RICARDO JOSÉ MENEZES MAIA

ANÁLISE DA VIABILIDADE DA IMPLEMENTAÇÃO DE ALGORITMOS PÓS-QUÂNTICOS BASEADOS EM QUASE-GRUPOS MULTIVARIADOS QUADRÁTICOS EM PLATAFORMAS DE PROCESSAMENTO LIMITADAS

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para obtenção
do Título de Mestre em Engenharia Elétrica.

São Paulo
2010

RICARDO JOSÉ MENEZES MAIA

ANÁLISE DA VIABILIDADE DA IMPLEMENTAÇÃO DE ALGORITMOS PÓS-QUÂNTICOS BASEADOS EM QUASE-GRUPOS MULTIVARIADOS QUADRÁTICOS EM PLATAFORMAS DE PROCESSAMENTO LIMITADAS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de Concentração:

Sistemas Digitais

Orientador:

Paulo Sérgio Licciardi Messeder Barreto

São Paulo
2010

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 13 de outubro de 2010.

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Maia, Ricardo José Menezes

Análise da viabilidade da implementação de algoritmos pós-quânticos baseados em quase-grupos multivariados quadráticos em plataformas de processamento limitadas / R.J.M. Maia. – São Paulo, 2010.

78 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Criptossistemas de Chaves Públicas 2. Redes de Sensores sem Fio 3. Quase-Grupos Multivariados Quadráticos I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Ao meu orientador pelo constante apoio e oportunidade dada de trabalhar com tema motivador.

Aos meus pais, minha amada e futura esposa, irmãos, familiares, amigos e a todos que colaboraram direta ou indiretamente, na execução deste trabalho.

À Suframa, pelo patrocínio financeiro do Minter.

À Capes, pela criação e regulamentação do Minter

À Universidade do Estado do Amazonas, por gerar as condições e propiciar o oferecimento de programa de Pós da EPUSP em Manaus.

RESUMO

Redes de sensores sem fio (RSSF) tipicamente consistem de nós sensores com limitação de energia, processamento, comunicação e memória. A segurança em RSSF está se tornando fundamental com o surgimento de aplicações que necessitam de mecanismos que permitam autenticidade, integridade e confidencialidade. Devido a limitações de recursos em RSSF, adequar criptossistemas de chaves públicas (PKC) para estas redes é um problema de pesquisa em aberto.

Meados de 2008, Danilo Gligoroski et al. propuseram um novo PKC baseado em quase-grupos multivariados quadráticos (MQQ). Experimentos feitos por Gligoroski na plataforma FPGA mostram que MQQ executou em tempo menor que principais PKC (DH, RSA e ECC) existentes, tanto que alguns artigos afirmam que MQQ possui velocidade de uma típica cifra de bloco simétrica. Além disto, o MQQ exibiu o mesmo nível de segurança que outros PKC (DH, RSA e ECC) necessitando chaves menores. Outra propriedade que chama atenção no MQQ é o uso das operações básicas XOR, AND e deslocamento de bits nos processos de encriptação e decriptação, fato importante considerando que uma RSSF possui processamento limitado. Estas características tornam o MQQ promissor a levar um novo caminho na difícil tarefa de dotar redes de sensores sem fio de criptossistemas de chaves públicas.

Neste contexto se insere este trabalho que analisa a viabilidade de implementar o algoritmo MQQ em uma plataforma de RSSF. Sendo importante considerar que este trabalho inova na proposta de levar para RSSF este novo PKC baseado quase-grupos multivariados quadráticos, além de contribuir com um método para reduzir o tamanho da chave pública utilizada pelo MQQ.

Foram feitos testes com MQQ nas plataformas TelosB e MICAz, sendo que o MQQ exibiu os tempos de 825, 1 ms para encriptar e 116, 6 ms para decriptar no TelosB e 445 ms para encriptar no MICAz.

Palavras-Chave: Criptossistemas de Chaves Públicas. Redes de Sensores sem Fio. Quase-Grupos Multivariados Quadráticos

ABSTRACT

Wireless sensor networks (WSN) typically consist of sensor nodes with limited energy, processing, communication and memory. Security in WSN is becoming critical with the emergence of applications that require mechanisms for authenticity, integrity and confidentiality. Due to resource constraints in sensor networks, public key cryptosystems suit (PKC) for these networks is an open research problem.

In 2008 Danilo Gligoroski et al. proposed a new PKC based on quasi-groups multivariate quadratic (MQQ). Experiments by Gligoroski on FPGA platform show that MQQ performed in less time than most popular PKC (DH, RSA and ECC), so that some papers say MQQ has a typical speed of symmetric block cipher. Moreover, the MQQ exhibited same level of security that other PKC (DH, RSA and ECC) requiring keys minors. Another property that draws attention in MQQ is the use of basic operations XOR, AND, and bit shifting in the processes of encryption and decryption, important fact considering that a WSN has limited processing. These features make the MQQ promising to take a new path in the difficult task of providing wireless sensor networks in public key cryptosystems.

Appears in this context that this study examines the feasibility of implementing MQQ a platform for WSN. Is important to consider this innovative work in the proposal to bring this new PKC for WSN based multivariate quadratic quasigroups, and contribute a method to reduce the size public key used by MQQ.

Tests with MQQ on platforms TelosB and MICAz, the MQQ exhibited 825ms to encrypt and 116ms to decrypt on TelosB and 445 ms to encrypt on MICAz.

Keywords: Public Key Cryptosystems. Wireless Sensor Networks. Multivariate Quadratic Quasigroups.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DH	Diffie e Hellman
ECC	Elliptic Curve Cryptography
MEMS	Micro Eletro-Mecanical System
MQQ	Multivariate Quadratic Quasigroup
PKC	Public Key Cryptosystem
RSSF	Redes de Sensores Sem Fio

1 INTRODUÇÃO

Avanços nas técnicas de miniaturização e comunicação sem fio proporcionam o desenvolvimento de um novo paradigma, onde se destacam redes de sensores sem fio (RSSF). As RSSF são tipicamente compostas de pequenos dispositivos dotados de unidade de processamento, sensoriamento e comunicação, denominados nós sensores. Uma rede de sensores sem fio não possui uma infra-estrutura fixa (LOUREIRO et al., 2003; LOPEZ; ZHOU, 2008; WANG; ZHANG, 2009; CAYIRCI; RONG, 2009).

Os sensores extraem e transmitem dados ambientais para um ou mais pontos de saída da rede, chamados nós sorvedouros. Posteriormente, os dados enviados pelos sensores serão armazenados e em seguida processados. A instalação de tais sensores pode ser em locais pré-definidos ou não, na área alvo. Estes possuem recursos extremamente limitados de suprimento de energia, poder de processamento, memória para armazenamento e sistemas de comunicação com baixa largura de banda (LOUREIRO et al., 2003; MARGI et al., 2009; LIU; NING, 2007; PALAFOX; GARCIA-MACIAS, 2008).

RSSF podem ser utilizadas em diversas aplicações, tais como controle do trânsito, monitoramento de variáveis ambientais, detecção de material perigoso, detecção de movimentos inimigos (aplicações militares), identificação e cadastramento de pessoas em grandes ambientes (aeroportos), monitoramento da saúde humana, monitoração dos níveis de umidade em áreas agrícolas para realizar irrigação seletiva, detecção de invasores em zonas de fronteira, entre outros (LOUREIRO et al., 2003; LOPEZ; ZHOU, 2008; CAYIRCI; RONG, 2009; SABBAH ERIC; KANG, 2009; PALAFOX; GARCIA-MACIAS,

2008).

RSSF utilizam comunicação sem fio, sendo mais vulneráveis a ataques, uma vez que neste tipo de comunicação, o modo de transmissão utilizado é *broadcast*. Ao utilizar *broadcast*, a rede fica mais susceptível a ação de intrusos, que podem facilmente escutar, interceptar e alterar os dados que trafegam na rede (LOUREIRO et al., 2003; MARGI et al., 2009; LOPEZ; ZHOU, 2008; LIU; NING, 2007; SABBAH ERIC; KANG, 2009; CAYIRCI; RONG, 2009).

Em RSSF há aplicações críticas que necessitam de propriedades de segurança, tais como integridade, confidencialidade e autenticidade. Exemplos de sistemas críticos com RSSF estão em aplicações militares e monitoramento remoto de sinais vitais de pacientes (OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008).

Considerando as limitações de recursos em uma RSSF, verifica-se um profundo impacto na adoção dos protocolos e algoritmos de comunicação e de segurança. Portanto, uma questão básica é satisfazer os requisitos das aplicações de forma segura, considerando as restrições existentes nestas redes (MARGI et al., 2009; GUIMARÃES et al., 2005; SZCZECHOWIAK et al., 2008).

A solução atual para o problema de estabelecer premissas de segurança em RSSF é em torno de criptossistemas simétricos, mesmo considerando a maior segurança proporcionada por criptossistemas de chave pública (*public key cryptosystem* - PKC) (OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008).

Apesar de PKC tornar um sistema mais seguro que criptossistemas simétricos, a adequação PKC no cenário de recursos limitados de RSSF é um problema de pesquisa aberto (OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008).

Recentemente foi proposto um novo esquema de chaves públicas chamados de Multivariado Quadrático Quase Grupo (MQQ) (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008). Experimentos feitos com o MQQ e outros criptossistemas de chaves

públicas mais populares, tais como RSA e ECC, mostram que o MQQ foi mais rápido tanto para cifrar quanto decifrar nas plataformas FPGA e PC(GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

Neste trabalho é feita a a implementação de algoritmo MQQ em RSSF. A motivação para utilizar MQQ em RSSF deve-se ao fato de MQQ mostrar-se resultados promissores com relação a outros PKC.

Nos próximos capítulos são feitas descrições breves sobre RSSF e seus aspectos de segurança relevantes para este trabalho, além de detalhamento das propriedades do MQQ que o tornam atraentes para RSSF. A estrutura dos capítulos encontra-se na seção 1.6. Uma breve explicação dos principais conceitos que não foram detalhados no texto encontram-se no glossário.

1.1 Caracterização do Problema

Os avanços ocorridos nas áreas de microprocessadores (através de miniaturização), novos materiais de sensoriamento, micro-sistemas eletromecânicos (MEMS - *Micro Eletro-Mecanical System*) e a evolução da comunicação sem fio criaram um novo paradigma na monitoração de ambientes (LOPEZ; ZHOU, 2008).

Neste cenário destacam-se as RSSF, como sendo um tipo de sistema distribuído reativo que pode estar ou não associado à aplicações críticas. Os equipamentos envolvidos em uma RSSF possuem limitações de energia, poder de processamento, capacidade de armazenamento e largura de banda dos sistemas de comunicação (LOPEZ; ZHOU, 2008; MARGI et al., 2009; LOUREIRO et al., 2003).

As RSSF abrem espaço para uma ampla variedade de aplicações, as quais irão exigir requisitos de segurança como confidencialidade, integridade e autenticidade. Em aplicações de automação residencial a autenticidade é primordial, para possibilitar que os sensores sejam monitorados apenas pelos proprietários. Nas aplicações industriais a

exigência dos três requisitos pode ser fundamental para evitar espionagem ou que outras empresas possam obter vantagem competitiva (GAUBATZ et al., 2005; GUIMARÃES et al., 2005; MARGI et al., 2009; WANG; ZHANG, 2009; LIU; NING, 2007; CAYIRCI; RONG, 2009).

Muitos esforços foram realizados para aumentar a segurança em RSSF, sendo que a maioria dos sistemas para RSSF utiliza criptografia simétrica para aumentar a segurança dos dados. A opção por criptossistemas simétricos deve-se à eficiência destes esquemas em RSSF, ao contrário de PKC (SZCZECHOWIAK et al., 2008).

Devemos considerar que esquemas simétricos possuem inconvenientes à segurança, devido ao uso de uma chave tanto para cifrar quanto para decifrar, o que pode comprometer todo o sistema, caso a chave seja exposta. Neste caso é bom salientar que nas aplicações em RSSF os sensores geralmente ficam expostos no ambiente, sendo possível um intruso violar algum sensor para obter a chave privada igualmente utilizada por todos os outros sensores (MARGI et al., 2009; LOUREIRO et al., 2003; OLIVEIRA et al., 2008).

Apesar de PKC possibilitarem maior segurança que esquemas simétricos, as restrições impostas por RSSF torna a implantação de PKC em RSSF um problema em aberto (SZCZECHOWIAK et al., 2008; OLIVEIRA et al., 2008).

Recentemente surgiu um novo esquema de chaves públicas, chamado Multivariado Quadrático Quase Grupo (MQQ), baseado em polinômios multivariados quadráticos e transformações de *strings* quase grupos (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008). Experimentos realizados mostram o MQQ várias ordens de grandeza mais rápido que os mais populares algoritmos de chaves públicas como RSA, DH e ECC (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008; AHLAWAT; PAL, 2009).

Experimentos mostram que MQQ exibe mesmo nível de segurança que o RSA,

onde o MQQ com chaves de 160 bits obtem mesmo nível de segurança do RSA com chaves de 1024 bits (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008). O MQQ mostrou-se veloz tanto para encriptar quanto para decriptar, possuindo velocidade de uma típica cifra de bloco simétrica (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

Considerando a necessidade crescente de maior segurança para RSSF, os recursos limitados deste tipo de rede e por último as características promissoras do MQQ para plataformas com recursos limitados pode-se considerar o MQQ um novo caminho para dotar RSSF de PKC. Apesar dos resultados promissores do MQQ em outras plataformas, não há trabalhos anteriores relacionando MQQ a RSSF. Portanto, o problema a ser abordado neste trabalho consiste em analisar se o MQQ é viável em uma plataforma de RSSF.

1.2 Justificativa e Motivação

Nas aplicações críticas em RSSF, é vital o uso de mecanismos de segurança proporcionados por PKC, tais como: autenticidade, integridade e confidencialidade. Tornar viável a utilização de mecanismos de segurança em RSSF requer a busca por algoritmos criptográficos que consumam o mínimo de recursos existentes, tais como energia, memória e processamento (GAUBATZ et al., 2005; GUIMARÃES et al., 2005; OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008; MARGI et al., 2009).

Os resultados obtidos em (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008) e validados em (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008) são promissores, afinal estes dois documentos mostram que MQQ nas plataformas FPGA e PC é mais rápido que tradicionais criptossistemas de chaves públicas, tais como RSA e ECC.

Este desempenho do MQQ com relação a tradicionais PKC (RSA e ECC) chama atenção, afinal até o momento a performance de ECC era superior a outros PKC tradici-

onais, como o RSA (OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008; PAL; SUMITRA, 2009). MQQ consegue o mesmo nível de segurança que tradicionais PKCs, consumindo muito menos recursos computacionais (PAL; SUMITRA, 2009; GAUBATZ et al., 2005; BOGDANOV THOMAS EISENBARTH; WOLF, 2008).

Portanto, o MQQ torna-se uma promessa não somente para RSSF, mas para plataformas de processamento limitadas como um todo. Dado que o MQQ oferece a segurança de tradicionais PKC consumindo muito menos recursos computacionais, tais como memória, processamento e energia. (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

1.3 Objetivos

O objetivo deste trabalho é implementar os módulos de encriptação e decriptação do criptossistema de chaves públicas baseado em Quase-Grupos Multivariados Quadráticos (MQQ) em uma plataforma de rede de sensores sem fio (RSSF). Desta forma, busca-se encontrar uma nova proposta para dotar redes de sensores sem fio de criptossistemas de chaves públicas oferecendo maior segurança a dispositivos com recursos limitados de processamento, energia e memória.

1.4 Metodologia

Tendo em vista o cenário que é analisar a viabilidade de implementar o MQQ em RSSF, o método a ser utilizado neste trabalho consiste primeiramente em pesquisar sobre conceitos matemáticos envolvidos no algoritmo MQQ.

