada em sua memória (5) e o segredo Z_A armazenado no módulo criptográfico (6) e executar uma operação de ou-exclusivo entre ambos também no módulo criptográfico. Para o nó B recuperar sua chave, o mesmo procedimento deve ser seguido, passos (9) e (10) da Figura 7. Para o sensor A enviar uma mensagem M para o sensor B, o sensor A utiliza um algoritmo de encriptação, executado no módulo criptográfico, que recebe como parâmetros a mensagem M e a chave $K_i(A)$ recuperada previamente (7). Quando o Nó B receber a mensagem M (8), este deve recuperar a chave de A $K_i(A)$ e utiliza-la juntamente com a mensagem criptografada como parâmetros de um algoritmo de decriptação executado no módulo criptográfico (11). Se um nó A for violado, uma mensagem de tentativa de violação é enviada (12). O mesmo ocorre para o nó B.

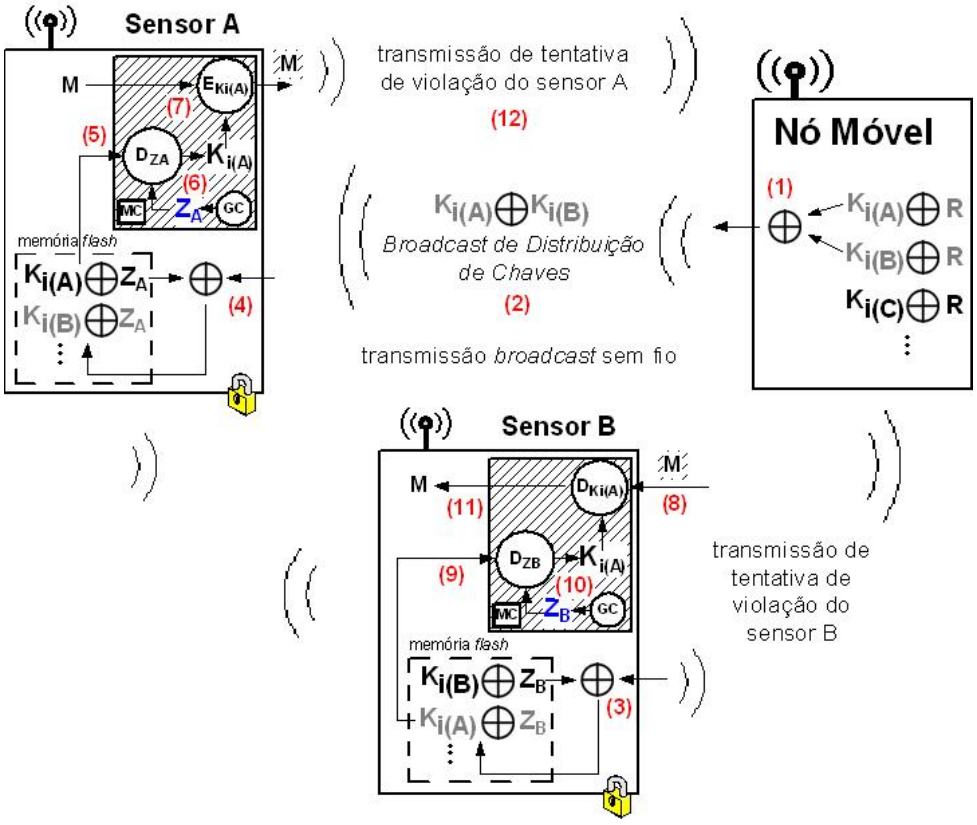

Figura 7: Esquema de distribuição e armazenamento seguros de chaves criptográficas [11]. (1) o nó móvel realiza o ou-exclusivo entre o par de chaves(contendo o segredo R) a serem distribuídas

Além do armazenamento seguro, também é proposto um alarme de tentativa de violação

que tem a função de alertar o nó móvel de que um determinado sensor foi atacado e o nó móvel por sua vez pode decidir qual é a maneira mais adequada de responder a este evento, que pode ser por exemplo, a revogação de todas as chaves conhecidas pelo sensor atacado.

Para demonstrar que o sistema proposto em [11] eleva a segurança em relação ao sistema de Oliveira e Barros, um modelo de atacante é desenvolvido. As possíveis atividades do atacante em tal modelo e suas respectivas repercussões são:

1. **O atacante é capaz de escutar a comunicação através dos canais sem fio:** neste caso, uma vez que a distribuição das chaves é feita através da técnica de codificação de redes, um atacante só poderia obter o resultado da operação de ou-exclusivo entre um par de chaves de dois nós da rede, o que não representa informação útil contanto que o mesmo não conheça nenhuma das duas chaves envolvidas.

2. **O atacante pode capturar qualquer um dos nós sensores da rede, mas não o nó móvel e extrair o conteúdo de suas memórias:** diferentemente do que ocorre no sistema de Oliveira e Barros, essa situação não fornece risco a segurança do sistema proposto por Kazienko e Albuquerque. Uma vez que o atacante poderia extrair da memória dos nós somente o conjunto de dados $\{K_1 \oplus Z_1, K_2 \oplus Z_1, \ldots, K_n \oplus Z_1\}$ e como demonstrado em [11] o conhecimento desses dados não aumenta a probabilidade do atacante obter qualquer umas das chaves K_n, o ataque não seria bem sucedido. É importante frisar que a segurança desse sistema depende da impossibilidade de descoberta do segredo Z que por sua vez depende do pleno funcionamento do módulo criptográfico com tecnologia *tamper proof*

3. **O atacante pode capturar o nó móvel, mas não os nós sensores e extrair o conteúdo de sua memória:** como demostrado em [19], o conhecimento do conteúdo da memória do nó móvel, ou seja o conhecimento de $\{K_1 \oplus R, K_2 \oplus R, \ldots, K_n \oplus R\}$ não aumenta a probabilidade do atacante em descobrir qualquer uma das K_n chaves.

4. **O atacante pode capturar e acessar o conteúdo das memórias de qualquer nó sensor e também do nó móvel:** como demonstrado em [11], o conhecimento do conteúdo da memória do nó móvel associado ao conhecimento do conteúdo das memórias dos nós sensores, não aumentam a probabilidade do atacante obter qualquer uma das chaves K_n.

Pode ser argumentado que a abordagem de Kazienko e Albuquerque compartilham a desvantagem do sistema de Oliveira e Barros com relação a necessidade do uso de tecnologia

tamper proof. Entretanto, na abordagem de Kazienko e Albuquerque, apenas o conteúdo referente aos bits da sequência Z precisam ser protegidos, ao contrário do sistema de Oliveira e Barros onde a proteção precisa englobar toda a memória.

No que diz respeito ao desempenho, uma vez que a abordagem de Kazienko e Albuquerque faz com que seja necessário que cada nó sensor da rede execute uma operação de ou-exclusivo para obter a chave do nó com o qual ele deseja se comunicar, é possível que haja alguma sobrecarga na quantidade de processamento realizado nos nós sensores. Entretanto, a utilização de um módulo criptográfico reduz esse problema, já que a execução dos algoritmos de criptografia das chaves ocorrem em uma região dedicada do *hardware*.

Quanto ao consumo de memória nos sensores, o armazenamento de $K_n \oplus Z$ ao invés de K_n não representa um aumento no consumo já que $|K_n| = |K_n \oplus Z|$, onde $|x|$ representa o módulo de x.

Por fim, a única mensagem extra introduzida por este sistema é a mensagem de alarme que é enviada apenas esporadicamente e assim minimiza a possibilidade de contribuir para o congestionamento da rede.

CAPÍTULO 4 - METODOLOGIA

Para avaliar o sistema proposto em [11], foi desenvolvida a VITORIA, uma aplicação que simula a distribuição de chaves através do processo de codificação de redes entre dois sensores que desejam se comunicar. Após receber as chaves, os nós sensores as utilizam para se comunicar de forma encriptada garantindo assim o serviço de confidencialidade de mensagens. Para o desenvolvimento desta aplicação, foi utilizado o sistema operacional TinyOs [15], a linguagem de programação NesC [7] e o simulador TOSSIM [13], que acompanha a distribuição do TinyOS.

Com o objetivo de se obter uma análise rápida e automatizada das informações de tempo geradas pelo programa durante a simulação, foi desenvolvida também o VSLA (VITORIA Simulation Log Analyser), uma aplicação na linguagem Java que a partir de dois arquivos texto com dados referentes a simulação, gera como saída o atraso médio proporcionado pela execução dos algoritmos utilizados para recuperação das chaves em cada nó sensor e para encriptar/decriptar as mensagens trocadas pelos mesmos.

Para o desenvolvimento do VSLA foi utilizada a IDE *open source* NetBeans que facilita a criação de interface com o usuário tornando esta tarefa mais rápida e direta. O VSLA, assim como o VITORIA, é descrito detalhadamente no Capítulo 5.