Faz parte deste trabalho a especificação, implementação e análise do desempenho dos algoritmos de encriptação e decriptação do MQQ em uma plataforma com recursos limitados, neste caso RSSF. Visando facilitar a análise comparativa do desempenho, é feita análise comparativa entre o MQQ e o algoritmo RSA utilizando como parâmetros

tempo de execução e mesmo nível de segurança. O nível de segurança é obtido com base no tamanho da chave criptográfica em bits, de acordo com trabalhos recentes, o nível de segurança dado pelo RSA com 1024 bits equivale ao do MQQ com chaves de 160 bits.

O tempo de execução foi medido e analisado para verificar a viabilidade do MQQ em uma plataforma de RSSF.

O espaço ocupado em memória RAM e ROM também é analisado, afinal um nó sensor possui este recurso limitado. O parâmetro utilizado para verificar se o MQQ é viável em uma plataforma de RSSF é o limite de memória do nó sensor.

As operações envolvidas nos processos de encriptação e decriptação não foram otimizadas neste trabalho, pois o MQQ necessita de instruções básicas com bits, exemplo XOR, AND e deslocamentos. Além disso não faz parte deste trabalho implementar o algoritmo de geração de chaves do MQQ. A análise do consumo de energia também não fazer parte deste trabalho.

O método de programação utilizado é similar para os algoritmos, procurando não favorecer um algoritmo em detrimento de outro.

1.5 Contribuições

A principal contribuição deste trabalho consiste em propor uma nova abordagem para dotar redes de sensores sem fio com criptossistemas de chaves públicas, por meio do algoritmo MQQ. Além disso, foi proposto um método de acomodar as estruturas de dados necessárias para o processo de encriptação do MQQ, uma vez que nos trabalhos originais as estruturas de dados para encriptar necessitam de um espaço em memória que excede a memória disponível nas plataformas TelosB e MICAz. Nesta abordagem considera-se que termos dos polinômios envolvidos no MQQ podem estar repetidos, esta abordagem é melhor definida no capítulo 4.

1.6 Estrutura do Texto

Apresenta-se a seguir um sumário estruturado da dissertação.

No capítulo 2 há a descrição do que são redes de sensores sem fio e segurança nestas redes. Considerando o amplo campo da segurança em RSSF este capítulo focaliza nos problemas da utilização de esquema de chaves públicas em RSSF, além de descrever cenários onde a segurança é primordial em RSSF.

No capítulo 3 são descritas as propriedades do algoritmo MQQ, mostra-se porque o MQQ pode ser uma proposta mais interessante que outros criptossistemas de chaves públicas. O capítulo também inclui a descrição detalhada do algoritmo MQQ.

No capítulo 4 descreve-se como foi realizado o experimento com MQQ e RSA no TinyOS. Discute-se sobre os parâmetros utilizados para comparar os dois algoritmos: espaço em memória e tempo de execução.

No capítulo 5 são feitas considerações finais sobre o trabalho e discute-se trabalhos futuros.

2 REDES DE SENSORES SEM FIO

Neste capítulo são abordados os conceitos de redes sensores sem fio e seus aspectos de segurança relevantes para o presente trabalho. Este trabalho foca na necessidade das premissas confidencialidade, integridade e autenticidade e o custo computacional para implementar estas premissas em RSSF.

Uma RSSF é uma rede *ad hoc*, principalmente compreendendo nós sensores que são normalmente utilizados para acompanhar e observar um fenômeno ou uma cena (LOUREIRO et al., 2003).

Os nós sensores são fisicamente implantados dentro ou perto do fenômeno ou cena a ser monitorada. Posteriormente os dados coletados serão enviados de volta para uma estação de base, também chamada nó sorvedouro (MARGI et al., 2009).

A figura 1 representa uma estação monitorando os dados obtidos de uma RSSF através da Internet. No *gateway* há um PC comunicando-se com um sorvedouro através da comunicação serial RS232. O PC por sua vez torna-se o servidor dos dados obtidos da RSSF.

RSSF são compostas tipicamente por equipamentos que possuem recursos limitados, tais como baixo poder de processamento, pouca memória para armazenamento, baixa largura de banda e fonte de alimentação com capacidade limitada. Um nó sensor pode realizar operações simples e só pode se comunicar com sensores e outros nós dentro de um curto alcance. Os nós sensores geralmente utilizam baterias e tem o tempo de vida limitado por este fornecimento de energia (LOUREIRO et al., 2003; LOPEZ; ZHOU,

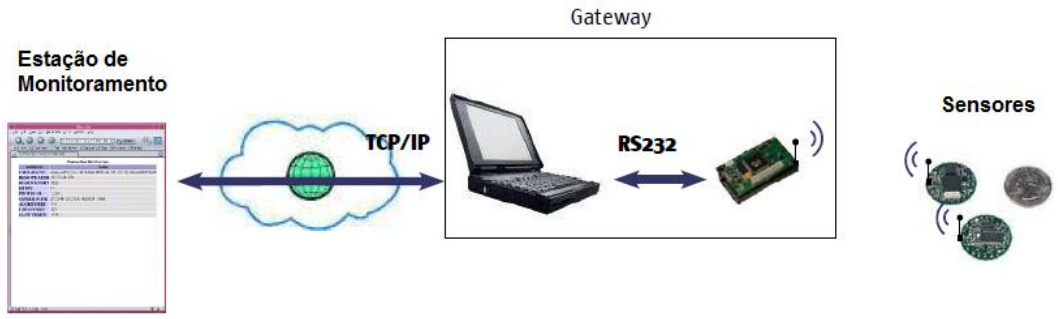

Figura 1: Exemplo de estação de monitoramento acessando dados de uma RSSF pela internet

2008; WANG; ZHANG, 2009).

As RSSF podem ser auto-organizadas depois que sensores sejam implantados no local a ser monitorado, onde os nós sensores cooperam com outros sensores para realizar a tarefa de controlar ou observar a cena de destino ou o fenômeno alvo, e comunicar-se com um nó sorvedouro que coleta os dados de todos os nós sensores. A cooperação pode envolver encontrar uma rota para transmitir dados para um destino específico, transmitindo dados a partir de um vizinho para outro vizinho, quando os dois vizinhos não estão ao alcance de uns aos outros, e assim por diante. Na especificação de protocolos de comunicação para RSSF é primordial levar em consideração a otimização do consumo de energia pelos nós (LOUREIRO et al., 2003; MARGI et al., 2009; LOPEZ; ZHOU, 2008; LIU; NING, 2007; PALAFOX; GARCIA-MACIAS, 2008).

A compreensão sobre as fontes de consumo de energia, sobre processamento, sensoreamento e comunicação proporciona o conhecimento necessário para economizar energia em RSSF. Esta especificação permite que um sensor alterne entre os estados de atividade, inatividade (*idle*) e modo de baixo consumo (*sleep*), economizando energia (MARGI et al., 2009; LIU; NING, 2007; SABBAH ERIC; KANG, 2009; CAYIRCI; RONG, 2009).

2.1 Aplicações em RSSF

O uso de RSSF permite desenvolver aplicações em diversas áreas como controle do trânsito, monitoramento ambiental, detecção de material perigoso, detecção de movimentos inimigos (aplicações militares), controle de temperatura e umidade de ambientes fechados, detecção de intrusos em fronteiras entre outros. Um possível cenário seria os dados coletados pelos nós sensores são enviados para um nó sorvedouro, que iria encaminhá-los para um servidor conectado na internet (LOUREIRO et al., 2003; MARGI et al., 2009; LOPEZ; ZHOU, 2008; WANG; ZHANG, 2009; LIU; NING, 2007; PALAFOX; GARCIA-MACIAS, 2008; SABBAH ERIC; KANG, 2009; CAYIRCI; RONG, 2009).

Alguns exemplos de aplicações em RSSF são descritos a seguir:

- Aplicações médicas podem monitorar os sinais vitais, tais como ritmos cardíacos e os níveis de oxigênio no sangue, e para comunicar essas leituras para os leitores através de uma rede sem fio no hospital, em uma ambulância e na residência. Nestes sistemas de monitoramento de pacientes a confidencialidade evita que pessoas não autorizadas tenham acesso a dados do paciente, a integridade irá garantir que dados sobre o paciente não tenham sido alterados evitando diagnóstico errado e autenticidade possibilita verificar a identidade dos médicos.

- Detecção de incêndio em florestas, onde RSSF podem ser densa e aleatoriamente dispostas sobre florestas, podendo emitir alerta da exata localização do foco de incêndio, antes mesmo do fogo tornar-se incontrolável. Na detecção de enchentes as RSSF podem ser usadas para detecção de enchentes em locais de difícil acesso (LOUREIRO et al., 2003). Nestes dois casos a integridade dos dados pode evitar falsos alertas, além disso a autenticidade pode evitar que pessoas não autorizadas tenham acesso ao sistema e possam enviar falsos alertas (MARGI et al., 2009).

- Agricultura de precisão, onde é possível monitorar a concentração de pesticidas na água, o grau de erosão do solo e o nível de poluição do ar, tudo em tempo real (LOUREIRO et al., 2003). Nesta aplicação a integridade pode evitar que os dados monitorados sejam imprecisos (MARGI et al., 2009).

- Vigilância em campo de batalha, onde terrenos estrategicamente críticos e rotas importantes poderiam ser rapidamente cobertas por RSSF. Em caso de movimentação de tropas inimigas sistemas de alarme seriam acionados e possíveis contra medidas tomadas, até mesmo automaticamente (LOUREIRO et al., 2003). Nas aplicações militares premissa de segurança como autenticidade, integridade e confidencialidade são fundamentais.

2.2 Segurança em RSSF

As RSSF, por utilizarem comunicação sem fio, são vulneráveis a ataques, uma vez que neste tipo de comunicação o modo de transmissão naturalmente utilizado é *broadcast*. O uso de *broadcast* torna RSSF passíveis da ação de intrusos, que podem facilmente "escutar", interceptar e alterar os dados que trafegam na rede (MARGI et al., 2009; LOPEZ; ZHOU, 2008; SABBAH ERIC; KANG, 2009; GUIMARÃES et al., 2005).

De acordo com a aplicação a ser implementada são escolhidos os requisitos de segurança, onde os principais requisitos de segurança são (GUIMARÃES et al., 2005):

- **Autenticidade** assegura que o receptor consegue verificar a identidade do emissor, evitando assim a injeção de dados maliciosos na rede. Evitando que sensores de RSSF vizinhas interajam entre si acidentalmente.

- **Confidencialidade** assegura que o conteúdo da mensagem somente seja acessado por nós autorizados. Garantindo a proteção contra a revelação dos dados para pessoas não autorizadas.

- **Integridade** permite detectar se o conteúdo da mensagem foi alterado no decorrer da transmissão, onde o receptor pode identificar estas alterações.

- **Irretratabilidade** garante que nem o remetente nem o destinatário de determinada mensagem possam negar sua transmissão, recepção ou posse.

Problemas de segurança podem ser apresentados nos mecanismos de acesso, em falhas dos protocolos de comunicação, em falhas no software ou no hardware dos mecanismos de transporte de dados ou na falsificação da identidade de nós sensores (GUIMARÃES et al., 2005).

As limitações existentes em RSSF, tais como limitações de processamento, limitações na conservação da energia das baterias e memória para armazenamento, ocasionam um profundo impacto na adoção dos protocolos e algoritmos de comunicação e de segurança (MARGI et al., 2009; GUIMARÃES et al., 2005).

As limitações de recursos de poder de processamento, memória para armazenamento e energia disponível nas RSSF devem ser consideradas durante a escolha de criptossistemas. Considerando estes recursos limitados, a solução atual para o problema de estabelecer segurança em RSSF é através de sistemas de criptografia simétrica, os quais exigem menor recurso computacional que os criptossistemas assimétrico, mas possuem os seguintes inconvenientes (MARGI et al., 2009; OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008):

- Única chave para cifrar e decifrar compromete a rede toda caso a chave secreta do nó seja comprometida. Caso um instruso conheça a chave existente em algum sensor, este intruso conseguirá estabelecer comunicação com todos os outros sensores da rede (MENEZES; OORSCHOT; A.VANSTONE, 1999; MARGI et al., 2009; TERADA, 2008).

- Inviabilidade de soluções que usam gerenciamento de chave centralizado em

RSSF com muitos nós, onde é verificada a identidade do nó e com quais nós pode-se estabelecer conexões seguras.

Criptossistemas de chaves públicas são freqüentemente utilizados em comércio eletrônico, uma vez que fornece a integridade de mensagem e autenticação da fonte de informação, além de confidencialidade. Um dos sistemas de encriptação popular é o algoritmo RSA, o qual se baseia na dificuldade de fatoração de grandes números em seus fatores primos.

Apesar de criptossistemas simétricos exigirem menos recursos que um PKC, percebe-se que aplicações críticas em RSSF exigem os requisitos de segurança somente obtidos com PKC (OLIVEIRA et al., 2008).

A inviabilidade no uso de PKC em RSSF motivou pesquisas para obter criptossistemas de chaves públicas que fossem adequados nestas redes (GUIMARÃES et al., 2005; MARGI et al., 2009; SZCZECHOWIAK et al., 2008; OLIVEIRA et al., 2008; GURA et al., 2004).

Atualmente a técnica baseada em criptografia de curvas elípticas (*Elliptic Curve Cryptography* - ECC) é a base para pesquisas que tenham objetivo de dotar RSSF de um PKC. Afinal com ECC obtém-se o mesmo nível de segurança que o RSA, mas consumindo poucos recursos computacionais (OLIVEIRA et al., 2008; SZCZECHOWIAK et al., 2008; SABBAH ERIC; KANG, 2009; LOPEZ; ZHOU, 2008; CAYIRCI; RONG, 2009; LOUREIRO et al., 2003).

3 QUASE-GRUPO MULTIVARIADO QUADRÁTICO

A necessidade de um melhor sistema de codificação levou ao desenvolvimento de algoritmos simétricos e assimétricos. Criptossistemas simétricos tem a desvantagem do comprometimento da comunicação, caso a chave de um dos participantes seja comprometida. Criptografia de chave pública ou criptografia de chave assimétrica é a base para a maioria dos protocolos de segurança utilizados na Internet. Criptossistemas de chaves públicas possibilitam a integridade de mensagem e autenticação da fonte de informação, além da confidencialidade dos dados (STALLINGS, 2008).

Os mais populares criptossistemas de chaves públicas são:

- O esquema de Diffie e Hellman (DH) é um esquema de trocas de chaves baseado na dificuldade do problema de logaritmo discreto (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008) (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

- O RSA(Rivest, Shamir e Adleman) é um esquema baseado na dificuldade computacional de fatorar um número inteiros em primos (RIVEST; SHAMIR; ADLEMAN, 1978).

- Criptografia de Curvas Elípticas (*Elliptic Curve Cryptography*-ECC) de Koblitz e Miller é um esquema baseado na dificuldade de se resolver o problema do logaritmo discreto sobre curvas elípticas (KOBLITZ, 1987) (Miller, 1986).

Há duas características em comum destes conhecidos criptossistemas de chaves

públicas (DH, RSA e ECC):

1. Sua velocidade (que freqüentemente é mil vezes menor do que os criptossistemas simétricos (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

2. Sua segurança se baseia em dois problemas matemáticos intratáveis computacionalmente: eficiência computacional do cálculo de logaritmos discretos e fatoração de inteiros (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

Em 2008, foi proposto um novo esquema de chaves públicas chamado multivariado quadrático quase-grupo (MQQ) (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008). Experimentos feitos em hardware mostraram que MQQ pode ser tão rápido quanto uma típica cifra de bloco simétrica, sendo várias ordens de grandeza mais rápido que os mais populares criptossistemas de chaves públicas conhecidos como RSA, DH e ECC (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

O MQQ baseia-se em polinômios multivariados quadráticos e transformações de quase grupo e possui as seguintes propriedades (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008; AHLAWAT; PAL, 2009):

- Altamente paralelizável ao contrário de outros algoritmos que são essencialmente sequenciais (RSA, DH e ECC).

- Possui mapeamento determinístico de um para um, permitindo que não haja muita expansão da mensagem.

- Na encriptação a velocidade é comparável a outros criptossistemas de chaves públicas baseados em multivariados quadráticos.

- Na decriptação a velocidade é de uma típica cifra de bloco simétrica.

- Algoritmo pós-quântico. Quanto a esta propriedade deve-se considerar que não está longe o momento quando computadores quânticos serão concebidos para resolver problemas práticos e serão utilizados para quebrar muitos algoritmos criptográficos existentes (NIELSEN; CHUANG, 2003). Quando este momento chegar, será necessário o projeto de sistemas criptográficos baseados em problemas intratáveis num computador quântico.

MQQ dá uma nova direção para o campo da criptografia, podendo ser utilizado para desenvolver novos criptossistemas de chave pública, bem como melhorar esquemas criptográficos existentes (AHLAWAT; PAL, 2009).

3.1 Preliminares Matemáticas e Notação

A seguir são descritos alguns conceitos necessários para o entendimento do algoritmo MQQ (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008; GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008).

Definição 1. Um quase-grupo é um grupóide que satisfaz a seguinte lei

$$(\forall u, v \in Q)(\exists! x, y \in Q) u * x = v \& y * u = v. \tag{3.1}$$

Tomando como base a equação 3.1 pode-se concluir que para cada $a, b \in Q$ há um único $x \in Q$ tal que $a * x = b$. Então $x = a\backslash_* b$ onde $\backslash_*$ é uma operação binária em Q e o grupóide $(Q, \backslash_*)$ também é um quase-grupo. A álgebra $(Q, *, \backslash_*)$ satisfaz a equação 3.2.

$$x\backslash_*(x * y) = y, x * (x\backslash_* y) = y \tag{3.2}$$

Considere um alfabeto, ou seja, um conjunto finito Q, e Q^+ o conjunto de todas as palavras não vazias formados pelos elementos de Q. Neste trabalho tanto as notações $Q^+ : a_1 a_2 ... a_n$ e $(a_1, a_2, ..., a_n)$ poderão ser utilizadas, onde $a_i \in Q$. Considere $*$ a operação sobre o quase-grupo no conjunto Q. Para cada $l \in Q$ foram definidas duas funções

$e_{l,*}, d_{l,*} : Q^+ \to Q^+$.

Definição 2. Considere $ai \in Q, M = a_1a_2...a_n$. Então

$e_{l,*}(M) = b_1b_2...b_n \iff$

$b_1 = l * a_1, b_2 = b_1 * a_2, ..., b_n = b_{n-1} * a_n,$

$d_{l,*}(M) = c_1c_2...c_n \iff$

$c_1 = l * a_1, c_2 = a_1 * a_2, ..., c_n = a_{n-1} * a_n,$

ou seja, $b_{i+1} = b_i * a_{i+1}$ e $c_{i+1} = a_i * a_{i+1}$ para cada $i = 0, 1, ..., n-1$, onde $b_0 = a_0 = l$.

As funções $e_{l,*}$ e $d_{l,*}$ são chamadas transformações e e d de Q^+ baseadas na operação $*$ com o cabeça l respectivamente. Representações gráficas das transformações e e d são mostradas na figura 2.

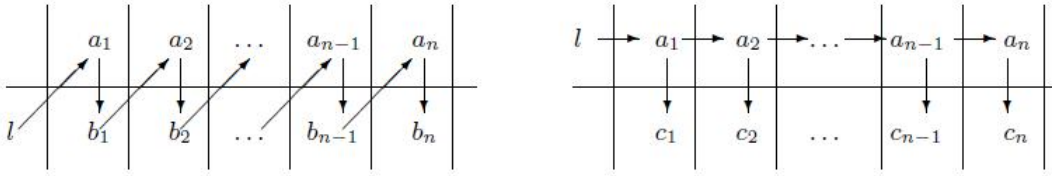

Figura 2: Representação gráfica das transformações $e_{l,*}$ e $d_{l,*}$

Teorema 1. Se $Q, *$ é um quase-grupo finito, então $e_{l,*}$ e $d_{l,\_*}$ são permutações mutuamente inversas de Q^+, ou seja,

$$d_{l,\backslash *}(e_{l,*}(M)) = M = e_{l,*}(d_{l,\backslash *}(M))$$

para cada cabeça $l \in Q$ e para cada string $M \in Q^+$

3.1.1 Quase-grupos como vetor de funções booleanas

Para definir um criptossitemas de chaves públicas baseado em multivariado quadrático, pode-se utilizar a representação de quase-grupos finitos $(Q, *)$ de ordem 2^d por um vetor de funções com valor booleano. Será escolhida a bijeção $\beta : Q \to \{0, 1, ..., 2^d - 1\}$, onde $a \in Q$ possui uma representação de d bits $\beta(a)$. Consequentemente, para cada $a \in Q$ existem unicamente os bits $x_1, x_2, ..., x_d \in \{0, 1\}$ (os quais

dependem da escolha da bijeção β) de tal forma que a é representado por $x_1x_2...x_d$. Logo a e os d bits serão representados como $a = x_1x_2...x_d$ ou $a = (x_1, x_2, ..., x_d)$. A operação $*$ sobre Q pode ser vista como uma operação no vetor $*_{vv} : \{0,1\}^{2d} \to \{0,1\}$ definida como:

$$a * b = c \Longleftrightarrow *_{vv} (x_1, x_2, ..., x_d, y_1, y_2, ..., y_d) = (z_1, z_2, ..., z_d),$$

onde $x_1...x_d, y_1...y_d, z_1...z_d$ são representações binárias de a, b, c respectivamente. Cada z_i depende dos bits $x_1, x_2, ..., x_d, y_1, y_2, ..., y_d$, além de ser unicamente determinado por eles. Então cada z_i pode ser visto como $z_i = f_i(x_1, x_2, ..., x_d, y_1, y_2, ..., y_d)$, onde $f_i : \{0,1\}^{2d} \to \{0,1\}$ depende estritamente e é exclusivamente determinado por $*$.

Lema 1. Para cada quase-grupo $Q, *$ de ordem 2^d e para cada bijeção $Q \to \{0, 1, ..., 2^d - 1\}$ há unicamente um vetor de funções booleanas $*_{vv}$ e d unicamente determinado pelas funções $f_1, f_2, ..., f_d$, tal que para cada $a, b, c \in Q$

$a * b = c \Longleftrightarrow *_{vv}(x_1, ..., x_d, y_1, ..., y_d) =$

$(f_1(x_1, ..., x_d, y_1, ..., y_d), ..., f_d(x_1, ..., x_d, y_1, ..., y_d))$

Cada função booleana $f(x_1, ..., x_k)$ pode ser representada de forma única por sua forma normal algébrica, ou seja, como uma soma de produtos.

$$FNA(f) = a_0 + \sum_{i=1}^{k} a_i x_i + \sum_{1 \le i < j \le k} a_{i,j} x_i x_j + \sum_{1 \le i < j < s \le k} a_{i,j,s} x_i x_j x_s + ..., \qquad (3.3)$$

onde os coeficientes $a_0, a_i, a_{i,j}, ...$ estão no conjunto 0,1 e adição e multiplicação estão no corpo $GF(2)$. No decorrer do texto a função booleana f e FNA considerando $f = FNA(f)$, ou seja, $f = ANF(f)$. Onde os argumentos do polinômio $f = (x_1, ..., x_k)$ são as variáveis indeterminadas $x_1, x_2, ..., x_k$. As formas normais algébricas das funções f_i fornecem informações sobre a complexidade do quase-grupo $(Q, *)$ através dos graus das funções Booleanas f_i. Os graus dos polinômios $FNA(f_i)$ aumentam com a ordem do quase-grupo. Em geral, para um quase-grupo gerado aleatoriamente de ordem $2^d, d \ge 4$, os graus são maiores que 2. Estes quase-grupos não são adequados para a

construção de criptossistemas de chaves públicas baseados em sistemas multivariados quadráticos (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008).

3.1.2 Quase-grupos Multivariados Quadráticos

A seguir será descrita uma classe de quase-grupo, chamado de quase-grupo multivariado quadrático (*multivariate quadratic quasigroups* MQQ) que pode ser de diferentes tipos, em função do número de termos quadráticos e lineares. Para o MQQ é imprescindível obter polinômios multivariados quadráticos (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008).

Definição 3. Um quase-grupo $Q, *$ de ordem 2^d é chamado Quase-Grupo Multivariado Quadrático do tipo $Quad_{d-k}Lin_k$ se exatamente $d - k$ de polinômios f_i são de grau 2 (quadráticos) e k deles são de grau 1 (linear), onde $0 \leq k < d$.

O teorema 2 fornece condições suficientes para um quase-grupo $(Q, *)$ tornar-se MQQ.

Teorema 2. Considere $A_1 = [f_{ij}]_{d \times d}$ e $A_2 = [g_{ij}]_{d \times d}$ duas matrizes d x d de expressões booleanas lineares, e $b_1 = [u_i]_{d \times 1}$ e $b_2 = [v_i]_{d \times 1}$ dois vetores com dimensão d x 1 de expressões booleanas lineares ou quadráticas.

Considere as funções f_{ij} e u_i dependendo unicamente das variáveis $x_1, ..., x_d$, e as funções g_{ij} e v_i dependendo unicamente das variáveis $x_{d+1}, ..., x_{2d}$. Se

$$Det(A_1) = Det(A_2) = 1 \; in \; GF(2) \tag{3.4}$$

e se

$$A_1 * (x_{d+1}, ..., x_{2d})^T + b_1 \equiv A_2 * (x_1, ..., x_{2d})^T + b_2 \tag{3.5}$$

então a operação sobre o vetor ${}_{*vv}(x_1, ..., x_2 d) = A_1.(x_{d+1}, ..., x_{2d})^T + b_1$ define um quase-grupo $(Q, *)$ de ordem 2^d que é MQQ.

Usando o teorema 2 define-se o procedimento MQQ(d,k) para produzir MQQ de

ordem 2^d e de tipo $Quad_{d-k}Lin_k$ como descrito na tabela 1.

Tabela 1: Heurística para encontrar MQQ de ordem 2^5

$MQQ(d,k)$
Entrada: Inteiro d e inteiro $k, 0 \leq k < d$
Saída: Um quase-grupo de ordem 2^d e do tipo $Quad_{d-k}Lin_k$
1. Gerar aleatoriamente uma matriz $A_{1_{dxd}}$ de expressões booleanas lineares das variáveis $x_1, ..., x_d$, de modo que $Det(A_1) = 1$ em $GF(2)$ e o número de constantes (#Const) 0 ou 1 na matriz A_1 satisfaz a inequação $kd \leq \#Const < (k+1)d$. 2. Randomicamente gerar um vetor $b1$ de expressões lineares booleanas com as variáveis $x_1, ..., x_d$. 3. $_{*_{vv}} = A_1.x_2 + b_1$, onde $x_2 = (x_{d+1}, ..., x_{2d})^T$ 4. $_{*_{vv}} = A_2.x_1 + b_2$, onde $x_1 = (x_1, ..., x_d)^T$. 5. Se $(Det(A_2) = 1)$ em $GF(2)$ e ($_{*_{vv}}$ é do tipo $Quad_{d-k}Lin_k$) então retorne ($_{*_{vv}}$) senão retorne para o passo 1.

O procedimento MQQ(d,k) é um algoritmo aleatório para encontrar os MQQ de ordem 2^d e do tipo $Quad_{d-k}Lin_k$. Para $d = 5$ o número médio de tentativas para encontrar MQQ do tipo $Quad_4Lin_1$ é em torno de 2^{15} e para encontrar os MQQ do tipo $Quad_5Lin_0$ é em torno de 2^{16}. No entanto, $MQQ(6,0)$ não fornecem qualquer MQQ de ordem 2^6. Encontrar MQQ de ordem 2^d, $d \geq 6$, é considerado um problema de pesquisa em aberto.

A definição de MQQ implica no seguinte teorema:

Teorema 3. Sendo $x_1 = (f_1, f_2, ..., f_d)$ e $x_2 = (f_{d+1}, f_{d+2}, ..., f_{2d})$ dois vetores de dimensão d com funções booleanas lineares de variáveis $x_1, ..., x_d$. Considerando $(Q, *)$ um quase-grupo multivariado quadrático do tipo $Quad_{d-k}Lin_k$. Se $x_1 * x_2 = (g_1, ..., g_d)$ então no máximo, $d - k$ de polinômios g_i são multivariados quadráticos e pelo menos k polinômios são lineares.

Um processo de geração aleatória de MQQ do tipo $Quad_{d-k}Lin_k$, geralmente o número de polinômios quadráticos é exatamente $d - k$, e o número de polinômios linear é exatamente k. No entanto, há casos raros em que todos os termos quadráticos podem anular-se mutuamente, e o número de polinômios linear será maior do que k enquanto

o número de polinômios quadráticos for menor que $d - k$. No entanto, nestes casos, se ocorrer, pode ser facilmente detectado e quase-grupos com tais propriedades podem ser omitidos da consideração como um dos candidatos para a chave privada.

3.1.3 Bijeção de Dobbertin

Uma parte do algoritmo MQQ utiliza a bijeção de Dobbertin (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008). A função $Dob(X) = X^{2m+1+1} + X^3 + X$ é uma bijeção em $GF(2^{2m+1})$, além de ser multivariada quadrática. A implementação do criptossistemas de chave pública baseada em MQQ utilizada neste trabalho utiliza a bijeção de Dobbertin para m = 6, ou seja uma bijeção em $GF(2^{13})$ (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008).

3.2 Descrição do Algoritmo

Uma descrição genérica para o esquema utilizado é de um típico sistema multivariado quadrático $ToP'oS : \{0,1\}^n \rightarrow \{0,1\}^n$ onde T e S são duas transformações lineares não singulares, e P' é um mapeamento bijetivo multivariado quadrático sobre $\{0,1\}^n$.

O mapeamento $P' : \{0,1\}^n \rightarrow \{0,1\}^n$ é definido pelo algoritmo da tabela 2.

Além disso, o conjunto de índices $I = (i_1, i_2, ..., ik - 1)$, onde $i_j \in \{1, 2, ..., 8\}$, pode ser público ou privado. A segurança do algoritmo não depende do segredo desse conjunto (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008). O algoritmo para gerar a chave pública e privada é descrito na tabela 3.

Na tabela 4 é descrito o algoritmo para decriptar usando chave privada $(T, S, *_1, ..., *_8)$.

O algoritmo para encriptação com a chave pública é a aplicação direta do conjunto de n polinômios multivariados $P = \{P_i(x_1, ..., x_n) | i = 1, .., n\}$ sobre o vetor $x = (x_1, ..., x_n)$, ou seja, $y = P(x)$.