Além das simulações, uma versão alterada da VITORIA foi desenvolvida para operar em uma rede de sensores real com apenas dois nós. Devido a limitação no número de nós, essa versão alterada não possui a etapa de distribuição de chaves de forma que os sensores já conhecem as chaves um do outro previamente. O armazenamento das chaves encriptadas com o segredo Z permanece igual a VITORIA original. Para esse experimento, foram utilizados dois sensores do tipo TelosB (Figura 8) da marca Crossbow cujas características são:

- Rádio compatível com o padrão IEEE 802.15.4/ZigBee com 250 kbps de taxa de dados;
- Microcontrolador TI MSP430 com *clock* de 8 MHz e 10 kB de RAM;
- Memória Flash com 1 MB;
- Sensores de temperatura, luminosidade e umidade integrados.

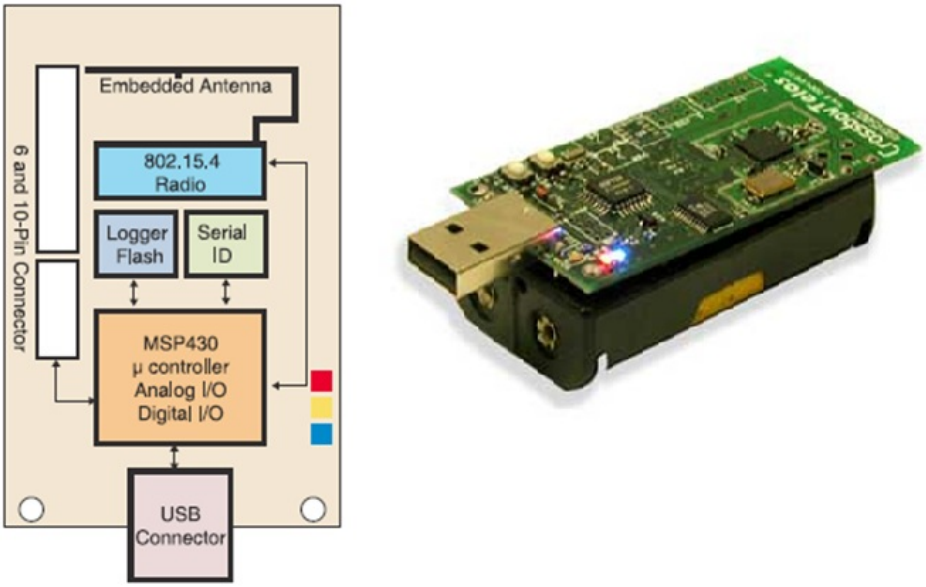

Figura 8: Imagem e diagrama esquemático de um sensor do tipo TelosB.

As sessões a seguir têm o objetivo de explicar brevemente as características mais importantes de cada uma das ferramentas, ambientes e plataformas utilizadas no desenvolvimento das aplicações propostas neste trabalho.

4.1 A LINGUAGEM DE PROGRAMAÇÃO NESC

A linguagem de programação NesC é uma extensão da linguagem C projetada para gerar código otimizado para sistemas embarcados como os nós sensores em uma RSSF. Todas as aplicações em NesC são desenvolvidas através da construção e reutilização de pedaços de código denominados componentes. Os componentes são divididos em duas categorias: Módulos e Configurações. Os módulos contém a implementação dos algoritmos propriamente ditos através de declarações de variáveis, funções, etc. As configurações são utilizadas para se conectar componentes de forma a montar componentes maiores e mais complexos. Além dos módulos e configurações, também existem as interfaces que são compostas de assinaturas de funções (sem implementação) e podem ser providas ou utilizadas pelos componentes. As funções contidas nas interfaces podem representar ações ou eventos onde as ações devem ser implementadas pelo componente que prove a interface enquanto que os eventos devem ser implementados pelo componente que a usa. As interfaces implementadas por um componente são o único meio de se acessar tal componente [7].

Para facilitar o desenvolvimento de aplicações na linguagem NesC, foi utilizado o IDE

Eclipse que após configurado com o *plug-in* Yeti 2 [8], torna-se um ambiente de desenvolvimento contendo realçamento de erros de sintaxe em tempo de codificação além do modo gráfico onde é possível ver o diagrama de dependência entre os componentes do sistema que está sendo desenvolvido, dentre outros benefícios.

4.2 O SISTEMA OPERACIONAL TINYOS

O sistema operacional TinyOs, desenvolvido na linguagem NesC, foi projetado para auxiliar o desenvolvimento de aplicações para as RSSF. Sua filosofia se distingue radicalmente dos sistemas operacionais tradicionais visto que ele não reside nos sensores de forma independente das aplicações, como por exemplo ocorre com o Windows e o Unix. Na verdade, quando um programa é desenvolvido, ele utiliza componentes que representam abstrações de componentes reais do hardware, para realizar as atividades desejadas. Assim, o TinyOs pode ser considerado um *framework* para desenvolvimento de aplicações para sensores.

Os componentes que constituem o TinyOs são ordenados de forma hierárquica sendo aqueles que se comunicam diretamente com o hardware as folhas da árvore. Dessa forma, torna-se possível estender facilmente o TinyOs para acomodar novos modelos de microcontroladores sem a necessidade de modificar os programas existentes já que os componentes adicionados poderiam ser ligados a componentes de mais alto nível já utilizados por outros programas.

O desenvolvimento de um sistema auxiliado pelo TinyOs consiste em criar componentes através da utilização de outros componentes já disponíveis no *Framework*. Por exemplo, no desenvolvimento da aplicação proposta neste trabalho, foi desenvolvido um componente que troca mensagens encriptadas entre os sensores da rede. A lógica desta aplicação foi desenvolvida, entretanto a maneira pela qual um sensor envia realmente um pacote para a rede é abstraída através do uso de componentes específicos para comunicação presentes no TinyOs.

Uma característica muito importante no TinyOs é que nenhum de seus componentes possui chamadas de função bloqueantes, o que significa que todas as chamadas de função que dependem da prontidão de um componente externo são implementadas através do paradigma *split-phase*. Isso significa que uma função como por exemplo enviarMensagem() que depende da prontidão do rádio do nó sensor, é dividida em uma função e um evento a saber, enviarMensagem() e enviarMensagemConcluida(). Quando a função enviarMensagem() é chamada, a tarefa de enviar a mensagem é passada ao componente que a implementa e o programa chamador pode continuar a realizar outras tarefas. Quando o componente finalmente termina o envio da mensagem ou um erro ocorre, ele invoca o evento enviarMensagemConcluida() do componente

que requisitou a ação. Dessa forma, o TinyOs evita o desperdício de ciclos de processador que ocorreria caso este entrasse no processo de espera ocupada (*busy wait*).

O TinyOs é um sistema gratuito e *open source* e pode ser baixado em [14]. Para este trabalho, foi escolhido obter o TinyOs através de uma versão pré-pronta do sistema operacional Xubuntos, que é uma distribuição do Linux. Para rodar esta versão, foi utilizada a máquina virtual VMWare Player V. 3, que oferece a possibilidade de executar uma máquina virtual ou ainda criar uma nova. Uma vez configurado na máquina virtual, o Xubuntos já possui instalado todos os recursos necessários para iniciar o desenvolvimento de uma aplicação com o TinyOs. Além da versão do Xubuntos, existe também a possibilidade de instalar o TinyOs nos sistemas operacionais Linux e Windows manualmente. Este modo entretanto, torna-se muitas vezes confuso e propenso a erros já que são muitos os arquivos necessários além da necessidade de se configurar algumas variáveis de ambiente o que pode ser desafiador para um usuário iniciante.

Neste trabalho, foi utilizado como referência a documentação sobre o TinyOs encontrado em [14] e também o livro [12].

4.3 O SIMULADOR TOSSIM

O TOSSIM é um simulador de RSSF que vem junto com o ambiente do TinyOs [13]. Para que uma simulação seja realizada, é necessário, primeiramente, definir a topologia da rede que se quer simular e o modelo de ruídos do canal. A topologia estabelece os identificadores de cada nó, os enlaces que a rede deve ter e a potência de cada um desses enlaces. O modelo de ruídos é utilizado para simular os ambientes onde uma RSSF deve operar.

Após o estabelecimento da topologia, é necessária a construção de um *script* em Python (para a versão que acompanha o TinyOs 2.x) que define as diretrizes de como a execução da simulação irá proceder.

Para que possam ser obtidas informações sobre a execução de um programa na simulação, o comando dbg pode ser utilizado nos programas escritos em NesC. O comando dbg imprime um determinado dado em um dispositivo de saída como a tela ou um arquivo texto. O dispositivo de saída utilizada é definida através da criação de um canal no *script* em Python. O canal possui um nome e este deve ser passado como argumento para a função dbg no programa em NesC.

CAPÍTULO 5 - O SISTEMA DESENVOLVIDO

O objetivo deste trabalho é implementar o sistema proposto por Kazienko e Albuquerque em [11] e avaliar tal implementação quanto a sobrecarga de processamento e consumo de memória. Entretanto, a imposição de um módulo criptográfico nos nós sensores torna sua implementação muito complicada dada a dificuldade de encontrar sensores equipados com tais dispositivos.