Tabela 2: Definição do mapeamento não linear $P' : \{0,1\}^n \rightarrow \{0,1\}^n$

$P'(n)$
Entrada: Inteiro n, onde $n = 5k, k \geq 28$ e o vetor $x = (f_1, ..., f_n)$ de n funções Booleanas lineares com n variáveis
Saída: Oito quase-grupos $*_1, ..., *_8$ e n polinômios multivariados quadráticos $P'_i(x_1, ..., x_n), i = 1, ..., n$
Pré-processamento: Chamar os procedimentos MQQ(4,1) e MQQ(5,0) gera dois grandes conjuntos $Quad_4Lin_1$ e $Quad_5Lin_0$ (com mais de 2^{20} elementos cada conjunto) de MQQ do tipo $Quad_4Lin_1$ e do tipo $Quad_5Lin_0$ tal que a classificação mínima dos seus polinômios quadráticos quando representado na forma matriz é no mínimo 8; Transformar por permutação as coordenadas de todos quase-grupos no conjunto $Quad_4Lin_1$ tal que a primeira coordenada seja linear.
1. Represente um vetor $x = (f_1, ..., f_n)$ de n funções booleans lineares de n variáveis $x_1, ..., x_n$, como string $x = X_1, ..., X_k$ onde X_i são vetores de dimensão 5. 2. Escolha aleatoriamente diferentes quase-grupos $*_1, *_2 \in Quad_4Lin_1$ e diferentes quase-grupos $*_3, *_4, *_5, *_6, *_7, *_8 \in Quad_5Lin_0$. 3. Defina uma tupla $I = (i_1, i_2, ..., i_{k-1})$ onde $i_j \in 1, 2, ..., 8$, que seria usado como um conjunto de índices para determinar qual quase-grupo seria utilizado na transformação não-linear de y. O requesito para este conjunto de índices é que o número total de índices que são referenciados por um quase-grupo de classe $Quad_4Lin_1$ seja 8. 4. $y = Y_1...Y_k$ onde: $Y_1 = X_1, Y_{j+1} = X_{j*i_j}X_{j+1}$, para $j = 1, 2, ..., k-1$. 5. $Z = Y_1 \| Y_{\mu_1}1 \| Y_{\mu_2}1 \| ... \| Y_{\mu_8}1$, onde todos 13 componentes são funções booleanas linear. A notação $Y_{\mu_j}1$ significa a primeira coordenada do vetor Y_{μ_j}. 6. Transforme Z por bijeção de Dobbertin: $W = Dob(Z)$. 7. $Y_1 = (W_1, W_2, W_3, W_4, W_5), Y_{\mu_1},1 = W_6, Y_{\mu_2},1 = W_7, Y_{\mu_3},1 = W_8, Y_{\mu_4},1 = W_9, Y_{\mu_5},1 = W_{10}$, $Y_{\mu_6},1 = W_{11}, Y_{\mu_7},1 = W_{12}, Y_{\mu_8},1 = W_{13}$ 8. Saída: Quase-grupos $*_1, ..., *_8$ e y as n polinômios multivariados quadráticos $P'_i(x_1, ..., x_n), i = 1, ..., n$

Podendo ser representado como:

$P_i(x_1, ..., x_n) = a_{i,0,0} + \sum_{j=1}^{n} a_{i,j,0}x_j + \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} a_{i,j,k}x_{k-j}x_k$, onde $a_{i,j,k} \in \{0,1\}$.

Significando que a encriptação pode ser representada como:

$$y = P(x) \equiv y \equiv A.X$$

A matriz booleana A, representada na figura 3, é a chave pública e é uma matriz com dimensão $n \times 1 + n + \frac{n(n-1)}{2}$. O vetor X, representado na figura 4, possui dimensão $[(1 + n + \frac{n(n-1)}{2})\ , 1]$ e é obtido do vetor $x = x_1, ..., x_n$.

Tabela 3: Algoritmo para gerar chave pública e privada

Algoritmo para gerar chave Pública e Privada para MQQ
Entrada: Inteiro n, onde $n = 5k, k \geq 28$
Saída: Chave Pública $\boldsymbol{P}$: n polinômios multivariados quadráticos $P_i(x_1, ..., x_n)$, i=1,...,n, Chave Privada: Duas matrizes booleanas T e S não singulares de ordem n x n e oito quase-grupos $*_1, ..., *_8$
1. Gerar duas matrizes booleanas T e S não sigulares de ordem n x n (uniformemente randômicas) 2. Chamar $P'(n) : \{0, 1\}^n \rightarrow 0, 1^n$ e obter os quase-grupos $*_1, ..., *_8$. 3. $y = T(P'(S(x)))$ onde $x = (x_1, ..., x_n)$ 4. Saída: A chave pública é y com n polinômios multivariados quadráticos $P_i(x_1, ..., x_n), i = 1, ..., n$ e a chave privada é a tupla $(T, S, _{*_1}, ..., _{*_8})$

Tabela 4: Algoritmo para decriptar e assinar

Algoritmo para decriptar/assinar com chave privada $T, S, *_1, ..., *_8$
Entrada: Um vetor $y = y_1, ..., y_n$.
Saída: Um vetor $x = (x_1, ..., x_n)$ tal que $P(x) = y$
1. $y' = T^{-1}(y)$. 2. $W = y'_1, y'_2, y'_3, y'_4, y'_5, y'_6, y'_{11}, y'_{16}, y'_{21}, y'_{26}, y'_{31}, y'_{36}, y'_{41}$. 3. $Z = Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8, Z_9, Z_{10}, Z_{11}, Z_{12}, Z_{13} = Dob^{-1}(W)$. 4. $y'_1 \longleftarrow Z_1, y'_2 \longleftarrow Z_2, y'_3 \longleftarrow Z_3, y'_4 \longleftarrow Z_4, y'_5 \longleftarrow Z_5, y'_6 \longleftarrow Z_6,$ $y'_{11} \longleftarrow Z_7, y'_{16} \longleftarrow Z_8, y'_{21} \longleftarrow Z_9, y'_{26} \longleftarrow Z_{10}, y'_{31} \longleftarrow Z_{12}, y'_{41} \longleftarrow Z_{13}$. 5. $y' = Y_1...Y_k$ onde Y_i são vetores de dimensão 5. 6. Sendo $_{*_i}$, $i = 1, ..., 8$, obter $x' = X_1...X_k$, de modo que, $X_1 = Y_1, X_2 = X_{1\setminus 1}Y_2, X_3 = X_{2\setminus 2}Y_3$ e $X_i = X_{i-1\setminus_{3+((i+2)\ mod\ 6)}Y_i}$ 7. $x = S^{-1}(x')$

$$A = \begin{bmatrix} a_{1,0,0} & a_{1,1,0} & ... & a_{1,n,0} & a_{1,1,2} & a_{1,1,3} & ....a_{1,n-1,0} \\ a_{2,0,0} & a_{2,1,0} & ... & a_{2,n,0} & a_{2,1,2} & a_{2,1,3} & ....a_{2,n-1,0} \\ . & . & . & . & . & . & .. \\ . & . & . & . & . & . & .. \\ . & . & . & . & . & . & .. \\ . & . & . & . & . & . & .. \\ . & . & . & . & . & . & .. \\ a_{n,0,0} & a_{n,1,0} & ... & a_{n,n,0} & a_{n,1,2} & a_{n,1,3} & ....a_{n,n-1,0} \end{bmatrix}.$$

Figura 3: Chave Pública A

$$X = \begin{bmatrix} 1 \\ x_1 \\ . \\ . \\ . \\ . \\ . \\ x_n \\ x_1 x_2 \\ x_2 x_3 \\ . \\ . \\ . \\ . \\ . \\ x_{159} x_{160} \\ x_1 x_3 \\ x_2 x_4 \\ . \\ . \\ . \\ . \\ . \\ x_{158} x_{160} \\ . \\ . \\ . \\ . \\ . \\ x_1 x_{159} \\ x_2 x_{160} \\ x_1 x_{160} \end{bmatrix}$$

Figura 4: Vetor X

4 TESTES E COMPARAÇÃO COM OUTRO PKC

No capítulo 3, foi feita uma descrição detalhada do MQQ e o embasamento matemático necessário para a compreensão dos conceitos relacionados a quase-grupos multivariados quadráticos. Neste capítulo, é apresentada a prova de conceito do MQQ, em plataforma real de redes de sensores sem fio. O tempo de execução do MQQ é comparado com o RSA para ter uma relação de desempenho entre o MQQ e outro PKC. O algoritmo RSA (RIVEST; SHAMIR; ADLEMAN, 1978) foi escolhido pela facilidade de implementá-lo quando comparado ao algoritmo ECC (KOBLITZ, 1987).

São comparados os procedimentos de encriptação e decriptação do MQQ e RSA, onde o tamanho das chaves criptográficas utilizadas nos experimentos baseia-se nos documentos (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008) que confirmam o mesmo nível de segurança do MQQ com chaves de 160 bits e RSA com chaves de 1024 bits.

Para analisar se o MQQ é viável em uma plataforma de sensores foi coletado o espaço de memória ocupado pelo MQQ em RAM e ROM nas plataformas TelosB (CROSSBOW, 2008b) e MICAz (CROSSBOW, 2008a). Analisar o espaço ocupado em memória é imprescindível para avaliar se o MQQ pode ser uma proposta viável para RSSF, afinal é necessário haver espaço na memória do sensor para executar aplicações úteis.

4.1 Plataformas Utilizadas

Nos experimentos foi escolhida a plataforma de RSSF CrossBow TelosB (CROSSBOW, 2008b), o qual é uma plataforma aberta desenvolvida para permitir experimentos e estudos em laboratórios. Possui como recursos programação via USB, rádio com antena integrada compatível com IEEE 802.15.4, taxa de transmissão de dados a 250 kbps. Possui o micro-controlador TI MSP430 fabricado pela Texas Instruments com 8 MHz, arquitetura RISC de 16 bits, 10 KBytes de RAM e 48 KBytes de ROM para programa, além de suportar TinyOS. Além disso, possui como opcionais os sensores integrados de temperatura, umidade, luminosidade. Outra plataforma de RSSF utilizada foi o CrossBow MICAz (CROSSBOW, 2008a), o qual possui taxa de transmissão de 250 kbps e a taxa de transmissão do módulo de rádio pode variar de 2,4 até 2,48 GHz. Possui processador ATmega128L, arquitetura de 8 bits, 4 KBytes de RAM e 128 KBytes de ROM para programa, além de suportar TinyOS. É importante ressaltar que a memória flash programável existente nas duas plataformas é chamada de ROM.

O sistema operacional utilizado nos experimentos é o TinyOS 2.0.2, possui código fonte aberto e foi desenvolvido para redes de sensores sem fio. Apresenta uma arquitetura baseada em componentes que permite uma rápida inovação e implementação, minimizando o tamanho do código. Essa característica é importante considerando as restrições de memória inerentes às RSSF. Seu modelo de execução orientado a eventos permite um melhor gerenciamento de energia, além de permitir a flexibilidade de programação necessária considerando a natureza imprevisível de comunicação sem fio (TINYOSFORUM, 2009).

A linguagem de programação utilizada é o NesC *"Network Embedded Systems C"*, possui sintaxe semelhante ao C, mas com recursos para suportar a estrutura e o modelo de execução das aplicações em TinyOS (LEVIS, 2006). Os procedimentos de encriptar e decriptar foram escritos em C ANSI, onde os fontes em NesC fazem a chamada

aos procedimentos em C ANSI. É importante considerar que o módulo de encriptação tanto do RSA quanto MQQ foram escritos em C ANSI para utilizar o mesmo critério para os dois algoritmos.

O ambiente de desenvolvimento utilizado foi a distribuição XubunTOS (XUBUNTOS, 2004), sendo utilizado o simulador TOSSIM (LEVIS; LEE, 2003) para realizar testes e validar a implementação. Posteriormente os fontes da encriptação e decriptação foram executados nas plataformas TelosB e MICAz.

4.2 Implementação do MQQ

Os procedimentos para encriptar e decriptar do MQQ e RSA foram escritos em C (ANSI) e posteriormente sendo chamados em código nesC, não tendo sido feitas otimizações visando executar em determinada plataforma de sensor MICAz ou TelosB.

Foi utilizada a distribuição linux XubunTOS o qual traz um ambiente já configurado com os pacotes necessários para trabalhar com TinyOS e NesC (XUBUNTOS, 2004).

Foram criados em NesC os componentes MqqAppC e MqqC, onde MqqAppC é um componente que provê toda a configuração para a aplicação e conecta outros componentes tais como:

- MqqC implementa algumas interfaces e realiza a chamada do procedimento para encriptar e decriptar do MQQ, onde os procedimentos são escritos em C padrão ANSI.
- TimerMilliC usado para medir o tempo de execução, em milissegundos, dos procedimentos a serem testados.
- MainC componente possui interface Boot que tem o evento *booted* implementado para realizar a chamada dos procedimentos encriptar e decriptar.

- PrintfC é utilizado para exibir os tempos medidos.

Para debugar a implementação do MQQ e RSA foi utilizado o simulador TOSSIM (LEVIS; LEE, 2003), onde as linhas que seriam necessárias debugar acrescenta-se a variável DBG. Por exemplo: *dbg("Mqq", "Valor: %d", variavel);*.

Para automatizar a compilação foi utilizado o arquivo *Makefile*, tendo sido adicionados no *Makefile* que o componente principal da aplicação é MqqAppC e foi adicionada a flag *-I$ (TOSDIR)/lib/printf* para permitir o uso do componente PrintfC.

Detalhes sobre os códigos escritos em NesC, os códigos escritos em C padrão ANSI, o arquivo Makefile utilizado para automatizar a compilação da aplicação e o script em python utilizado para debugar em TOSSIM encontram-se no apêndice A. A linguagem python é disponibilizada pelo simulador TOSSIM para executar as aplicações em NesC.

4.3 Encriptação

O algoritmo para encriptação é a multiplicação de um conjunto de n polinômios multivariados **P**=$\{P_i(x_1, ..., x_n)|i = 1, ..., n\}$ sobre um vetor $x = (x1, x2, ..., x_n)$, ou seja $y = P(x)$ (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008).

Nesta implementação cada polinômio P_i é interpretado com seus coeficientes $c_i \in \{0, 1\}$,

Exemplo: Tendo $P_i = c_0 + c_1 \times x_1 + c_2 \times x_2 + c_3 \times (x_1 \times x_2)$ logo $P_i = 0110$ equivale a $P_i = x_1 + x_2$

Para a implementação da encriptação do MQQ, P foi representado como matriz e x como vetor, onde $y = P(x)$ foi representado com o resultado da multiplicação de P por x, onde as operações de soma são representados por uma operação de XOR entre

bits e a multiplicação de duas variáveis é representada por uma operação AND entre bits.

Exemplo: Tendo $x = \{x_1 = 1, x_2 = 0, x_3 = 1\}$ e o polinômio P_1

$P_1(x) = (x_1 \times x_2 + x_1 \times x_3)$

$P_1(x) = (x_1$ AND x_2 XOR x_1 AND $x_3)$

$P_1(x) = 1$ AND 0 XOR 1 AND $1 = 0$ XOR 1

$P_1(x) = 1$

Pode-se representar $y = P(x)$ conforme segue:

$P_i(x_1, ..., x_n) = a_{i,0,0} + \sum_{j=1}^{n} a_{i,j,0}x_j + \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} a_{i,j,k}x_(k - j)x_k$,

onde $a_{i,0,0} \in \{0, 1\}$.

Significa que $y = P(x)$ equivale a $y = A \times X$.

Neste caso A é a chave pública do MQQ e representa os coeficientes existentes em cada polinômio, ou seja termo inexistente no polinômio terá coeficiente igual 0. Por possuir coeficientes para n polinômios e para a permutação dos n termos $x_1, ..., x_n$ de um polinômio, logo a matriz A possui a dimensão de $A_{n\times(1+n+\frac{n\times(n-1)}{2})}$.

Isto significa que para uma implementação do MQQ com chaves de 160 bits a matriz e chave pública A ficaria com dimensão $A_{160\times12881}$, o que daria um tamanho de $160 \times 12881 = 2060960$ bits, transformando em kilobytes $(2060960 \div 8) \div 1024 = 251,58$ KBytes.

X é um vetor de dimensão $X_{12881\times1}$, sendo obtido da permutação de $x = (x_1, ..., x_n)$. Logo o processo de encriptação pode ser descrito como:

$y(y_1, ..., y_n) = A_{160\times12881} \times X_{12881\times1}$

Conforme (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008), y=$A \times X$ representa o processo de encriptação, de acordo com a figura 5.

Mas é importante realizar algumas considerações sobre o tamanho das chaves e as

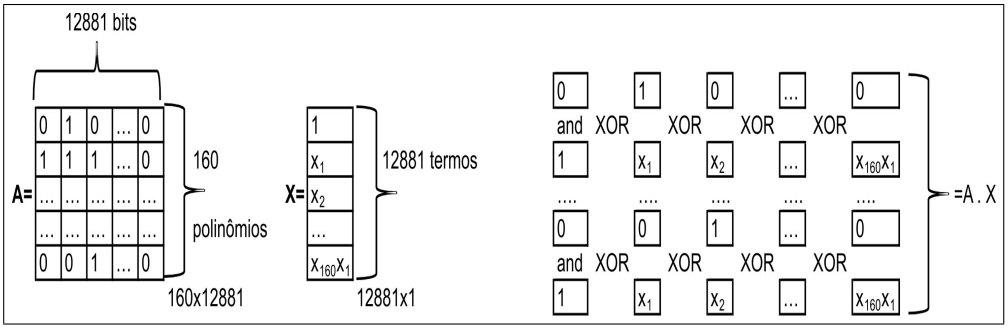

Figura 5: Representação da multiplicação das chave pública A com o vetor X

instruções utilizadas na encriptação.

A primeira consideração refere-se às instruções utilizadas na encriptação que são resumidas a operação AND e XOR de bits, que são instruções simples de serem executadas em um nó sensor.

Considerando o tamanho da chave pública de 251,58 KBytes inviabiliza a alocação destas chaves na memória de um sensor, para isso foi necessário diminuir o tamanho da chave pública, e neste trabalho usou-se o fato que um termo pode estar redundante em vários polinômios. Logo é utilizado um vetor de tamanho 12881 bits, caso um dos bits de A seja 0, então significa que o termo não existe nos polinômios.

Um bit de A sendo 1 significa que o termo existe em alguns polinômios, tendo sido criado um vetor auxiliar de 160 bits que informa em qual dos 160 polinômios o termo está presente. No vetor auxiliar quando o bit é 1 significa que o termo encontra-se em determinado polinômio, conforme representado na figura 6.

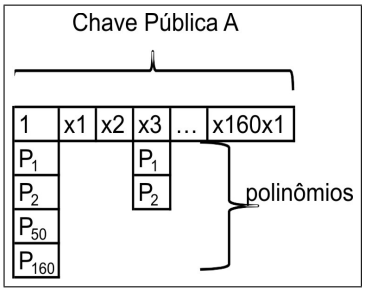

Figura 6: Proposta nova de representação para chave pública utilizada no MQQ

Nesta solução, os 12881 bits representados pela permutação dos termos $(x_1, ..., x_n)$

estão fixos na memória ROM do sensor, no entanto os 160 bits do vetor auxiliar são alocados dinamicamente, informando então em qual polinômio o termo está presente.

Na nova abordagem a matriz A é representada em um vetor de tamanho A[12881/16], ficando A[806] um vetor de 16 bits, sendo que a estrutura auxiliar possui tamanho $Auxiliar[160/16]$ o que equivale a um vetor $Auxiliar[10]$ de 16 bits. A vantagem desta abordagem consiste em diminuir o tamanho necessário para armazenar a chave pública, podendo em alguns casos ótimos diminuir a chave pública de 251,58 KBytes para $(12881 \div 8) \div 1024 = 1,57$ KBytes. Portanto, no cenário que é implementar a encriptação do MQQ em redes de sensores sem fio a maior dificuldade está em encontrar mecanismos para diminuir o tamanho da chave pública, podendo ser utilizadas outras formas de representação, tal como matrizes esparsas.

4.4 Decriptação

A definição do algoritmo de decriptação do MQQ é descrita na tabela 4 do capítulo 3.

O algoritmo para decriptar utiliza uma chave privada composta de duas matrizes não singulares T e S e oito quase-grupos $*_1, *_2, *_3, *_4, *_5, *_6, *_7, *_8$. As matrizes $T_{n\times n}$ e $S_{n\times n}$, no caso desta implementação as dimensões ficam $T_{160\times160}$ e $S_{160\times160}$. Portanto, para armazenar as matrizes T e S utiliza-se uma estrutura que armazene $2 \times (160 \times 160) = 51200$ bits, ficando $(51200 \div 8) \div 1024 = 6,25$ KBytes.

Cada um dos oito quase-grupos $*_1, ..., *_8$ é uma matriz de dimensão de 32 x 32 = 1024, onde uma das 1024 posições de um quase-grupo possui 5 bits.

Logo a dimensão de um dos oito quase-grupos é $32 \times 32 \times 5 = 5120$ bits, portanto para armazenar os oito quase-grupo é necessário um espaço de $8\times (32^2 \times 5) = 40960$ bits o que ficaria $(40960 \div 8) \div 1024 = 5$ KBytes.

Ao todo, a chave privada necessita de 11,25 KBytes para ser armazenada e para se

tornar viável a implementação da encriptação a chave privada foi colocada na memória ROM do sensor. Mas é importante considerar que dentro de cada quase-grupo há termos de polinômios redundantes abrindo espaço para implementações futuras que otimizem o espaço ocupado pelos quase-grupos.

A entrada para o algoritmo de decriptação consiste em um vetor y de 160 bits representando a mensagem encriptada. Enquanto a saída do algoritmo é representada por um vetor x de 160 bits representando a mensagem decriptada. Sendo y e x representados em um vetor de 16 bits com 10 posições.

O passo 1 da tabela 4 consiste na multiplicação da matriz inversa de $T^{-1}_{160\times160}$ com o vetor de entrada $y_{160\times1}$. Nesta implementação a matriz inversa T^{-1} está fixa na memória ROM do sensor, sendo representada como uma matriz de 16 bits com dimensão $T^{-1}_{160\times16}$. As operações da multiplicação e soma consistem em ANDs e XOR entre os bits de T^{-1} e y, sendo o resultado atribuído para o vetor y'.

O passo 7 da tabela 4 consiste do mesmo procedimento do passo 1, sendo que ao invés de $y' = T^{-1}(y)$ ficaria $x = S^{-1}(x')$.

Para a implementação do MQQ em sensores, as matrizes $T^{(}-1)$ e $S^{(}-1)$ são fixas na memória ROM do sensor.

O passo 2 tem por objetivo obter 13 bits do vetor $y' = T^{-1}(y)$, sendo estes 13 bits dispostos no vetor W. O vetor W recebe cada um dos bits $y'_1, y'_2, y'_3, y'_4, y'_5, y'_6, y'_{11}$, $y'_{16}, y'_{21}, y'_{26}, y'_{31}, y'_{36}, y'_{41}$ do vetor y'.

O passo 3 consiste em uma pesquisa na matriz inversa de Dobbertin Dob^{-1}, tendo como parâmetro de entrada o vetor W de 13 bits. O resultado desta pesquisa na matriz Dob^{-1} será atribuído para o vetor Z de 13 bits. A dimensão Dob^{-1} é $13 \times 2^{13} = 106496$, totalizando 13 KBytes.

O passo 4 consiste em atribuir cada um dos bits do vetor Z novamente para o vetor y'.

No quinto passo os bits de y' são organizados de 5 em 5 bits e dispostos no vetor $Y = Y_1, ..., Y_{160/5} = Y_1, ..., Y_{32}$, sendo Y um vetor de dimensão 32x5.

O sexto passo tem como resultado o vetor $x' = X_1, ..., X_{32}$ de dimensão 32 x 5, onde cada X_i possui 5 bits resultantes da pesquisa nos quase-grupos $*1, ..., *8$. Para obter um elemento de determinado quase-grupo é passado um valor de 10 bits representando o endereço de um elemento do quase-grupo.

Na decriptação os dados armazenados na ROM do sensor foram T^{-1}, S^{-1}, Dob^{-1} e os quase-grupos $*_1, ..., *_8$ dando um total de estruturas estáticas a serem armazenadas de $(2 \times 160^2) + (13 \times 2^{13}) + (8 \times 32^2 \times 5) = 198656$ bits, ou seja 24,25 KBytes.

Considerando que nos quase-grupos podem existir termos redundantes nesta implementação não foram feitas otimizações para diminuir o espaço ocupado pelos quase-grupos.

4.5 Resultados Obtidos com o MQQ

Os resultados foram obtidos da execução dos módulos de encriptação e decriptação nas plataformas TelosB e MICAz, tendo medido o tempo de execução e a memória utilizada. Foram obtidas 10 amostras do tempo de execução dos procedimentos de encriptação e decriptação nas plataformas TelosB e MICAz, utilizando o MQQ com chaves de 160 bits.

Nas implementações foi utilizado o componente nesC Timer para verificar o tempo de execução de cada módulo, tendo sido utilizado o componente TimerMilliC e a interface Timer<TMilli> para obter o tempo de execução em milissegundos. O comando getNow() é utilizado para obter o tempo em determinado instante. O tempo de execução de cada procedimento foi determinado a partir da diferença dos valores resultante de duas chamadas do comando getNow(), antes e depois das chamadas dos métodos encriptar e decriptar do MQQ.

Para medir o espaço ocupado em RAM e ROM pelos procedimentos de encriptação e decriptação, os componentes TimerMilliC e PrintfC foram retirados, com intuito de conseguir uma medição mais precisa do espaço ocupado pelos procedimentos de encriptar e decriptar.

Tabela 5: Espaço ocupado pelos procedimentos encriptar e decriptar do MQQ

Plataforma	TelosB		MICAz	
MQQ	Encriptação	Decriptação	Encriptação	Decriptação
RAM (bytes)	26	28	1658	31024
ROM (bytes)	3436	34582	2778	33748
RAM (KBytes)	0,025	0,027	1,61	30,29
ROM (KBytes)	3,35	33,77	2,71	32,95

De acordo com a tabela 5 o espaço ocupado pela RAM e ROM no processo de encriptação e decriptação no MICAz estão maiores que os obtidos para a plataforma TelosB, por dois motivos. O primeiro deve-se ao fato da quantidade de matrizes estáticas armazenadas na memória do sensor na decriptação ser maior, devendo-se considerar que nenhuma otimização foi feita no sentido de diminuir o tamanho dos quase-grupos que possuem termos redundantes. O segundo fato à implementação, pois na declaração das matrizes foi colocada a diretiva *const* e quando o TelosB encontra esta diretiva ele aloca estas estruturas automaticamente na ROM o que não ocorre com o MICAz.

Pode-se observar na tabela 6 o módulo decriptação não executou no MICAz, faltando modificar a decriptação para alocar as estruturas estáticas necessárias na ROM. Sendo que por não haver otimização para o código do MQQ para MICAz não foi possível colocar todas as constantes na memória ROM do sensor MICAz. Na plataforma TelosB para colocar um dado na ROM apenas necessita declarar a variável com *const*.

Nesta seção foram exibidos os resultados somente dos experimentos com MQQ, e o método para coletar o tempo foi através do componente TimerMilliC e PrintfC que permite obter o tempo em milissegundos e o outro componente permite exibir este tempo para que seja tabulado.

Tabela 6: Tempo de execução do procedimentos encriptar e decriptar do MQQ

Plataforma	TelosB		MICAz
MQQ	Encriptação	Decriptação	Encriptação
amostra 1 (milissegundos)	825	117	445
amostra 2 (milissegundos)	822	116	445
amostra 3 (milissegundos)	826	117	445
amostra 4 (milissegundos)	826	116	445
amostra 5 (milissegundos)	823	117	445
amostra 6 (milissegundos)	827	117	445
amostra 7 (milissegundos)	827	116	445
amostra 8 (milissegundos)	825	117	445
amostra 9 (milissegundos)	825	116	445
amostra 10 (milissegundos)	825	117	445
Média (milissegundos)	825,1	116,6	445
Desvio Padrão (milissegundos)	1,6	0,5	0

Os espaço ocupado em memória (RAM e ROM) pelo MQQ foram obtidos através da execução do arquivo *Makefile*, descrito no apêndice A, onde foram compilados para as plataformas MICAz e TelosB.

4.6 Resultados Obtidos com MQQ e RSA

Nesta seção são obtidos os tempos do MQQ e RSA, onde foi foi utilizado um método para coletar o tempo baseado utilizando um multímetro digital (MARGI et al., 2010). A escolha por este outro método ocorre pelo fato que o tempo(milissegundos) do RSA extrapolar o tamanho da variável que armazena o tempo do componente TimerMilliC.

4.6.1 Método utilizado para medição

A idéia consiste em medir a corrente consumida pelo sensor durante determinado período de tempo para uma tarefa em execução. Quando o sensor não possui uma tarefa sendo executada a corrente consumida é X e quando uma tarefa entra em execução a corrente varia para X+Y, posteriormente, o sensor voltando para o estado *idle* (não

há tarefas específicas sendo executadas) o consumo volta a ser X. Medindo a corrente a cada instante de tempo e considerando a evidente variação de corrente quando o sensor alterna do estado *idle* para o estado ativo, pode-se obter a variação do tempo que o sensor esteve ativo ou executando determinada tarefa.

4.6.2 Plataforma utilizada

Neste experimento, a plataforma de RSSF utilizada neste experimento será TelosB. Utiliza-se medição direta no TelosB para obter uma medição mais precisa. Foi utilizada a fonte de alimentação Agilent E3631A (AGILENT, 2009) configurada para fornecer 3,00 volts para TelosB. O multímetro digital Agilent 34401A (AGILENT, 2007) foi utilizado para medir o fluxo de corrente do TelosB durante a execução dos algoritmos MQQ e RSA. O multímetro foi configurado para fornecer a leitura da corrente consumida a uma taxa de 60 Hz, onde o multímetro foi colocado em série com TelosB para medir a corrente a cada instante de tempo. Foi utilizado um cabo GPIB para conectar o multímetro a um computador executando o software LabView (INSTRUMENTS, 2009), o qual coleta e armazena as amostras do tempo em milissegundos e corrente em ámpere. A figura 7 representa as plataformas utilizadas neste método.

Os fontes do MQQ e RSA neste experimento não utilizaram os componentes TimerMilliC e PrintfC, afinal os tempos já são coletados pelo multímetro digital.