Por essa razão, foi desenvolvido neste trabalho um modelo simplificado baseado no trabalho de Kazienko e Albuquerque que elimina o uso do módulo criptográfico. Comparando a Figura 7 mostrada no Capítulo 3 e a Figura 9, que descreve o modelo simplificado, podemos notar as seguintes diferenças:

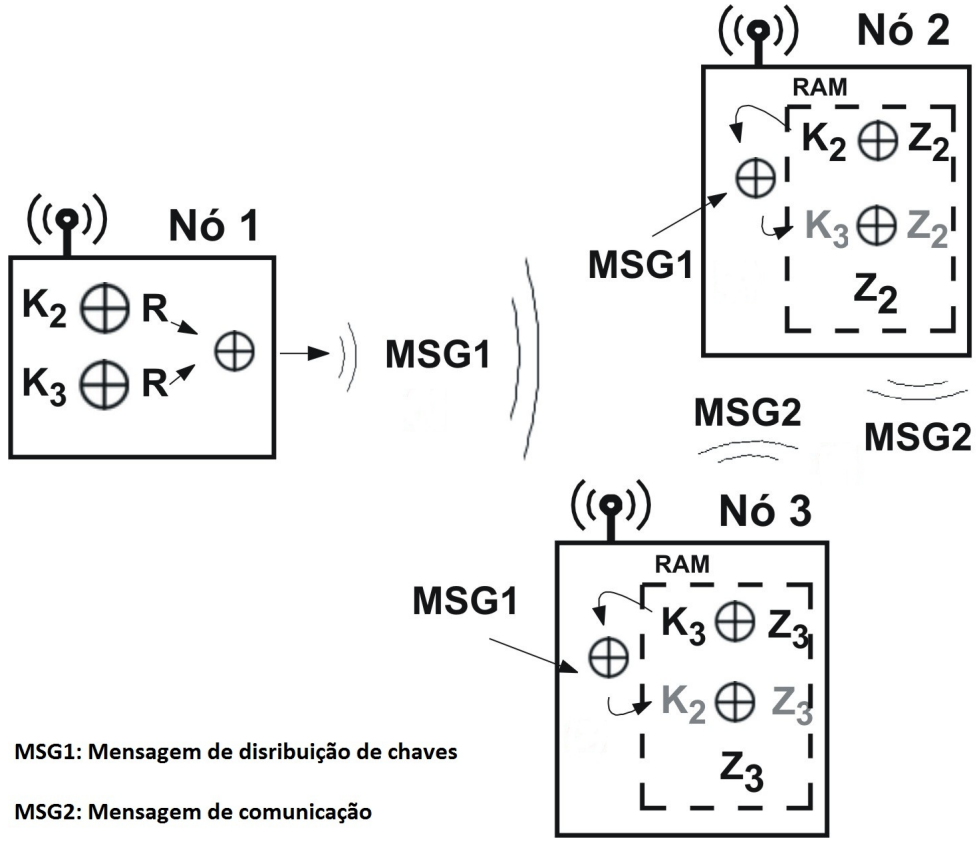

Figura 9: Sistema simplificado de distribuição e armazenamento de chaves.

1. **Ausência do Módulo Criptográfico (MC):** o modelo simplificado armazena o segredo Z e as chaves na memória RAM dos nós sensores. Além disso, os algoritmos de encriptação

e decriptação de chaves que eram executados dentro do MC, agora são executados dentro do microcontrolador do sensor. A remoção do MC permitiu a implementação do sistema proposto neste trabalho, mas por outro lado reduziu criticamente a segurança do sistema, já que o mesmo deixou de ser resistente a ataques de captura como era o sistema original. Além disso, o desempenho também é reduzido já que a tarefa de encriptar e decriptar chaves passa a ser do microcontrolador e não mais do MC.

2. **RSSF com número limitado de nós:** a abordagem de Kazienko e Albuquerque foi feita para acomodar um número arbitrário de nós. Entretanto, para as necessidades deste trabalho, foi suficiente a implementação de uma RSSF com apenas três nós sensores, sendo que um deles atua como o nó móvel descrito no trabalho de Oliveira e Barros [19].

3. **Apenas uma chave por sensor:** diferentemente do que foi proposto no esquema de distribuição de Oliveira e Barros, onde cada nó sensor detinha um número *n* de chaves e seus respectivos identificadores, neste trabalho cada nó sensor detém apenas uma chave. Essa simplificação têm a desvantagem de enfraquecer a segurança já que facilita a criptoanálise de um texto cifrado, mas em compensação simplifica o programa já que remove a necessidade de realizar uma escolha sobre qual chave utilizar a cada mensagem trocada. Uma vez que existe apenas uma chave na memória de cada sensor, esta não precisa mais de um identificador próprio já que o identificador do nó sensor identifica também a sua respectiva chave.

Apesar do programa VITORIA, descrito mais adiante, ser baseado no modelo simplificado descrito anteriormente, seria teoricamente simples introduzir mecanismos para estende-lo ao modelo original proposto por Kazienko e Albuquerque. A seguir, é feita uma descrição de como cada simplificação apresentada anteriormente pode ser expandida individualmente.

1. **Ausência do Módulo Criptográfico (MC):** esta é sem dúvida a simplificação mais difícil de expandir, já que é necessária uma modificação no hardware dos sensores para acomodar um módulo criptográfico.

2. **RSSF com número limitado de nós:** o aumento do número de nós no VITORIA versão de simulação seria um pouco complicado já que seria necessário a inserção de tantas instruções *If* quanto o número de nós adicionados para acomodar os novos sensores. Já no VITORIA versão de implementação em sistemas reais, essa alteração seria teoricamente simples, pois bastaria apenas instalar o programa em cada nó da rede, modificando apenas as chaves de cada nó juntamente com seus identificadores e o identificador do nó.

3. **Apenas uma chave por sensor:** com o objetivo de aumentar o número de chaves por nó, seria necessário utilizar uma estrutura de dados como uma matriz ou uma tabela hash (*hash table*) de forma que o identificador de cada chave fosse utilizado como índice e a chave codificada $K_i \oplus Z_i$ seria o dado. Além dessa modificação, também seria necessária a inclusão de um campo no pacote destinado a guardar o identificador da chave utilizada para encriptar a mensagem. Dessa forma, o destinatário da mensagem saberia qual chave deve ser usada na processo de decriptação.

O modelo proposto por esse trabalho têm ainda mais uma diferença em relação ao trabalho proposto por Kazienko e Albuquerque. Neste último, quando um nó A precisava enviar uma mensagem para o nó B, ele obtinha uma de suas chaves através da realização de um ou-exclusivo, como descrito no Capítulo 3, e então a utilizava para encriptar a mensagem. Quando o nó B recebia a mensagem, ele utilizava o identificador da chave presente no pacote para saber qual das chaves de A utilizar. Logo após, a operação de ou-exclusivo era realizada para obter a chave em texto claro de A para que esta então pudesse ser utilizada para decriptar a mensagem. Esse processo funciona bem se o nó B conhece todas as chaves do nó A, o que não ocorre na prática, pois o nó móvel distribui um par de chaves por vez.

Com o objetivo de resolver essa limitação sem a adoção de métodos mais complexos onde um nó precisa gerenciar quais de suas chaves são conhecidas por um outro determinado nó, o modelo simplificado utiliza uma das chaves do nó destinatário conhecidas pelo nó remetente para encriptar a mensagem. Dessa forma, é possível garantir que o nó destinatário conseguirá decriptar a mensagem, bastando para isso utilizar o identificador presente no pacote para indexar sua tabela de chaves. Essa medida não causa qualquer efeito sobre o modelo simplificado, já que neste cada sensor tem apenas uma chave, mas por outro lado facilita o processo de expansão do número de chaves por sensor.

Nas próximas sessões são apresentadas os programas VITORIA e VSLA, levando em conta seus algoritmos e estruturas.

5.1 O PROGRAMA VITORIA

O VITORIA é um programa escrito em NesC utilizando o sistema operacional TinyOs que tem a função de distribuir chaves criptográficas par-a-par como descrito por Oliveira e Barros e armazenar tais chaves de forma segura como descrito no modelo simplificado abordado anteriormente. Após o processo de distribuição e armazenamento das chaves, os sensores então podem se comunicar através da troca de mensagens criptografadas ou não.

5.1.1 MODOS DE FUNCIONAMENTO

O VITORIA possui dois modos de funcionamento: modo normal e modo de distribuição. No modo normal, o VITORIA age como um nó tradicional da rede e apenas envia e recebe mensagens criptografadas. No modo de distribuição, o VITORIA age como o nó móvel e efetua apenas uma operação de distribuição de chaves para os outros dois nós da rede.

Por padrão, se o VITORIA for instalado em um sensor com identificador 1, então o programa passa a operar em modo de distribuição. Caso contrário, o programa opera em modo normal. Dessa forma, a escolha entre qual modo de operação adotar é automática e inerente ao sensor. A seguir estão descritos os ciclos de vida do programa em cada um dos modos de operação.

- **Modo Normal:** quando o sensor é ligado, um evento de Boot é disparado pelo sistema operacional TinyOs. A partir de então o Algoritmo 1 é executado, carregando a chave e o segredo do nó sensor em sua memória. Em seguida, um temporizador é executado e gera um evento a cada 1 ms onde em cada um deles é feita uma tentativa de envio de uma mensagem com tamanho de 224 bits. Se a tentativa de envio for feita antes que o nó tenha recebido a chave do destinatário da mensagem através do processo de distribuição de chaves, o envio é abortado. As tentativas de envio continuam a cada 1 ms indefinidamente até que o sensor seja desligado.

- **Modo de Distribuição:** a operação de boot funciona da mesma forma que no modo anterior entretanto, como pode ser visto no Algoritmo 1, neste modo todas as chaves são carregadas no sensor e ao invés da inicialização do temporizador, neste modo é executada a função de envio de chaves. Após as chaves terem sido enviadas, o nó móvel desliga seu rádio para economizar bateria e nada mais faz.

5.1.2 TIPOS DE MENSAGEM

O VITORIA utiliza dois tipos de mensagens diferentes: mensagens de distribuição e mensagens de comunicação.

Uma mensagem de distribuição é utilizada pelo nó móvel e pelos nós sensores para realizar a distribuição das chaves entre os outros dois nós da rede. A Figura 10 mostra o formato de uma mensagem de distribuição. Cada campo possui uma função específica importante para o processo de distribuição de chaves:

Algoritmo 1 CarregaChaves (*ID*)

```
1:  if (ID == 1) then
2:     //O nó móvel conhece todas as chaves;
3:     R = 156782168906551636
4:     k2 = 184444073709551615 ⊕ R;
5:     k3 = 178953296589432169 ⊕ R;
6:     sendKeys();
7:  else
8:     if (ID == 2) then
9:        //O nó sensor com identificador 2 conhece a chave K2 e o segredo Z2
10:       k2 = 184444073709551615;
11:       z2 = 169874345676543321;
12:       k2 = z2 ⊕ k2;
13:    else
14:       if (ID == 3) then
15:          //O nó sensor com identificador 3 conhece a chave K3 e o segredo Z3
16:          k3 = 178953296589432169;
17:          z3 = 172123489098765447;
18:          k3 = z3 ⊕ k3;
19:       end if
20:    end if
21:    startTimer();
22: end if
```

- O campo **ID1** contém o identificador de uma das chaves utilizadas na operação de ou-exclusivo. No caso específico do VITORIA, esse campo é preenchido com o identificador de um sensor e não da chave, já que cada sensor tem apenas uma chave.

- O campo **ID2** contém o identificador da outra chave utilizadas na operação de ou-exclusivo. No caso específico do VITORIA, esse campo também é preenchido com o identificador de um sensor e não da chave.

- O campo **XOR** Contém o ou-exclusivo entre as duas chaves definidas pelos identificadores ID1 e ID2.

Uma mensagem de comunicação é utilizada somente pelos nós sensores e serve para enviar e receber dados de um sensor para outro. A Figura 11 mostra o formato deste tipo de mensagem e os seus campos são descritos a seguir.

- O campo Status é utilizado para identificar se uma determinada mensagem está criptografada ou não. Este campo é útil para fazer com que o VITORIA envie tanto mensagens criptografadas quanto mensagens em texto claro. Assim é possível decidir quando uma mensagem recebida deve ou não ser decriptada.

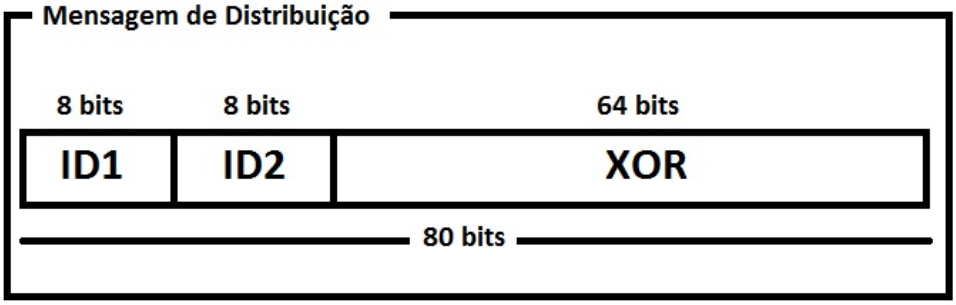

Figura 10: Formato de uma mensagem de distribuição de chaves.

- O campo **Sender** é utilizado para identificar o sensor remetente de uma mensagem.
- O campo **Seq** contém o número de sequência de uma determinada mensagem. É útil para identificar uma determinada mensagem.
- O campo **Message** contém a mensagem propriamente dita. Este campo é fixo em 16 bytes, mas a mensagem pode não necessariamente utilizar os 16 bytes. Em uma comunicação por mensagens criptografadas, esse campo é encriptado.
- O campo **Length** indica o número de bytes do campo Message que são realmente dados e não lixo. Esse campo é importante pois os algoritmos de encriptação e decriptação o utilizam para evitar de processar além do tamanho real da mensagem. Em uma comunicação por mensagens criptografadas, esse campo é encriptado.

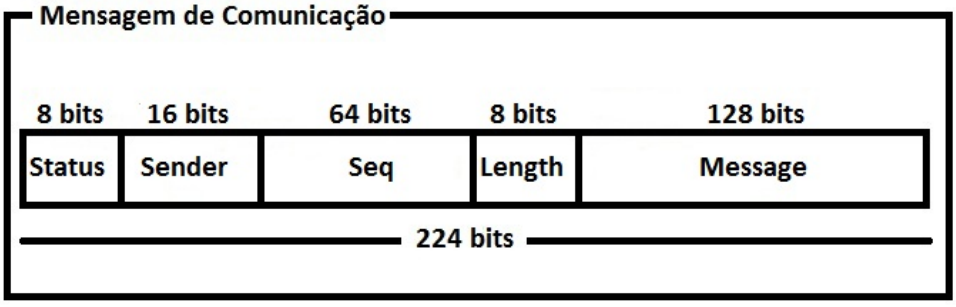

Figura 11: Formato de uma mensagem de comunicação.

5.1.3 PROCESSO DE DISTRIBUIÇÃO DE CHAVES

A distribuição de chaves ocorre logo quando o sensor com identificador 1 é ligado. Inicialmente uma mensagem de distribuição é criada com os campos ID1 e ID2 contendo os identificadores 2 e 3. Em seguida, uma operação de ou-exclusivo é realizada da mesma forma que o descrito na abordagem de Kazienko e Albuquerque. O resultado da operação de ou-exclusivo é

então armazenada no campo XOR da mensagem. Logo após a mensagem é enviada em *Broadcast* para a rede.

5.1.4 PROCESSO DE ENVIO DE MENSAGENS

No VITORIA, existem duas formas de se mandar uma mensagem: modo criptografado e modo em texto claro. Para trocar de um modo para o outro, basta modificar o valor de uma constante no sistema.

O envio de uma mensagem no modo texto plano exige apenas a criação de uma mensagem de comunicação com o campo Status contendo o valor 1 (Texto Claro), o campo **Sender** contendo o identificador do nó que está enviando a mensagem, o campo **Seq** contendo o valor de uma variável que é incrementada toda vez que uma mensagem é enviada com sucesso, o campo **Message** contendo a mensagem a ser enviada e o campo **Length** contendo o numero de bytes do campo mensagem.

O envio de uma mensagem no modo criptografado também exige a criação de uma mensagem de comunicação como descrito acima, mas o conteúdo dos campos **Status, Message e Length** serão diferentes. O campo **Status** será preenchido com o valor 2 (Encriptado). Antes de prosseguir com a descrição do conteúdo dos campos **Message e Length**, é necessário entender o funcionamento dos Algoritmos 2 e 3.

Algoritmo 2 encryptMessage(key, message, msgLength)

```
1:  aux = key;
2:  j = 0;
3:  for i = 0; i < msgLength; i++ do
4:      keyPortion = (uint8_t)aux;
5:      message[i] = message[i] ⊕ keyPortion;
6:      if (j = 7) then
7:          aux = key;
8:          j = 0;
9:      else
10:         aux >>= 8;
11:         j++;
12:     end if
13: end for
```

O Algoritmo 2 é utilizado para criptografar uma determinada mensagem. Uma vez que as chaves criptográficas utilizadas no VITORIA tem 64 bits e uma mensagem é um vetor de caracteres, cada um com 1 byte, não é possível realizar a operação de ou-exclusivo diretamente entre a chave e cada carácter, pois se isso fosse feito, o resultado teria mais de 8 bits e a mensagem

teria que ser expandida, aumentando assim o tamanho do pacote. Ao invés disso, a instrução $keyPortion = (uint8_t)aux$ captura os 8 bits menos significativos do conteúdo da variável *aux* que inicialmente contém a chave. Essa porção de 8 bits da chave é então utilizada para fazer um ou-exclusivo com um carácter da mensagem. Após esse processo, se a mensagem ainda tem mais caracteres não criptografados, é realizado um deslocamento de 8 bits para direita na variável *aux* com o objetivo de na próxima vez que a instrução $keyPortion = (uint8_t)aux$ for executada, os próximos 8 bits sejam capturados. Se o número de bytes da chave se esgotar antes da mensagem, então a variável *aux* é reiniciada com o valor original da chave e a variável j volta ao valor 0 para indicar que agora será dado o primeiro deslocamento. Dessa forma, o algoritmo gera um ciclo que se repete a cada 7 deslocamentos de bits.

O Algoritmo 3 é uma versão compacta do Algoritmo 2. Neste caso, o algoritmo é utilizado para criptografar o tamanho da mensagem e como o campo de tamanho de mensagem tem apenas 8 bits, é necessário extrair apenas a primeira parcela de 8 bits da chave para então realizar o ou-exclusivo. A utilização desse algoritmo é importante para evitar que até mesmo o tamanho de uma mensagem seja conhecida pelo atacante.