4.6.3 Tempo

Neste experimento para ter idéia do intervalo de tempo que deveria ser considerado é importante saber qual o valor da corrente enquanto o sensor estivesse ativo (executando o algoritmo). Conforme especificação do TelosB (CROSSBOW, 2008b) o valor da corrente no momento que o micro-controlador do TelosB (MSP430) estiver ativo é aproximadamente 1,8mA e o valor da corrente quando o micro-controlador não estiver em execução, modo de baixo consumo (*sleep*), é 5,1μA. Subtraindo o intervalo de

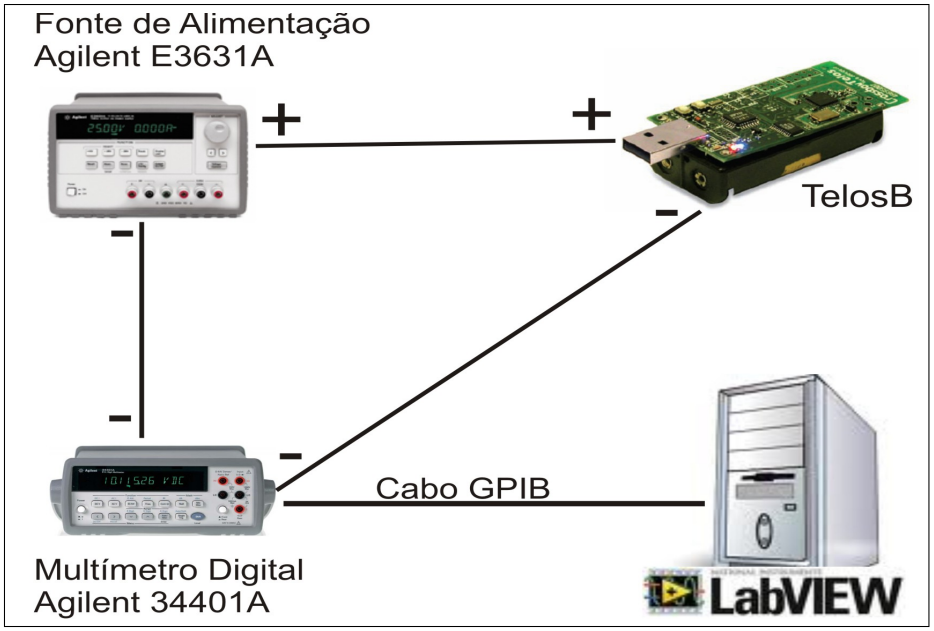

Figura 7: Método utilizado na medição dos tempos de MQQ e RSA na plataforma TelosB

tempo nos quais o micro-controlador do TelosB esteve ativo, foram obtidos os intervalos de tempo para o MQQ e RSA. Além disso, o tamanho da chave utilizada no MQQ é 160 bits e do RSA 1024 bits.

Na tabela 7 há somente os tempos na execução dos módulos de encriptação e decriptação do MQQ e RSA.

Tabela 7: Tempo MQQ e RSA

Plataforma	TelosB	
Algoritmo	Encriptação	Decriptação
MQQ (milissegundo)	843,5	125
RSA (milissegundo)	767046,0	683768,75

4.7 Análise dos Resultados Obtidos

Nesta seção são analisados os resultados obtidos com os experimentos com MQQ e RSA. O espaço ocupado em memória pelo MQQ foi analisado tomando como base o limite de memória RAM e ROM das plataformas TelosB e MICAz.

O tempo de execução do MQQ é analisado usando dois métodos de coleta de dados nas plataformas TelosB e MICAz. Primeiramente é analisado o tempo obtido pelos procedimentos de encriptação e decriptação do MQQ individualmente, utilizando os componentes NesC PrintfC e TimerMilliC. Posteriormente os tempos de execução do MQQ são comparados com o do RSA, sendo estes tempos coletados utilizando um multímetro digital.

4.7.1 Memória

Na tabela 8 há o percentual ocupado pelo MQQ nas plataformas TelosB e MICAz, onde podemos verificar o percentual de ocupação em memória (RAM e ROM) dos procedimentos de encriptação e decriptação do MQQ nas plataformas TelosB e MICAz.

Tabela 8: Percentual espaço ocupado em memória pelos procedimento encriptar e decriptar

Plataforma	TelosB		MICAz	
MQQ	Encriptação	Decriptação	Encriptação	Decriptação
RAM %	0,25	0,27	40,25	757,25
ROM %	6,97	70,35	2,11	25,74

Nas figuras 8 e figura 9 há a relação entre o espaço ocupado em memória pelo MQQ e o tamanho da memória disponível nas plataformas TelosB e MICAz.

Podemos verificar que na plataforma TelosB o consumo de RAM pelo MQQ é ínfimo. O consumo de ROM no TelosB é maior na decriptação com relação a encriptação, onde a maior concentração na ROM deve-se ao uso da diretiva *const* que possibilita colocar dados na ROM. A decriptação necessita de maior espaço de memória devido a necessidade das estruturas de dados representadas pelo quase-grupos, as matrizes inversas T, S e a inversa de Dobbertin. Considerando que o objetivo deste trabalho é uma comparação justa do MQQ nas plataformas testadas, logo não foi feita otimização específica para MICAz com o intuito de colocar as estruturas estáticas na

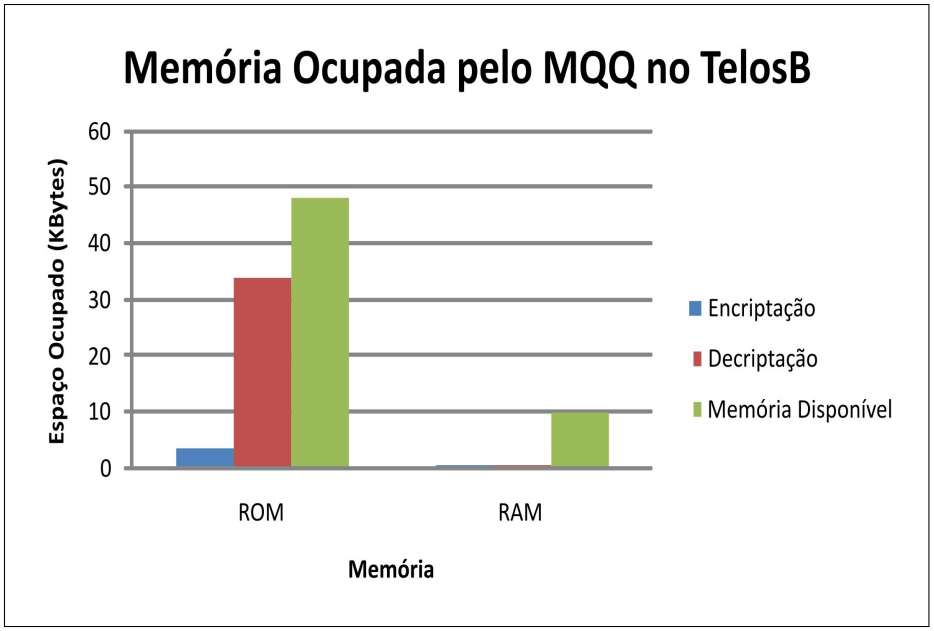

Figura 8: Memória ocupada pelo MQQ na plataforma TelosB

ROM. É importante salientar que este trabalho não abrange a otimização do algoritmo proposto por (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008), mas considerando que os quase-grupos podem possuir elementos repetidos é possível diminuir o espaço necessário para armazenar estes quase-grupos.

Na plataforma TelosB, o MQQ é promissor afinal o maior consumo de memória é na ROM, além disso há perspectivas de diminuir o tamanho ocupado pelos estruturas de dados utilizadas MQQ na encriptação e decriptação. Na plataforma MICAz também é promissor, afinal colocando as estruturas de dados na ROM o espaço não será crítico na plataforma MICAz.

No experimento com a decriptação com MICAz o espaço ocupado pela decriptação extrapolou a memória disponível, tanto que na seção 4.5 não foi possível executar e por conseguinte obter o tempo de execução do MQQ. Mas como foi comentado o problema com MICAz pode ser contornado colocando as estruturas de dados na ROM e melhorado otimizando o espaço ocupado pelas estruturas de dados do MQQ.

É importante salientar que é possível reduzir o tamanho das estruturas de dados necessárias pelo MQQ, pois os polinômios geralmente possuem termos repetidos. Logo

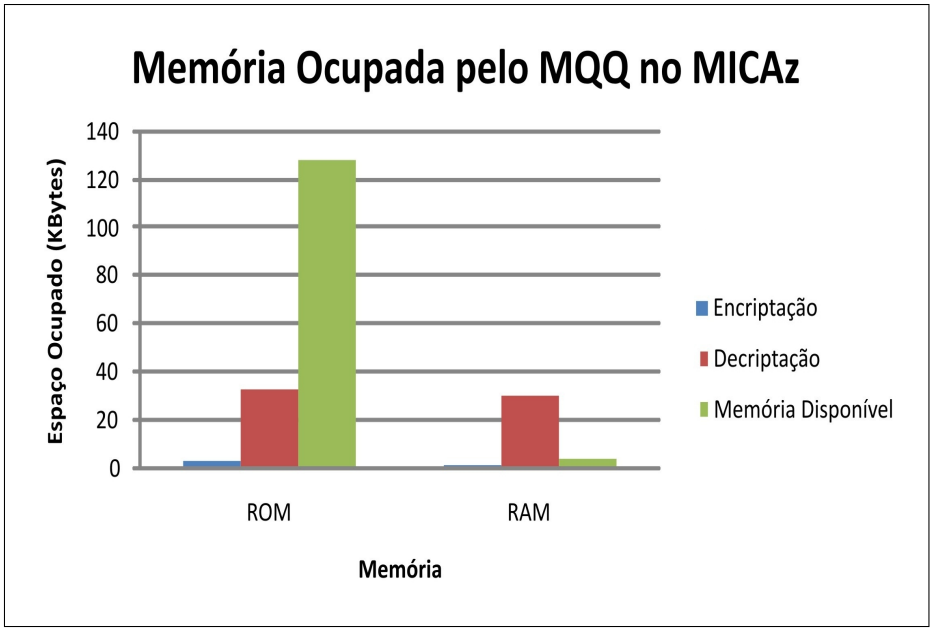

Figura 9: Memória ocupada pelo MQQ na plataforma MICAz

há espaço para trabalhos futuros com intuito de diminuir o tamanho das estruturas de dados utilizadas pelo MQQ.

Reduzir o espaço ocupado pelo MQQ é primordial para aplicações reais que necessitem de segurança em RSSF, afinal na pouca memória disponível no sensor deverá coexistir tanto a aplicação real de RSSF quanto os algoritmos criptográficos.

4.7.2 Tempo

Considerando o tempo de execução há os resultados descritos nas seções 4.5 e 4.6, onde a seção 4.5 possui o tempo do MQQ obtido através do componente NesC TimerMilliC e na seção 4.6 o tempo do MQQ e RSA na plataforma TelosB são obtidos através de um multímetro digital que contabiliza o tempo em que o sensor esteve em modo ativo (executando o MQQ ou RSA).

Na figura 10 podemos ter uma relação do tempo de execução nas duas plataformas testadas. Conforme podemos verificar no TelosB, o tempo para encriptar é aproximadamente sete vezes maior que o tempo para decriptar. Além disso, verifica-se que o tempo para encriptar na plataforma MICAz é metade do que foi coletado na plataforma

TelosB.

Considerando a relação de desempenho no TelosB, pode-se estimar que o tempo para decriptar no MICAz seria aproximadamente 63,00 milissegundos. É importante salientar que a decriptação no MICAz é apenas uma estimativa com base na execução do MQQ no TelosB.

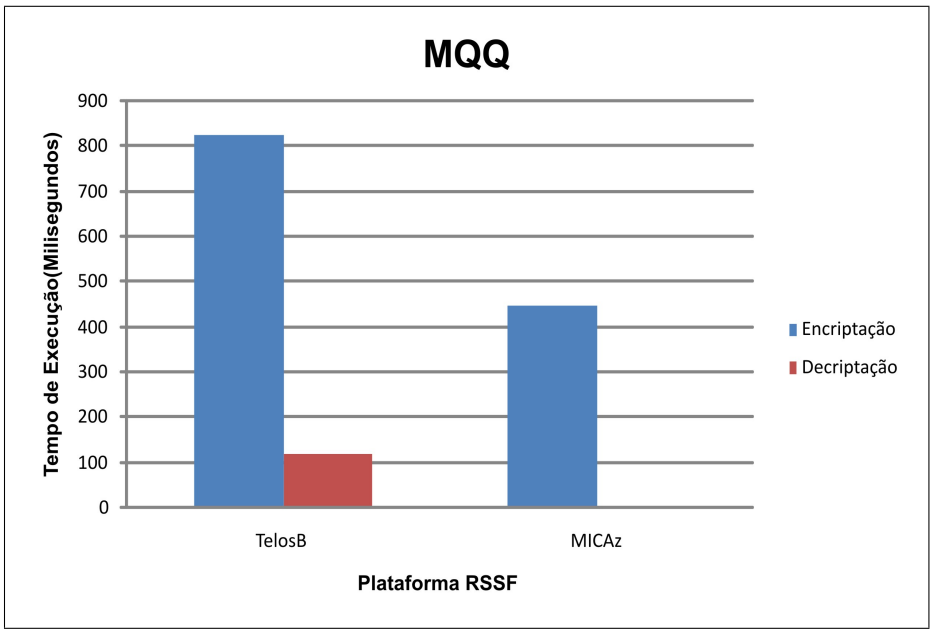

Figura 10: Tempo de execução do MQQ nas plataformas TelosB e MICAz

Relacionando os dois métodos de coleta do tempo a diferença de tempos do MQQ no TelosB é 18,4 milissegundos na encriptação e 8,4 milissegundos na decriptação. Sendo a diferença dos tempos do MQQ obtidos nos dois métodos de coleta de 97,81% na encriptação e 93,28% na decriptação.

Conforme podemos verificar na tabela 7 a relação de desempenho entre MQQ e RSA, verifica-se que o MQQ é aproximadamente 909 vezes mais rápido na encriptação e 5470 vezes mais rápido na decriptação. Mas é importante ressaltar que a mensagem encriptada pelo MQQ é de 160 bits, enquanto a mensagem encriptada pelo RSA é de 8 bits. Ou seja o tempo do RSA encriptando e decriptando uma mensagem de 160 bits é bem maior que o tempo medido. Aumentando a mensagem a ser cifrada pelo RSA a relação de desempenho MQQ e RSA aumenta consideravelmente. Ficando como

trabalho futuro analisar MQQ e RSA usando mesmo tamanho de mensagem, além de ser interessante analisar a relação de desempenho MQQ e ECC em uma plataforma de RSSF.

5 CONSIDERAÇÕES FINAIS

Este trabalho analisa uma nova abordagem para dotar redes de sensores sem fio de criptossistemas de chaves públicas, através do MQQ. No capítulo 2 foi mostrado o conceito sobre redes de sensores sem fio, bem como o entendimento da importância da segurança fornecida por PKC para aplicações em RSSF, além do entendimento dos impactos na utilização de PKC para o cenário com recurso (processamento, memória e energia) limitado das RSSF.

Neste contexto de levar PKC para RSSF este trabalho analisa a viabilidade de utilizar um novo criptossistemas de chaves públicas: MQQ, descrito no capítulo 3. MQQ foi proposto por Danilo Gligoroski, Smile Markovski e Svein Johan Knapskog (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008) e de acordo com seus autores possui velocidade de um típica cifra de bloco simétrica, sendo mais rápido que os mais populares esquemas de chaves públicas DH, RSA, ECC. As afirmações feitas em (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008) foram testadas e afirmadas na prova de conceito do MQQ, utilizando FPGA Xilinx (XILINX, 2010), existente no trabalho (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008). O trabalho feito em (GLIGOROSKI; MARKOVSKI; KNAPSKOG, 2008) realiza prova de conceito do MQQ em PC(processador PC Intel Core 2 Duo 64 bits) e em plataforma FPGA Xilinx. Posteriormente o trabalho (EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008) faz outra prova de conceito do MQQ na plataforma FPGA Xilinx (XILINX, 2010) que confirma o bom desempenho do MQQ.

Muitas características do MQQ, afirmadas nos trabalhos (GLIGOROSKI; MAR-

KOVSKI; KNAPSKOG, 2008; EL-HADEDY; GLIGOROSKI; KNAPSKOG, 2008), o tornam forte candidato a ser utilizado em plataformas de processamento limitada: não há expansão da mensagem, velocidade de típica cifra de bloco simétrica, mais veloz que tradicionais PKC RSA, DH ou ECC, necessita de chaves menores para manter nível de segurança do RSA, utilização freqüente das operações básicas de bits XOR e AND. A utilização das operações XOR e AND é um fato interessante, afinal o MQQ não necessita de cálculos pesados para obter nível de segurança de outros PKC, além do fato destas operações serem básicas até mesmo para plataformas com processamento limitado.