Algoritmo 3 encryptLength(key, *length)

1: $keyPortion = (uint8_t)aux$;
2: $*length = *length \oplus keyPortion$;

Antes que qualquer algoritmo de criptografia possa ser executado, é necessário obter a chave que será utilizada. Para tanto, basta fazer ou ou-exclusivo entre $K_i \oplus Z_i$ e Z_i que são dados conhecidos pelo sensor. De posse da chave, os Algoritmos 2 e 3 podem ser utilizados.

Após a execução do Algoritmo 2, a mensagem estará criptografada e será então carregada no campo **Message** da mensagem. Em seguida, o Algoritmo 3 é executado e produz uma versão criptografada do tamanho da mensagem e é então carregado no campo **Length**. É importante que o Algoritmo 2 receba o tamanho da mensagem em texto claro como parâmetro, pois caso contrário o mesmo não funcionará.

5.1.5 PROCESSO DE RECEPÇÃO DE MENSAGENS

Quando um sensor recebe uma mensagem de comunicação, é necessário analisar o valor do campo Status para saber como proceder. Se o valor do Status for 1 (Texto Claro), bastará então verificar o conteúdo do campo **Message** para saber do que a mensagem se trata. Se por outro lado o valor do campo Status for 2 (Encriptada), será necessário decriptar a mensagem antes de poder entende-la. Primeiramente, assim como no processo de encriptação, é necessário

obter a chave através da execução do ou-exclusivo entre $K_i \oplus Z_i$ e Z_i.De posse da chave, basta usar os mesmo algoritmos usados na encriptação, mas na ordem inversa. Assim, primeiro é necessário usar o Algoritmo 3 passando como parâmetros a chave e o valor contido no campo **Length** da mensagem. Assim, após obter o tamanho da mensagem em texto claro, pode-se utilizar essa informação, juntamente com a chave e o valor do campo **Message** da mensagem como parâmetros do Algoritmo 2 para obter a mensagem em texto claro, que então pode ser entendida.

5.2 VITORIA MODIFICADO

O VITORIA da maneira que foi desenvolvido, agrega as funções de nó móvel e troca de mensagens em um único programa. Para fins de simulação essa estrutura funciona bem, já que diferentes nós precisam desempenhar diferentes funções executando o mesmo código. Entretanto, instalar o vitória em sensores reais, apesar de possível, faz com seja gasta mais memória do que o necessário, já que cada sensor precisa armazenar tanto a lógica de distribuição quanto a de troca de mensagens.

Para sanar esse problema, o VITORIA foi modificado de forma a dividir as funções de distribuição e comunicação em duas partes distintas. Como consequência, o sensor que irá fazer o papel de nó móvel não será mais definido pelo seu identificador e sim pelo tipo de programa que ele irá executar.

Uma outra modificação pode ser feita com o objetivo de eliminar a etapa de distribuição de chaves. Essa modificação tem sentido no trabalho aqui descrito, pois o número de sensores disponíveis para experimentos era restrito (apenas dois). Dessa forma, todos os nós precisavam estar envolvidos no processo de troca de mensagens e não haveria nenhum nó responsável por efetuar a distribuição.

Para efetuar esta última modificação, bastou utilizar o processo de pré-distribuição de chaves, onde cada sensor é carregado com todas as chaves que irá precisar antes de ser integrado à rede.

5.3 O PROGRAMA VSLA

O VSLA (VITORIA Simulation Log Analyser) é um programa desenvolvido para analisar os *logs* gerados na simulação do VITORIA, de forma que seja possível extrair diretamente informações como maior latência, latência média, número de pacotes perdidos, etc. A figura 12

mostra a interface principal do VSLA.

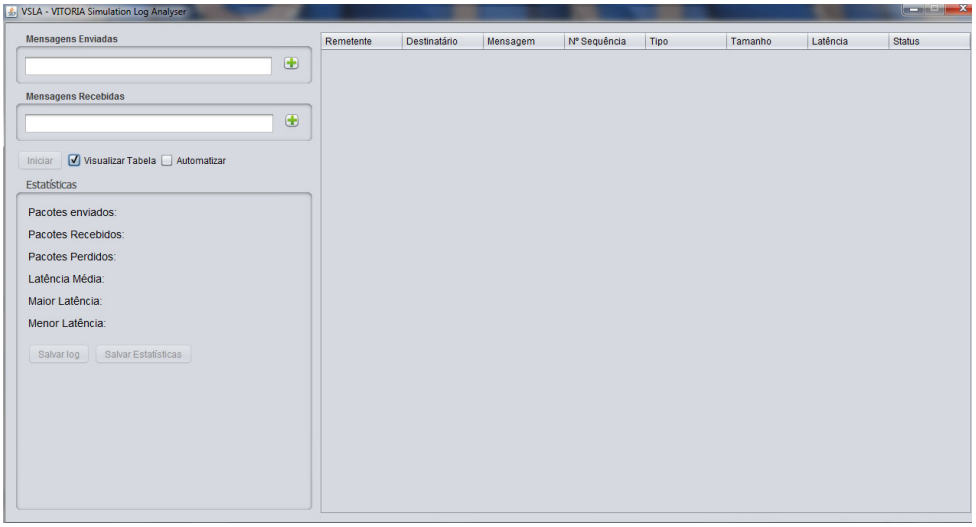

Figura 12: Interface Principal do Programa VSLA.

Para entender o funcionamento do VSLA primeiramente é necessário entender como funciona o processo de simulação do VITORIA. Sempre que uma mensagem de comunicação no modo criptografada é recebida ou enviada no VITORIA, uma nova linha é adicionada em um arquivo de *log* contendo os valores de todos os campos da mensagem discutidos anteriormente, mais o tempo em que aquela mensagem foi recebida ou enviada. Esse tempo é contado a partir do momento que o sensor é inicializado e os dois sensores precisam ser sincronizados de forma que, se uma mensagem foi enviada pelo sensor A no tempo 5 e recebida pelo sensor B no tempo 10, o resultado de tempo de chegada - tempo de envio reflita realmente o tempo decorrido no envio da mensagem. Para que tal sincronismo seja obtido na simulação, é possível estabelecer que todos os sensores devem iniciar (*boot*) ao mesmo tempo.

Através desse processo, são gerados então dois arquivos de *log*, um para mensagens enviadas e um outro para mensagens recebidas. Esses dois arquivos de *log* são usados como entrada para o VSLA que faz uma análise de ambos para gerar seus resultados.

Inicialmente, o VSLA varre o arquivo de mensagens recebidas coletando o identificador do nó remetente e o número de sequência do pacote. Cada uma das linhas do arquivo de mensagens enviadas é adicionada a um *hashtable* cujas chaves são a concatenação do identificador do nó remetente mais o caractere ”/”, mais o número de sequência da mensagem. Uma vez que existe uma cópia da variável incrementada, usada para formar o número de sequência das mensagens, em cada sensor, o seu valor sozinho não identifica globalmente a mensagem. Assim, a concatenação com o identificador do nó remetente tem a função de atingir esse objetivo.

Depois que o *hashtable* foi montado, o VSLA varre o arquivo de mensagens enviadas e

obtém de cada linha o número de sequência e o identificador do nó remetente da mensagem. Uma concatenação é feita então da mesma forma que no processo anterior e o resultado é usado para indexar o *hashtable* e obter a mensagem recebida que é correspondente a mensagem enviada. Se nenhum valor for encontrado no *hashtable* para aquela concatenação, isso significa que a mensagem foi perdida, já que uma mensagem no *log* de enviadas não tem correspondente no *log* de recebidas. Após encontrar a mensagem recebida correspondente, são extraídas as informações pertinentes e o atraso da mensagem é calculado através da subtração do tempo de chegada menos o tempo de envio.

O VSLA permite a opção de mostrar ou não as mensagens em uma tabela em sua interface gráfica. Essa opção é dada pois a execução do VSLA é mais rápida sem a exibição da tabela.

O VSLA permite ainda que o resultado da análise seja salvo no formato CSV, que pode ser aberto no Excel. As estatísticas obtidas só podem ser salvas no formato texto.

CAPÍTULO 6 - AVALIAÇÃO DO SISTEMA PROPOSTO

Com o objetivo de avaliar o desempenho do VITORIA, foram realizadas dez simulações, onde mensagens encriptadas com 224 bits foram trocadas entre os sensores 2 e 3 da rede. Uma vez que o processo de distribuição de chaves é idêntico ao descrito por Oliveira e Barros em [19], este não foi levado em consideração na análise de tempo descrita na próxima seção.

Além da análise de tempo, foi compilado para o sensor MicaZ uma versão do VITORIA que separa o código de distribuição de chaves do código referente a troca de mensagens. Dessa forma, foi possível observar o quanto de memória ROM (flash) e RAM o VITORIA consome em cada sensor, de acordo com o seu papel.

6.1 ANÁLISE DE TEMPO

Com o objetivo de avaliar o desempenho do VITORIA quanto a sobrecarga no tempo gasto para que uma mensagem enviada fosse recebida e decriptografada, foram realizadas, utilizando o simulador TOSSIM, trinta simulações com o VITORIA, dos quais dez só com troca de mensagens no modo criptografado e outras dez só com mensagens no modo de texto claro. Além disso, mais dez simulações adicionais foram feitas utilizando uma versão especial do VITORIA, onde as chaves, após serem recebidas, eram armazenadas sem a execução da operação de ou-exclusivo com o segredo Z. Essa versão especial corresponde ao modelo proposto por Oliveira e Barros em [19].

Para realizar as simulações com TOSSIM, é necessário informar o número máximo de eventos que se deseja simular. Cada vez que um evento é gerado no programa, este é inserido em uma pilha e posteriormente será processado pelo TOSSIM. Para obter um número razoável de mensagens trocadas, nas simulações descritas acima foi utilizado o número máximo de 10.000.000 de eventos onde cerca de 180.000 mensagens foram enviadas das quais cerca de 160.000 foram recebidas o que representa uma taxa de 11% de perda em média (causado por fatores como interferência, descarte de pacotes em filas de buffers, etc). A Tabela 2 mostra todos os parâmetros envolvidos nas simulações e os seus respectivos valores.

Para calcular o tempo total entre o envio de uma mensagem e o seu recebimento, incluindo

Tabela 2: Parâmetros de Simulação.

Parâmetro	Valor
Número de Eventos	10.000.000
Número de Simulações	10
Tamanho da Mensagem	8 bytes

o processo de encriptação e decriptação e os atrasos de propagação e transmissão, foi utilizado o programa VSLA, descrito no Capítulo 5. A partir do gráfico mostrado na Figura 13, é possível perceber que, como o esperado, na maioria das simulações a troca de mensagens utilizando a abordagem de Kazienko e Albuquerque levou mais tempo devido a necessidade da execução extra de operações de ou-exclusivo para se obter a chave tanto no processo de encriptação quanto no processo de decriptação de mensagens.

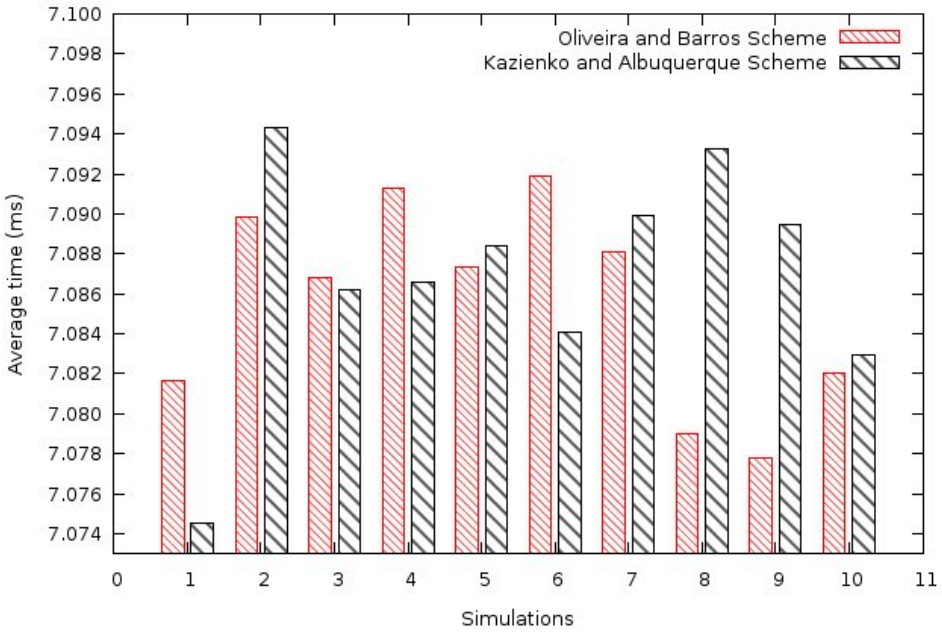

Figura 13: Comparação entre a média dos tempos de envio e recepção de mensagens em cada simulação entre a versão do VITORIA que implementa o modelo de Kazienko e Albuquerque [11] e a versão que implementa o modelo proposto por Oliveira e Barros [19].

Uma vez que dez simulações não é um número tão elevado, considerar apenas os resultados das simulações individuais pode ser um pouco impreciso. Assim, foi feita uma média entre os tempos apresentados em cada simulação para o esquema de Oliveira e Barros e o esquema de Kazienko e Albuquerque. Para ser ainda mais abrangente, também foram realizadas dez simulações com uma versão do sistema que apenas troca mensagens em texto claro ou seja,

não implementam nenhum algoritmo de criptografia e a média dos tempos para cada uma dessas simulações também foi calculada. A Figura 14 mostra um gráfico comparando as médias obtidas para os três esquemas, considerando as dez simulações realizadas para cada esquema.

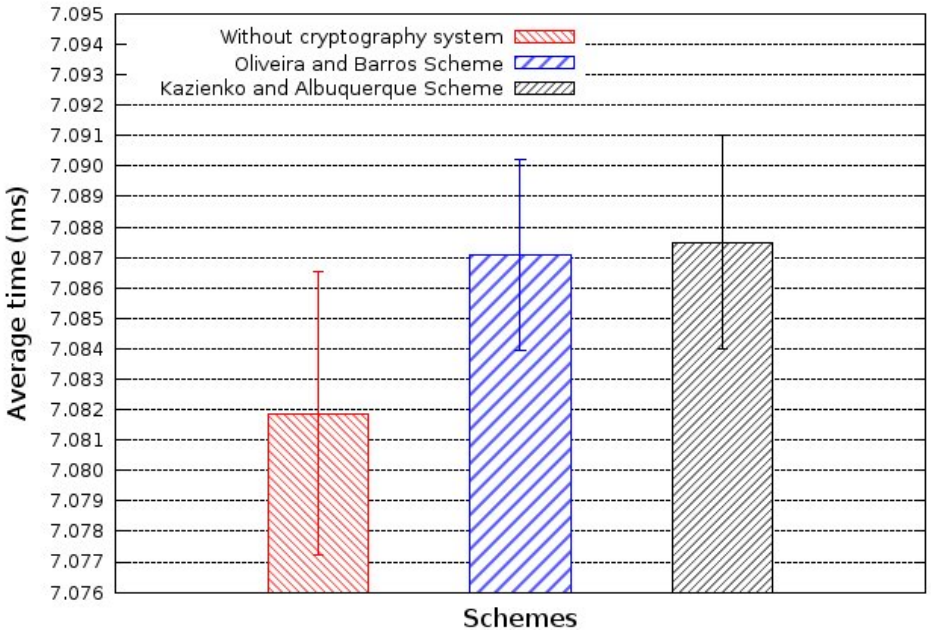

Figura 14: Comparação entre média dos tempos de envio e recebimento de mensagens obtidos em todas as simulações para os modelos de Kazienko e Albuquerque [11], Oliveira e Barros [19] e um modelo que não utiliza criptografia

Como pode ser observado, a diferença de tempo entre o modelo de Kazienko e Albuquerque e de Oliveira e Barros foi muito pequena (aproximadamente 407 ns). Se comparado ao modelo sem criptografia, o esquema de Oliveira e Barros apresenta uma diferença de tempo é de aproximadamente 5 microsegundos. Tais diferenças são muito pequenas e podem ser consideradas insignificantes.

6.2 CONSUMO DE MEMÓRIA

Para analisar o quanto o VITORIA consome de memórias ROM e RAM, foi utilizada sua versão modificada para instalação em sensores reais, conforme descrito nó tópico 5.2. Após compilar o VITORIA modificado para um sensor da plataforma mica2, observando as informações fornecidas pelo compilador sobre uso de memória RAM e ROM, chegou-se aos seguintes dados:

- **Consumo de memória para o código destinado ao nó móvel**

  1. **Memória RAM:** 235 bytes

2. **Memória ROM:** 10098 bytes

- **Consumo de memória para o código destinado a troca de mensagens**

1. **Memória RAM:** 598 bytes
2. **Memória ROM:** 19746 bytes

É importante ressaltar que os dados referentes ao consumo de memória RAM não incluem o consumo da pilha nem do heap. Além disso, a memória referida como ROM na verdade diz respeito a memória flash do sensor.

Se os dados de consumo de memória obtidos forem comparados a quantidade de memória RAM e Flash disponíveis nos modelos de sensores mostrados na Tabela 1, pode-se perceber que o VITORIA pode ser instalado sem problemas e no caso do Mica2, no módulo mais pesado (módulo de troca de mensagens), ainda restariam mais de 3 KB de memória RAM e mais de 108 KB de memória ROM (memória flash).

6.3 EXECUÇÃO EM SENSORES REAIS

Para avaliar a execução do VITORIA em sensores reais, foi instalado em dois sensores da plataforma TelosB, a versão do VITORIA que não utiliza distribuição de chaves (para mais informações sobre esta versão, ver o tópico 5.2). Para verificar se uma mensagem enviada de forma encriptada era recebida e decriptada de maneira correta, foi estabelecida uma mensagem conhecida pelos dois sensores e após um sensor receber uma mensagem e decripta-la, o mesmo comparava o resultado com a mensagem esperada e se fossem iguais, uma luz verde piscava caso contrário um luz vermelho piscava.