Considerando as aplicações de RSSF que necessitam do nível de segurança obtido por um criptossistema de chaves públicas e a limitação de recursos de processamento, energia e memória nas RSSF. Examinando todas as características do MQQ que o tornam um promissor candidato a ser utilizado em plataformas de processamento limitado, tal como RSSF. Este trabalho propõe uma nova abordagem (não há trabalhos até o momento relacionando MQQ e RSSF) para levar PKC para RSSF utilizando o algoritmo MQQ.

Apesar das características promissoras do MQQ para a plataforma de RSSF é imprescindível analisar a viabilidade do MQQ ser utilizado em plataformas de RSSF, o que é feito no capítulo 4. Nos experimentos feitos nas plataformas de RSSF TelosB e MICAz o MQQ exibiu bom desempenho, mesmo não havendo otimizações no algoritmo do MQQ para as plataformas TelosB e MICAz. A memória ocupada pelo procedimento de encriptação do MQQ ocupou 0,25% da RAM e 6,97% da ROM disponível no TelosB, a decriptação necessitou de 0,27% de RAM e 70,35% de ROM no TelosB. No TelosB o espaço ocupado em memória ainda pode ser reduzido, afinal os polinômios podem possuir termos repetidos. A memória ocupada pelo MQQ foi prejudicada pois alguma estruturas de dados estáticas ficaram na RAM ao invés da ROM, mas o objetivo do trabalho não é otimizar para uma plataforma específica de

RSSF. Com relação ao consumo de memória o MQQ traz grandes expectativas, afinal há espaço para pesquisas reduzam o tamanho das estruturas de dados do MQQ.

Considerando o tempo de execução o MQQ mostrou desempenho favorável, devendo-se em parte pelo algoritmo do MQQ necessitar de instruções básicas de XOR e AND de bits. Nos dois métodos de coleta do tempo o MQQ exibiu tempos de 825,1 e 843,5 milissegundos para a encriptação e tempos de 116,6 e 125 milissegundos para a decriptação.

As propriedades do MQQ e os resultados obtidos o tornam recomendável para aumentar o nível de segurança com melhor desempenho em redes de sensores sem fio e outras plataformas com limitação de recursos de memória, energia e processamento.

Durante a realização deste trabalho o artigo (MAIA; BARRETO; OLIVEIRA, 2010) foi aceito e está esperando publicação, neste artigo realiza-se prova de conceito do MQQ em uma plataforma de RSSF, analisando o tempo de execução e consumo de memória dos procedimentos de encriptação e decriptação do MQQ nas plataformas TelosB e MICAz.

5.1 Trabalhos Futuros

O MQQ exibiu desempenho favorável nas plataformas TelosB e MICAz, mas há aspectos do MQQ que ainda necessitam ser analisados em RSSF, tais como: algoritmo de geração de chaves do MQQ, consumo de energia dos algoritmos do MQQ, comparação do desempenho entre MQQ e ECC. Além de redes de sensores sem fio uma proposta para futuros trabalhos é analisar o desempenho do MQQ em outras plataformas com recursos limitados, tais como RFID.

Outra proposta de trabalho futuro consiste buscar mecanismos para reduzir o consumo de memória necessário para as estruturas de dados utilizadas pelo MQQ. Matrizes esparsas podem dar um direcionamento nas pesquisas para reduzir o tamanhos dos

quase-grupos e outras matrizes utilizadas pelo MQQ.

O MQQ possui a característica de ser altamente paralelizável, mas esta característica não fez parte do escopo estudado neste trabalho. Esta propriedade do MQQ torna possível desenvolver soluções criptográficas dedicadas que utilizem arquiteturas paralelas, tal como FPGA. Hardwares dedicados podem ser utilizados para gerar chaves criptográficas ou outras operações criptográficas que necessitem de alto desempenho.

Uma característica do MQQ não analisada neste trabalho é a propriedade do MQQ ser um algoritmo pós-quântico (AND; PAL, 2009; AHLAWAT; PAL, 2009). O desenvolvimento de computadores quânticos não está em futuro muito distante, onde estes computadores serão utilizados para resolver problemas práticos e utilizados para quebrar muitos algoritmos criptgráficos existentes. Ou seja problemas difíceis como problemas de fatoração, tal qual o problema do logaritmo discreto poderá ser resolvido em tempo polinomial em computadores quânticos (NIELSEN; CHUANG, 2003). É imprescindível buscar esquemas poderosos de difícil solução até mesmo para computadores quânticos, neste contexto o algoritmo MQQ promete ser um caminho para criar problemas de difícil solução para computadores quânticos (AND; PAL, 2009; AHLAWAT; PAL, 2009).

REFERÊNCIAS

AGILENT. *Agilent 34401A Multimeter*. 2007. http://cp.literature.agilent.com/litweb/pdf/5968-0162EN.pdf.

____. *E363xA Series Programmable DC Power Supplies*. 2009. http://cp.literature.agilent.com/litweb/pdf/5968-9726EN.pdf.

AHLAWAT, K. G. R.; PAL, S. K. From mq to mqq cryptography: Weaknesses new solutions. *Western European Workshop on Research in Cryptology*, 2009.

AND, K. G. R. A.; PAL, S. K. Fast generation ofmultivariate quadratic quasigroups for cryptographic applications. Farnborough, Hampshire, UK, 2009.

BOGDANOV THOMAS EISENBARTH, A. R. A.; WOLF, C. *Time-Area Optimized Public-Key Engines: MQ-Cryptosystems as Replacement for Elliptic Curves?* 2008. Cryptology ePrint Archive, Report 2008/349. http://eprint.iacr.org/.

CAYIRCI, E.; RONG, C. *Security in Wireless Ad Hoc and Sensor Networks*. [S.l.]: John Wiley & Sons Ltd, 2009.

CROSSBOW. *"MICAz Datasheet"*. 2008. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

____. *"TelosB Datasheet"*. 2008. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

EL-HADEDY, M.; GLIGOROSKI, D.; KNAPSKOG, S. J. High performance implementation of a public key block cipher - mqq, for fpga platforms. In: *RECONFIG '08: Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*. Washington, DC, USA: IEEE Computer Society, 2008. p. 427–432. ISBN 978-0-7695-3474-9.

GAUBATZ, G. et al. State of the art in ultra-low power public key cryptography for wireless sensor networks. In: *In 2nd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2005), Kauai Island*. [S.l.: s.n.], 2005. p. 146–150.

GLIGOROSKI, D.; MARKOVSKI, S.; KNAPSKOG, S. J. *Public Key Block Cipher Based on Multivariate Quadratic Quasigroups*. 2008. Cryptology ePrint Archive, Report 2008/320. http://eprint.iacr.org/.

GUIMARÃES, G. et al. Avaliação de mecanismos de segurança em uma plataforma para redes de sensores sem fio. In: *V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'05)*. Florianopolis, Brazil: [s.n.], 2005.

GURA, N. et al. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: . [S.l.: s.n.], 2004. p. 119–132.

INSTRUMENTS, N. *LabView*. 2009. http://www.ni.com/labview/.

KOBLITZ, N. Elliptic curve cryptosystems. *Mathematics of computation*, v. 48, p. 203–209, 1987.

LEVIS, P. *TinyOS Programming*. 2006. TinyOS Programming. http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf.

LEVIS, P.; LEE, N. *TOSSIM: A Simulator for TinyOS Networks*. 2003. TOSSIM: A Simulator for TinyOS Networks. http://www.eecs.berkeley.edu/~pal/pubs/nido.pdf.

LIU, D.; NING, P. *Security for Wireless Sensor Networks*. [S.l.]: Springer, 2007.

LOPEZ, J.; ZHOU, J. *Wireless Sensor Network Security*. [S.l.]: IOS Press, 2008.

LOUREIRO, A. A. et al. Redes de sensores sem fio. Natal, 2003.

MAIA, R. J. M.; BARRETO, P. S. L. M.; OLIVEIRA, B. T. de. Implementation of multivariate quadratic quasigroup for wireless sensor network. *Security in Computing - Springer Transactions on Computational Science, Springer Verlag*, 2010. Artigo aguardando aceite. Divulgação do aceite para agosto/2010.

MARGI, C. B. et al. Segurança em redes de sensores sem fio. In: (SBC), S. B. de C. (Ed.). *SBSEG 2009 / IX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. Campinas, 2009. p. 284.

____. Impact of operating systems on wireless sensor networks (security) applications and testbeds. Zurich, Switzerland, 2010.

MENEZES, A. J.; OORSCHOT, P. C. van; A.VANSTONE, S. *Handbook of Applied Cryptography*. Boca Raton, USA: CRC Press, 1999.

NIELSEN, M. A.; CHUANG, I. L. *Quantum Computation And Quantum Information*. [S.l.]: The Press Syndicate Of The University Of Cambridge, 2003.

OLIVEIRA, L. B. et al. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. In: *5th International Conference on Networked Sensing Systems (INSS'08)*. Kanazawa/Japan: [s.n.], 2008. To appear.

PAL, S. K.; SUMITRA. Development of efficient algorithms for quasigroup generation encryption. In: *Proc. IEEE International Advance Computing Conference IACC 2009*. [S.l.: s.n.], 2009. p. 940–945.

PALAFOX, L. E.; GARCIA-MACIAS, J. A. Handbook of research on wireless security. In: ____. [S.l.: s.n.], 2008. cap. XXXIV - Security in Wireless Sensor Networks.

RIVEST, R.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, p. 120–126, 1978.

SABBAH ERIC; KANG, K.-D. Guide to wireless sensor networks. In: ____. [S.l.]: Springer, 2009. cap. Security in Wireless Sensor Networks.

STALLINGS, W. *Criptografia e Segurança de Redes*. 4. ed. SÃO PAULO: Pearson Prentice Hall, 2008.

SZCZECHOWIAK, P. et al. Nanoecc: Testing the limits of elliptic curve cryptography in sensor networks. In: *EWSN*. [S.l.: s.n.], 2008. p. 305–320.

TERADA, R. *Segurança de Dados : Criptografia em Redes de Computador*. 2. ed. São Paulo: Blucher, 2008.

TINYOSFORUM. *TinyOS Community Forum || An open-source OS for the networked sensor regime*. 2009. `http://www.tinyos.net/`.

WANG, Q.; ZHANG, T. Security in rfid and sensor networks. In: ____. [S.l.]: Taylor and Francis Group, 2009. cap. A Survey on Security in Wireless Sensor Networks.

XILINX. *FPGA and CPLD Solutions from Xilinx, Inc.* 2010. `http://www.xilinx.com/`.

XUBUNTOS. *XubunTOS*. 2004. `http://toilers.mines.edu/Public/XubunTOS`.

APÊNDICE A – IMPLEMENTAÇÃO MQQ

Neste apêndice estão os fontes utilizados nos experimentos com MQQ, a implementação foi feita em NesC para TinyOS 2.

No código fonte A.1 há o arquivo Makefile utilizado para gerar os binários do MQQ. Para compilar pode ser utilizado o comando *make micaz* ou *make telosb*, onde *telosb* ou *micaz* indicam para qual plataforma os fontes serão compilados.

Para simular com TOSSIM é necessário adicionar a opção *sim*, ficando o comando *make micaz sim*.

Código Fonte A.1: Makefile

```
1 COMPONENT=MqqAppC
2 CFLAGS += -I$(TOSDIR)/lib/printf
3 include $(MAKERULES)
```

No código fonte A.2 o componente MqqAppC provê configuração e conecta os componentes MainC, PrintfC e TimerMilliC no componente MqqC.

Código Fonte A.2: MqqAppC.nc

```
1 #include "printf.h"
2 #include "Timer.h"
3
4 configuration MqqAppC { }
5
6 implementation {
7   components MainC, MqqC;
```

```
8     components PrintfC;
9     components new TimerMilliC() as Timer0;
10
11    MqqC.Timer0 -> Timer0;
12    MqqC -> MainC.Boot;
13    MqqC.PrintfControl -> PrintfC;
14    MqqC.PrintfFlush -> PrintfC;
15  }
```

No código fonte A.3 o componente MqqC é o módulo que implementa interfaces utilizando eventos. O evento Boot.booted faz chamada a implementação do Mqq. Para facilitar a obtenção do tempo será utilizada interface Timer<TMilli> para obter o tempo em milissegundos. Este módulo pode ser utilizado para chamar tanto os procedimentos encriptar() quanto decriptar().

Código Fonte A.3: MqqC.nc

```
1  #include "Mqq.c"
2  #include "printf.h"
3  #include "Timer.h"
4
5  module MqqC {
6    uses {
7      interface Boot;
8      interface SplitControl as PrintfControl;
9      interface PrintfFlush;
10     interface Timer<TMilli> as Timer0;
11   }
12 }
13
14 implementation {
15   event void Boot.booted() {
16     call PrintfControl.start();
17   }
```

```
18
19    event void PrintfControl.startDone(error_t error) {
20        int t1, t2;
21        t1 = call Timer0.getNow(); // tempo inicial
22        encriptar();
23        t2 = call Timer0.getNow(); // tempo final
24        printf("inicial:%i final:%i diferenca:%i\n", t1, t2, t2-t1);
25        call PrintfFlush.flush();
26    }
27    event void Timer0.fired() {  }
28    event void PrintfControl.stopDone(error_t error) {  }
29    event void PrintfFlush.flushDone(error_t error) {  }
30  }
```

No script A.4 há a chamada utilizada para rastrear o funcionamento do MQQ utilizando TOSSIM. Para debugar os procedimentos de encriptar e decriptar do MQQ foi utilizada a a variável DBG. Exemplo de linha de comando com DBG: *dbg("Mqq", "Mensagem %d", variável);*.

Código Fonte A.4: tossim.py

```
1  #! /usr/bin/python
2
3  from TOSSIM import *
4  import sys
5
6  t = Tossim([])
7  f = open("log.txt", "w")
8  t.addChannel("Mqq", f)
9
10 t.getNode(0).bootAtTime(1000);
11
12 for i in range(0, 100) :
13         t.runNextEvent();
```

A.1 Cifragem com MQQ

No código fonte A.5 encontra-se a implementação do procedimento encriptar() do MQQ, além das estruturas de dados necessárias. Nos experimentos os valores da chaves estão fixos nos fontes, estes valores de chaves não foram incluídos no código fonte A.5.