Após um minuto de execução (tempo suficiente para que um número razoável de mensagens pudessem ter sido trocadas), somente foi observado o ascendimento da luz verde, indicando assim que as mensagens estavam sendo encriptadas e decriptadas corretamente e que também estavam integras.

CAPÍTULO 7 - CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho implementou, utilizando o sistema operacional TinyOs, a linguagem de programação NesC e o simulador TOSSIM, o VITORIA, um sistema baseado na abordagem de Kazienko e Albuquerque [11] que tem o objetivo de prover distribuição e armazenamento seguros de chaves criptográficas em RSSF. Também foi desenvolvido o VSLA com o objetivo de realizar uma análise dos *logs* obtidos no processo de simulação. Por fim, foi feita uma análise da viabilidade de operação do VITORIA quanto ao consumo de memória e processamento.

Após a análise dos dados obtidos na avaliação do sistema desenvolvido neste trabalho, pode-se afirmar que para a maioria das aplicações que necessitam de segurança, o atraso imposto pelo armazenamento seguro da abordagem de Kazienko e Albuquerque [11] é muito pequeno em relação ao aumento de segurança que é proporcionado. O consumo de memória, como visto no Capítulo 6, é pequeno em relação a quantidade de memória disponível nos principais sensores utilizados nas aplicações atuais. Assim, concluímos que o VITORIA, que implementa um modelo simplificado da abordagem de Kazienko e Albuquerque, é viável tanto em relação ao tempo de comunicação (tempo que uma mensagem encriptada enviada leva para ser recebida e decriptada pelo sensor destinatário), quanto em relação ao consumo de memória RAM e Flash.

Apesar dos dados obtidos neste trabalho, ainda restam alguns outros pontos a serem discutidos antes de uma conclusão absoluta quanto a viabilidade de [11]. É necessário que se avalie o gasto de energia proporcionado pelo maior processamento, já que essa característica é de importância fundamental para o funcionamento das RSSF.

Um outro ponto importante é a segurança do Algoritmo 2 proposto neste trabalho. A utilização de simples operações de ou-exclusivo tornam o processamento mais rápido, mas por outro lado, as mensagem encriptadas com o mesmo podem ser criptoanalisadas utilizando técnicas como contagem de frequência de forma que a chave utilizada pode ser descoberta apenas a partir do conhecimento de textos cifrados, o que é muito fácil de se conseguir em canais sem fio.

Por fim, características como autenticação de sensores e nó móvel e integridade de mensagens poderiam ser implementadas. Além disso, o processo de distribuição de chaves também

poderia ser implementado de tal forma que fosse feito em múltiplos saltos, evitando assim a necessidade do nó móvel estar dentro do alcance de cada par de sensores da rede, o que na maioria das vezes não é possível.

APÊNDICE A – O PROGRAMA VITORIA PARA SIMULAÇÕES

Arquivo secM.nc

```
#include "Timer.h"
#include "sec.h"
#include "stdlib.h"
#include "stdio.h"

module secM
{
  uses
  {
     interface Boot;
     interface Receive as RegularReceiver;
     interface Receive as KeyReceiver;
     interface AMSend as KeySender;
     interface AMSend as RegularSender;
     interface Timer<TMilli> as MilliTimer;
     interface Timer<TMilli> as SecondCounter;
     interface SplitControl as AMControl;
     interface Packet;
     interface AMPacket;
  }
}
implementation
{
   message_t packet;

   //Bloqueia a distribuição de chaves enquanto outra distribuição está em andamento
   bool lockedKey;

   //Bloqueia o envio de mensagens enquanto outro envio está em andamento
   bool lockedRegular;

   //Indica se as chaves já foram recebidas pelo mote
   bool keyReceived = FALSE;

   //chave do mote 1 (no da rede)
   uint64_t k2 ;
```

```
//chave do mote 2 (no da rede)
uint64_t k3 ;

//contador sequencial utilizado para inicializar o gerador de números aleatórios
int secCount = 0;

//Número mínimo de mensagens enviadas sem criptografia
const bool ENABLE_ENCRYPTION = TRUE;

//Lista de mensagens possíveis. Quando um pacote é enviado, uma mensagem dessa lista é selecionada aleatoriamente
uint8_t messageList[3][MSG_MAX_LEN];

//Segredo utilizado para prover o armazenamento seguro das chaves nos sensores
uint64_t z2,z3;

uint64_t R;

//Sequeciador das mensagens
uint64_t count = 0;

//Povoa a lista de mensagens
void buildMessageList()
{
   strcpy(messageList[0], "Mensagem");
   strcpy(messageList[1], "Security");
   strcpy(messageList[2], "NescProg");
}

//Obtém uma mensagem aleatória
void getMessage(uint8_t message[MSG_MAX_LEN])
{
   int next;
   srand(secCount);
   next = (rand() % 3);
   strcpy(message,messageList[next]);
}

//Encripta uma mensagem
void encryptMessage(uint64_t key, uint8_t message[MSG_MAX_LEN], uint32_t msgLength)
{
   uint64_t aux = key;

  //Porção de 8 bits da chave key
  uint8_t keyPortion;
  int i,j;
  j = 0;
  for(i = 0; i < msgLength; i++)
  {
     //Extrai os 8 bits menos significativos de aux e atribui à keyPortion
 keyPortion = (uint8_t)aux;
 //Faz o xor entre o i-ésimo caracter da mensagem com a porção atual da chave
 message[i] = message[i] ^ keyPortion;
 if(j == 7)//Todos os bytes da chave foram usados
 {
```

```
    //aux é reiniciado com os bits originais da chave
    aux = key;
    j = 0;
   }
   else//o próximo byte da chave precisa ser extraido
   {
      //Faz um shift de 1 byte a direita
      aux>>= 8;
      j++;
   }
  }
}

//Encripta o tamanho da mensagem
void encryptLength(uint64_t key, uint8_t *length)
{
   //Extrai os 8 bits menos significativos de aux e atribui à keyPorti
  uint8_t keyPortion = (uint8_t)key;

  //Faz o xor entre o tamanho da mensagem e a porção de 8 bits da chave
  *length = *length ^ keyPortion;
}

// Evento de boot do mote
event void Boot.booted()
{
   buildMessageList();
   call SecondCounter.startPeriodic(1);
   call AMControl.start();
}

//Envia uma mensagem de distribuição de chaves
void sendKeyMessage(msg1_t* rcm)
{
   if (lockedKey)
   {
      return;
   }
   if (call KeySender.maxPayloadLength() < sizeof(msg1_t))
   {
  return;
   }
   if (call KeySender.send(AM_BROADCAST_ADDR, &packet, sizeof(msg1_t)) == SUCCESS)
   {
  lockedKey = TRUE;
   }
}

//Realiza o xor entre as chaves e depois as envia
void sendKeys()
{
   msg1_t* rcm = (msg1_t*)call KeySender.getPayload(&packet);
   rcm->id1 = 2;
   rcm->id2 = 3;
```

```
   rcm->xor = k2 ^ k3;
   sendKeyMessage(rcm);
}

//Obtém o tamanho de uma string
uint8_t getMessageSize(uint8_t msg[MSG_MAX_LEN])
{
   uint8_t i = 0;
  while(msg[i] != 0)
     i++;
  return i;
}

//Envia uma mensagem ao outro mote
void sendMessage()
{
   if (lockedRegular)
   {
      return;
   }
   else
   {
      msg2_t* rcm = (msg2_t*)call RegularSender.getPayload(&packet);

      //Mensagem a ser enviada
      uint8_t msg[MSG_MAX_LEN];

      //Cópia da mensagem usada para gerar o log
      uint8_t msgCopy[MSG_MAX_LEN];

      //Tamanho da mensagem
      uint8_t msgLength;

      //Cópia do tamanho de uma mensagem usada para gerar o log
      uint8_t msgLengthCopy;

      //Copia a mensagem para a variável msg
      getMessage(msg);

      //Copia a mensagem para a variável msgCopy
      strcpy(msgCopy, msg);

      msgLength = getMessageSize(msg);
      msgLengthCopy = msgLength;
      rcm->sender = TOS_NODE_ID;
      rcm->sec = count;

      //Se a chave foi recebida e o mote já enviou seu número mínimo de mensagem não encriptadas
      if(ENABLE_ENCRYPTION)
      {
         if(keyReceived)
         {
            uint64_t plainKey = 0;
            if(TOS_NODE_ID == 2)//Encripta com a chave do destinatário
```

```
            {
               plainKey = k3 ^ z2; // Obtém a chave em texto claro
               encryptMessage(plainKey, msg, msgLength);
               encryptLength(plainKey, &msgLength);
            }
            else
               if(TOS_NODE_ID == 3)//Encripta com a chave do destinatário
               {
                  plainKey = k2 ^ z3; // Obtém a chave em texto claro
                  encryptMessage(plainKey, msg, msgLength);
                  encryptLength(plainKey, &msgLength);
               }
             rcm->status = MSG_ENC;
          }
          else
             return;
      }
      else //Envia a mensagem sem criptografia
      {
         rcm->status = MSG_PLAIN;
      }
      rcm->length = msgLength;
      strcpy(rcm->message, msg);
      if (call RegularSender.maxPayloadLength() < sizeof(msg2_t))
      {
     return;
      }
      if (call RegularSender.send(AM_BROADCAST_ADDR, &packet, sizeof(msg2_t)) == SUCCESS)
      {
     int i = 0;
     lockedRegular = TRUE;
            count,sim_time_string());
      }
   }
}

event void AMControl.startDone(error_t err)
{
   if (err == SUCCESS)
   {
      if (TOS_NODE_ID == 1) // O nó móvel conhece as chaves de todos os sensores
      {
         R = 198656723456789865;
         k2 = 184444073709551615 ^ R;
         k3 = 178953296589432169 ^ R;
         sendKeys();
 call AMControl.stop();
      }
      else
      {
         if(TOS_NODE_ID == 2) // O sensor 2 conhece apenas sua chave e seu segredo z
         {
            k2 = 184444073709551615;
            z2 = 169874345676543321;
```

```
            k2 = z2 ^ k2; //Guarda a chave de forma encriptada
         }
         else
            if(TOS_NODE_ID == 3) // O sensor 3 conhece apenas sua chave e seu segredo z
            {
               k3 = 178953296589432169;
               z3 = 172123489098765447;
               k3 = z3 ^ k3; //Guarda a chave encriptada
            }
         call MilliTimer.startPeriodic(1);
      }
   }
   else
   {
      call AMControl.start();
   }
}

event void AMControl.stopDone(error_t err)
{
   // do nothing
}

event void MilliTimer.fired()
{
   sendMessage();
}

event message_t* RegularReceiver.receive(message_t* bufPtr, void* payload, uint8_t len)
{
   if (len != sizeof(msg2_t)) //Pacote corrompido
   {
      return bufPtr;
   }
   else
   {
      int i = 0;
      msg2_t* rcm = (msg2_t*)payload;
      uint8_t msg[MSG_MAX_LEN];
      uint8_t status = rcm->status;
      uint8_t msgLength = rcm->length;
      uint64_t sec = rcm->sec;
      strcpy(msg,rcm->message);
      if(status == MSG_ENC)//Se a mensagem está encriptada, decripta
      {
         uint64_t plainKey;
         if(TOS_NODE_ID == 2)
         {
            plainKey = k2 ^ z2; //Obtém a chave em texto claro
            encryptLength(plainKey, &msgLength);
            encryptMessage(plainKey, msg, msgLength);
         }
         else
            if(TOS_NODE_ID == 3)
```

```
            {
               plainKey = k3 ^ z3; //Obtém a chave em texto claro
               encryptLength(plainKey, &msgLength);
               encryptMessage(plainKey, msg, msgLength);
            }
         }
         dbg("msgin","%s;%hu;%hu;%hu;%"PRId64";%s\n",msg,rcm->status,msgLength,
             rcm->sender,sec,sim_time_string());
         return bufPtr;
      }
   }

   event void KeySender.sendDone(message_t* bufPtr, error_t error)
   {
      if (&packet == bufPtr)
      {
         dbg("distrib","Broadcast de distribuição concluído. De %hu no tempo @ %s\n", TOS_NODE_ID, sim_time_string());
         lockedKey = FALSE;
      }
   }


   event message_t * KeyReceiver.receive(message_t *msg, void *payload, uint8_t len)
   {
      if (len != sizeof(msg1_t))
      {
         return msg;
      }
      else
      {
         msg1_t* rcm = (msg1_t*)payload;
         uint64_t keys = rcm->xor;
 if(TOS_NODE_ID == 2)
            k3 = k2 ^ keys;
        else
            if(TOS_NODE_ID == 3)
               k2 = k3 ^ keys;
     //dbg("distrib", "Chave recebida por %hu.\n", TOS_NODE_ID);
     keyReceived = TRUE;
      }
      return msg;
   }

   event void RegularSender.sendDone(message_t *msg, error_t error)
   {
      count++;
      lockedRegular = FALSE;
   }

   event void SecondCounter.fired()
   {
      secCount = (secCount + 1) % 4294967296; // Conta até o máximo valor
   }
}
```

Arquivo secAppc.nc

```
#include "sec.h"

configuration secAppC {}
implementation {
  components MainC, secM as App;
  components new AMSenderC(AM_MSG1) as KeySender;
  components new AMSenderC(AM_MSG2) as RegularSender;
  components new AMReceiverC(AM_MSG2) as RegularReceiver;
  components new AMReceiverC(AM_MSG1) as KeyReceiver;
  components new TimerMilliC();
  components new TimerMilliC() as secondCounter;
  components ActiveMessageC;

  App.Boot -> MainC.Boot;
  App.RegularReceiver -> RegularReceiver;
  App.KeyReceiver -> KeyReceiver;
  App.KeySender -> KeySender;
  App.RegularSender -> RegularSender;
  App.AMControl -> ActiveMessageC;
  App.MilliTimer -> TimerMilliC;
  App.SecondCounter -> secondCounter;
}
```

Arquivo sec.h

```
#ifndef SEC_H
#define SEC_H

typedef nx_struct msg1
{
  nx_uint8_t id1;
  nx_uint8_t id2;
  nx_uint64_t xor;
} msg1_t;

enum
{
  AM_MSG1 = 1,
  AM_MSG2 = 2,
  MSG_ENC = 1,
  MSG_PLAIN = 2,
  MSG_MAX_LEN = 16
};

typedef nx_struct msg2
{
  nx_uint16_t sender;
  nx_uint8_t message[MSG_MAX_LEN];
  nx_uint8_t length;
  nx_uint8_t status;
  nx_uint64_t sec;
```

```
} msg2_t;

#endif
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Nadeem Ahmed, Salil S. Kanhere e Sanjay Jha. The holes problem in wirelesssensor networks: a survey. *SIGMOBILE Mob. Comput. Commun. Rev.*, pp. 4–18, 2005.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam e E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] A. Becher, Z. Benenson e M. Dornseif. Tampering with motes: Real-tampering physical attacks on wireless sensor networks. *3rd International Conference on Security in Pervasive Computing (SPC).*, pp. 1–15, abril de 2006.

[4] Chiara Buratti, Andrea Conti, Davide Dardari e Roberto Verdone. An overview on wireless sensor networks technology and evolution. *Sensors*, pp. 6869–6896, agosto de 2009.

[5] S. Deb, M. Effros, T. Ho, D. Karger, R. Koetter, D. Lun, M. Medard e N. Ratnakar. Network coding for wireless applications: A brief tutorial. *IWWAN.*, maio de 2005.

[6] L. Eschenauer e V. D. Gligor. A key-management cheme for distributed sensor networks. *Proc. 9th ACMConf. Computer and Communications Security.*, pp. 41–47, 2002.

[7] D. Gay, P. Levis, R. von Behren, M.Welsh, E. Brewer e D. Culler. The nesc language: A holistic approach to networked embedded systems. Em *Programming Language Design and Implementation (PLDI)*, 2003.

[8] Distributed Computing Group. Yeti 2 - tinyos 2 plugin for eclipse. [Online; Acessado em 14-Novembro-2010].

[9] Fei Hu e Neeraj K. Sharma. Security considerations in wireless sensor networks. *Ad Hoc Networks*, pp. 69–89, janeiro de 2005.

[10] Chris Karlof e David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *First IEEE International Workshop on Sensor Network Protocols and Applications*, pp. 113–127, 2002.

[11] Juliano F. Kazienko e Célio V. N. Albuquerque. Secure secret key distribution and storage in wireless sensor networks. *Third IEEE International Symposium on Trust, Security and Privacy for Emerging Applications (TSP'10) in conjunction with 10th IEEE CIT*, pp. 890–895, 2010.

[12] P. Levis e D. Gay. *TinyOS Programming.* Cambridge University Press, 2009.

[13] P. Levis, N. Lee, M. Welsh e D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. Em *1st International Conference on Embedded Networked Sensor Systems.*, pp. 126–137, 2003.

[14] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer e D. Culler. Tinyos - site oficial. [Online; Acessado em 14-Novembro-2010].

[15] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer e D. Culler. Tinyos: An operating system for sensor networks. *Springer-Verlag.*, 2004.

[16] D. Malan, M. Welsh e M. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. *1st IEEE Int. Conf. Sensor and Ad Hoc Communications and Networks.*, 2004. [Online; acessado em 13-Novembro-2010].

[17] Cíntia Margi, Marcos Simplício, Paulo Barreto e Tereza Carvalho. Segurança em redes de sensores sem fio. *Livro de Minicursos do IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'09)*, pp. 149–194, 2009.

[18] Daniel A. Merege e Cíntia B. Margi. Análise de viabilidade da biblioteca de criptografia tinyecc em redes de sensores sem fio. *Simpósio Brasileiro de Segurança.*, pp. 31–60, 2010.

[19] Paulo F. Oliveira e João Barros. A network coding approach to secret key distribution. *IEE Transactions on Information Forensics and Security*, 3(3):414–423, setembro de 2008.

[20] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen e D. E. Culler. Spins: Security protocols for sensor networks. *Wireless Netw.*, 8(5):521–534, 2002.

[21] William Stallings. *Criptografia e Segurança de Redes Princípios e Práticas 4ª Edição.* Pearson, 2008.

[22] Y. Zhou, Y. Fang e Y. Zhang. Securing wireless sensor networks: A survey. *IEEE Communications Surveys and Tutorials.*, 10(3):6–28, 2008.