Código Fonte A.5: Encriptação Mqq.c

```
1  #define tam_chave 160
2
3  /* valor a ser encriptado 160 bits
4   Entre {} deverá ser colocado o valor a ser encriptado */
5  const static uint16_t X[10]={};
6
7  // saida encriptada 160 bits
8  uint16_t Y[10];
9
10 // chave publica. Valor da chave pública deverá estar entre {}
11 const static uint16_t CP[806]={};
12
13 /* Auxilia a chave publica informa se o termo existe ou nao em
       varios polinomios. Estrutura criada para suportar a abordagem que
        considera que um termo do poliômio pode estar repetido */
14 const static uint16_t CP_aux[806]={};
15
16 struct termo_polinomios {
17   // informa em qual dos 160 polinomios o termo esta presente
18   uint16_t polinomios[10];
19
20   struct termo_polinomios *prox;
21 } termo_polinomios;
22
23 struct termo_polinomios *TP;
```

```
24
25 void encriptar() {
26     uint16_t desloc=15,i,k=12881, j=0, x1=0, x2=0, comeco_x2=0, m;
27     uint8_t  multiplicacao;
28
29     for(i=0; i<k; i++) {
30         if(x1!=0 && x2==0) multiplicacao=x1;
31         if(x1==0 && x2!=0) multiplicacao=x2;
32         if(x1==0 && x2==0) multiplicacao=1;
33
34         // multiplicacao entre dois termos
35         multiplicacao =
36                     ( ( X[ x1 >> 4 ] >> (((x1 >> 4) + 1)*16 - x1 )
                            ) & 1 &
37                     ( ( X[ x2 >> 4 ] >> (((x2 >> 4) + 1)*16 - x2 )
                            ) & 1);
38
39         // O termo existe em alguns polionomios.
40         // Bloco verifica quais dos polinômios encontra-se o termo
41         if(!(CP_aux[j]>>desloc) && TP!=NULL) {
42             for(m=0; m<10; m++) {
43                 if(i!=0)
44                     // XOR entre dois termos
45                     Y[i]=Y[i] ^ (Y[i] & multiplicacao);
46                 else
47                     Y[i]=Y[i] & multiplicacao;
48             }
49
50             TP= TP->prox;
51         }
52
53         // O termo encontra-se em todos polinômios
54         if(CP_aux[j]>>desloc) {
```

```
55              for(m=0;m<10;m++)
56                  if(i!=0)
57                      Y[i]=Y[i] ^ (0xFFFF & multiplicacao);
58                  else
59                      Y[i]=0xFFFF & multiplicacao;
60          }
61
62          if(desloc==0) {
63              desloc=16;
64              j++;
65          }
66
67          if(comeco_x2==1 && x1==tam_chave) comeco_x2++;
68
69          if(x1==tam_chave || x2==tam_chave) {
70            x1=0;
71            x2=comeco_x2+1;
72            comeco_x2++;
73          }
74
75          x1++;
76
77          if(comeco_x2>=1) x2++;
78
79          desloc--;
80      }
81  }
```

A.2 Decifragem com MQQ

No código fonte A.6 encontra-se a implementação do procedimento decriptar() do MQQ, além das estruturas de dados necessárias. Nos experimentos os valores da

chaves estão fixos nos fontes, estes valores de chaves não foram incluídos no código fonte A.6.

Código Fonte A.6: Decriptação Mqq.c

```
#define tam_chave 160

// Entrada Encriptada
// Entre {} deverá ser colocado a entrada a ser encriptada
static const uint16_t X[]={};

uint16_t Y[10], //Y é a Saida Decriptada
         Register_X[10];

//Estrutura para Decriptar

uint16_t
         W, // W possui 13 Bits
         Z; // Z possui 13 Bits

// Entre {} deverá ser colocado os 8 quase-grupos
const static uint8_t QUASEGRUPOS[8][1024] = {};

const static uint16_t
   // Entre {} deverá ser os valores da Matriz Inversa de T
   Inversa_T[tam_chave][10]={},

   // Entre {} deverá ser os valores da Matriz Inversa de S
   Inversa_S[tam_chave][10] = {},

   // Entre {} deverá ser os valores da Inversa de Dobbertin
   INVERSA_DOBBERTIN[8192] = {};

 uint16_t calculasoma(uint16_t mult) {
      return
```

```
31              ((mult>>15)&1) ^ ((mult>>14)&1) ^ ((mult>>13)&1) ^ \
32              ((mult>>12)&1) ^ ((mult>>11)&1) ^ ((mult>>10)&1) ^ \
33              ((mult>>9)&1)  ^ ((mult>>8)&1)  ^ ((mult>>7)&1) ^ \
34              ((mult>>6)&1)  ^ ((mult>>5)&1)  ^ ((mult>>4)&1) ^ \
35              ((mult>>3)&1)  ^ ((mult>>2)&1) ^ ((mult>>1)&1) ^ ((mult
                    >>0)&1);
36  }
37
38  void Private_Matrix_Inverse_T() {
39    int i=0, soma, j=0, posregx=0;
40
41    for(i=0; i<tam_chave; i++) {
42      //Multiplicando o vetor de entrada X com cada linha da Matriz
            Inversa T
43      soma=calculasoma(Inversa_T[i][0] & X[0]);
44      soma=soma ^ calculasoma(Inversa_T[i][1] & X[1]);
45      soma=soma ^ calculasoma(Inversa_T[i][2] & X[2]);
46      soma=soma ^ calculasoma(Inversa_T[i][3] & X[3]);
47      soma=soma ^ calculasoma(Inversa_T[i][4] & X[4]);
48      soma=soma ^ calculasoma(Inversa_T[i][5] & X[5]);
49      soma=soma ^ calculasoma(Inversa_T[i][6] & X[6]);
50      soma=soma ^ calculasoma(Inversa_T[i][7] & X[7]);
51      soma=soma ^ calculasoma(Inversa_T[i][8] & X[8]);
52      soma=soma ^ calculasoma(Inversa_T[i][9] & X[9]);
53
54      //Cada um dos 160 bits de Register_X recebera o resultado dos
            ANDs e XORs
55      //resultantes da multiplicacao de cada linha da inversa de T
            por X
56      if(j<=15)
57        Register_X[posregx]=Register_X[posregx] | (soma<<j);
58      else  {
59        posregx++;
```

```
60          j=0;
61        }
62
63        j++;
64      }
65    }
66
67    void Private_Matrix_Inverse_S() {
68      int i=0, soma, j=0, posregx=0;
69
70      for(i=0; i<tam_chave; i++) {
71        //Multiplicando o vetor de entrada X com cada linha da Matriz
              Inversa S
72        soma=calculasoma(Inversa_S[i][0] & X[0]);
73        soma=soma ^ calculasoma(Inversa_S[i][1] & X[1]);
74        soma=soma ^ calculasoma(Inversa_S[i][2] & X[2]);
75        soma=soma ^ calculasoma(Inversa_S[i][3] & X[3]);
76        soma=soma ^ calculasoma(Inversa_S[i][4] & X[4]);
77        soma=soma ^ calculasoma(Inversa_S[i][5] & X[5]);
78        soma=soma ^ calculasoma(Inversa_S[i][6] & X[6]);
79        soma=soma ^ calculasoma(Inversa_S[i][7] & X[7]);
80        soma=soma ^ calculasoma(Inversa_S[i][8] & X[8]);
81        soma=soma ^ calculasoma(Inversa_S[i][9] & X[9]);
82
83        //Cada um dos 160 bits de Register_X recebera o resultado dos
              ANDs e XORs
84        //resultantes da multiplicacao de cada linha da inversa de T
              por X
85        if(j<=15)
86          Register_X[posregx]=Register_X[posregx] | (soma<<j);
87        else  {
88          posregx++;
89          j=0;
```

```
90          }
91
92          j++;
93        }
94    }
95
96    void Inversa_Dobbertin() {
97     // passo 2
98     // W = (y_1, y_2, y_3, y_4, y_5, y_6, y_11, y_16, y_21, y_26, y_31, y_36, y_41
          )
99     // Register_X eh a saida do modulo  Private_Matrix_Inverse_T
100    W=((Register_X[0]>>0)&1)<<0; // W1 = bit 1 de Register_X
101    W=W|((Register_X[0]>>1)&1)<<1; // W2 = bit 2 de Register_X
102    W=W|((Register_X[0]>>2)&1)<<2; // W3 = bit 3 de Register_X
103    W=W|((Register_X[0]>>3)&1)<<3; // W4 = bit 4 de Register_X
104    W=W|((Register_X[0]>>4)&1)<<4; // W5 = bit 5 de Register_X
105    W=W|((Register_X[0]>>5)&1)<<5; // W6 = bit 6 de Register_X
106    W=W|((Register_X[0]>>10)&1)<<6; // W7 = bit 11 de Register_X
107    W=W|((Register_X[0]>>15)&1)<<7; // W8 = bit 16 de Register_X
108    W=W|((Register_X[1]>>4)&1)<<8; // W9 = bit 21 de Register_X
109    W=W|((Register_X[1]>>9)&1)<<9; // W10 = bit 26 de Register_X
110    W=W|((Register_X[1]>>14)&1)<<10; // W11 = bit 31 de Register_X
111    W=W|((Register_X[2]>>3)&1)<<11; // W12 = bit 36 de Register_X
112    W=W|((Register_X[2]>>8)&1)<<12; // W13 = bit 41 de Register_X
113
114    // passo 3
115    // Z=Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8, Z9, Z10, Z11, Z12, Z13=INVDOB(W)
116    Z=INVERSA_DOBBERTIN[W];
117    Register_X[0]=0;
118    Register_X[1]=0;
119    Register_X[2]=0;
120
121    // passo 4
```

```
122   // y_1=Z1, y_2=Z2, y_3=Z3, y_4=Z4, y_5=Z5, y_6=Z6, y_11=Z11, y_16=Z16,
123   // y_21=Z21, y_26=Z26, y_31=Z31, y_36=Z36, y_41=Z41
124
125   W=((Z>>0)&1)<<0; // bit 1 de Register_X = bit 1 de Z
126   W=W|((Z>>1)&1)<<1; // bit 2 de Register_X = bit 2 de Z
127   W=W|((Z>>2)&1)<<2; // bit 3 de Register_X = bit 3 de Z
128   W=W|((Z>>3)&1)<<3; // bit 4 de Register_X = bit 4 de Z
129   W=W|((Z>>4)&1)<<4; // bit 5 de Register_X = bit 5 de Z
130   W=W|((Z>>5)&1)<<5; // bit 6 de Register_X = bit 6 de Z
131   W=W|((Z>>6)&1)<<10; // bit 11 de Register_X = bit 7 de Z
132   W=W|((Z>>7)&1)<<15; // bit 16 de Register_X = bit 8 de Z
133    Register_X[0]=W;
134
135
136   W=((Z>>8)&1)<<4; // bit 21 de Register_X = bit 9 de Z
137   W=W|((Z>>9)&1)<<9; // bit 26 de Register_X = bit 10 de Z
138   W=W|((Z>>10)&1)<<14; // bit 31 de Register_X = bit 11 de Z
139    Register_X[1]=W;
140
141   W=((Z>>11)&1)<<3; // bit 36 de Register_X = bit 12 de Z
142   W=W|((Z>>12)&1)<<8; // bit 41 de Register_X = bit 13 de Z
143    Register_X[2]=W;
144  }
145
146  void Sequenciador() {
147   uint8_t i, desloc=0, yy[32]; // 160/5 bits
148   //passo5
149   // yy = (Y1 ... Y32), onde Yi possui 32 bits
150
151   //passo 6
152   // k-1 operacoes de lookup nas parastrophes quase-grupos
153
154   yy[0] = Register_X[desloc/16] << ((desloc++)%16) |
```

```
155             Register_X[desloc/16] << ((desloc++)%16) |
156             Register_X[desloc/16] << ((desloc++)%16) |
157             Register_X[desloc/16] << ((desloc++)%16) |
158             Register_X[desloc/16] << ((desloc++)%16);
159
160     for(i=1; i<32; i++) {
161        yy[i] = Register_X[desloc/16] << ((desloc++)%16) |
162                Register_X[desloc/16] << ((desloc++)%16) |
163                Register_X[desloc/16] << ((desloc++)%16) |
164                Register_X[desloc/16] << ((desloc++)%16) |
165                Register_X[desloc/16] << ((desloc++)%16);
166
167        if(i==1 || i==2)
168           yy[i] = QUASEGRUPOS [i] [ yy[i-1]<<5 | yy[i]];
169        else
170           yy[i] = QUASEGRUPOS [3+((i+2)%6)] [ yy[i-1]<<5 | yy[i]];
171     }
172
173     Register_X[0]=yy[0 ]      | yy[1 ]<<5  | yy[2 ]<<10 | (yy[3 ]&1 )
           <<15;  // 16 bits
174     Register_X[1]=yy[3 ]>>1   | yy[4 ]<<4  | yy[5 ]<<9  | (yy[6 ]&3 )
           <<14;  // 16 bits
175     Register_X[2]=yy[6 ]>>2   | yy[7 ]<<3  | yy[8 ]<<8  | (yy[7 ]&7 )
           <<13;  // 16 bits
176     Register_X[3]=yy[7 ]>>3   | yy[9 ]<<2  | yy[10]<<7  | (yy[11]&15)
           <<12;  // 16 bits
177     Register_X[4]=yy[11]>>4   | yy[12]<<1  | yy[13]<<6 |  yy[14]
           <<15;  // 16 bits
178     Register_X[5]=yy[16]      | yy[17]<<5  | yy[18]<<10 | (yy[19]&1 )
           <<15;  // 16 bits
179     Register_X[6]=yy[19]>>1   | yy[20]<<4  | yy[21]<<9  | (yy[22]&3 )
           <<14;  // 16 bits
180     Register_X[7]=yy[22]>>2   | yy[23]<<3  | yy[24]<<8  | (yy[25]&7 )
```

```
    <<13;  // 16 bits
181  Register_X[8]=yy[25]>>3  | yy[26]<<2  | yy[27]<<7  | (yy[28]&15)
    <<12;  // 16 bits
182  Register_X[9]=yy[28]>>4  | yy[29]<<1  | yy[30]<<6  | yy[31];
183 }
184
185 void decriptar() {
186  Private_Matrix_Inverse_T(); // passo 1  // y_ = Inversa_T * X //
    y_=Inversa_T(X)
187  Inversa_Dobbertin();        // passo 2 a 4
188  Sequenciador();             // passo 5 e 6
189  Private_Matrix_Inverse_S(); // passo 7 // Y = Inversa_S(x_) // Y =
     Inversa_S(x_)
190 }
```

GLOSSÁRIO

Cifra de Bloco Esquema para cifrar e decifrar em que um bloco de texto claro é tratado como um todo e usado para produzir um bloco de texto cifrado de mesmo tamanho., 25

Criptossistemas de chaves públicas Utilizam duas chaves, uma chave pública e uma chave privada. Também conhecido como Criptossistemas assimétricos., 24

Criptossistemas simétricos Utiliza a mesma chave tanto para cifrar quanto decifrar uma mensagem., 24

ECC Criptografia de Curvas Elípticas (Elliptic Curve Cryptography - ECC) foram sugeridas independentemente por Vic Miller e Neal Koblitz, em 1985. Sua segurança é baseada na dificuldade computacional de se resolver o problema do logaritmo discreto sobre curvas elípticas. ECC obtém mesmo nível de segurança que o RSA utilizando tamanho de chaves muito menor., 24

NesC Linguagem de programaçao utilizada no TinyOs, sendo uma Linguagem C estilizada., 36

Problema do Logaritmo Discreto Dados um primo p e inteiros $g, t : 0 < g, t < p$, calcular um inteiro s tal que $t = g^s mod p$. Por exemplo $p = 17, g = 7$ e $t = 10$, calcular um s tal que $10 = 7^s mod 17$. A resposta s=9. Para p relativamente longo não há um algoritmo eficiente, de tempo polinomial., 24

RFID RFID Radio-Frequency IDentification é um método de identificação automática através de sinais de rádio, recuperando e armazenando dados remotamente através de dispositivos chamados de tags RFID., 55

RSA Algoritmo publicado em 1978 tem seu nome derivado das iniciais dos autores: Ron Rivest, Adi Shamir e Len Adleman. Algoritmo baseado na dificuldade computacional de fatorar um número em inteiros primos. Quanto maior o número maior a dificuldade em encontrar os fatores primos., 24

TinyOS Sistema operacional open-source concebido para redes de sensores sem fio (TinyOS Community Forum)., 36

TOSSIM Simulador de eventos discretos para redes de sensores que utilizem TinyOS., 36

XubunTOS	Lançado em julho de 2007. Ele simplifica a instalação do TinyOS usando um live CD do Linux e consiste na agregação do sistema Xubuntu Debian e do TinyOS 2.x, incluindo todos seus pacotes e repositórios das versões anteriores (XubunTOS)., 37