UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO TÉCNICO INDUSTRIAL DE SANTA MARIA
CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

UM MIDDLEWARE PARA PROVER COMUNICAÇÃO SEGURA ENTRE DISPOSITIVOS

TRABALHO DE CONCLUSAO DE CURSO

Bolívar Menezes da Silva

Santa Maria, RS, Brasil
2015

STRC/UFSM, RS SILVA, Bolívar Menezes Tecnólogo 2015

UM MIDDLEWARE PARA PROVER COMUNICAÇÃO SEGURA ENTRE DISPOSITIVOS

Bolívar Menezes da Silva

Trabalho apresentado ao Curso de Graduação em Tecnologia em Redes de Computadores, Área de concentração em Segurança da Informação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Redes de Computadores.**

Orientador: Prof. Me. Tiago Antônio Rizzetti

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Colégio Técnico Industrial de Santa Maria
Curso Superior de Tecnologia em Redes de Computadores

A Comissão Examinadora, abaixo assinada,
aprova a Monografia

UM MIDDLEWARE PARA PROVER COMUNICAÇÃO SEGURA ENTRE DISPOSITIVOS

elaborada por
Bolívar Menezes da Silva

como requisito parcial para obtenção do grau de
Tecnólogo em Redes de Computadores

Comissão Examinadora

Tiago Antônio Rizzetti, Me.
(Presidente/Orientador)

Alfredo Del Fabro Neto, Tecg. (UFSM)

Renato Preigschadt de Azevedo, Me. (UFSM)

Santa Maria, 25 de junho de 2015

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família, principalmente aos meus pais, Paulo e Neíta, aos meus irmãos, Daiane e Junior, cujo o apoio e conselhos, tornaram possíveis esta conquista.

Ao meu orientador Tiago Antônio Rizzetti, pela amizade, pelos ensinamentos e pela disponibilidade de tempo. Certamente um grande exemplo de professor.

Aos demais professores, que fizeram parte desta jornada, os quais passei a considerar como amigos. Com toda certeza, tiveram uma grande participação em meu crescimento pessoal.

Por último, mas não menos importante, aos meus amigos e colegas. Esses, que fizeram uma grande contribuição ao meu processo de aprendizagem. E o mais importante, contribuíram de forma significativa, em meu crescimento pessoal. Um agradecimento especial para os amigos: Alexandre Rodrigues, Bruno Rizzetti, Celso Brossard, Clayton Alves, Glauco Rodrigues, Murilo Cervi, Pedro Wessel, Renato Azevedo, Tiago Rizzetti, William Floriano.

Meu muito obrigado e um grande abraço a todos.

Seja a mudança que você
deseja ver no mundo.

(Mahatma Gandhi)

RESUMO

Monografia
Curso Superior de Tecnologia em Redes de Computadores
Universidade Federal de Santa Maria

UM MIDDLEWARE PARA PROVER COMUNICAÇÃO SEGURA ENTRE DISPOSITIVOS

AUTOR: BOLÍVAR MENEZES DA SILVA
ORIENTADOR: TIAGO ANTÔNIO RIZZETTI
Data e Local da Defesa: Santa Maria, 25 de junho de 2015

A variedade de aplicações que utilizam redes de dados para automatizar tarefas já existentes, ou mesmo possibilitar novos serviços, é cada vez maior. O conceito de *Smart Grids*, por exemplo, está diretamente relacionado a esta questão, uma vez que depende intensivamente de uma rede de comunicação bidirecional e confiável para monitorar e gerenciar o sistema de distribuição de energia elétrica em tempo real. Porém para que aplicações como estas se tornem possíveis, necessita-se de uma rede de comunicação que possa prover uma elevada confiabilidade na comunicação. Desta forma, é necessário garantir a integridade, confidencialidade e autenticidade dos dados transmitidos. Esses requisitos poderão ser atingidos através da utilização de técnicas de criptografia. Portanto, este trabalho propõe uma arquitetura para comunicação onde utiliza-se chaves criptográficas gerenciadas por uma entidade certificadora local, que visa prover segurança, abstraindo a necessidade das aplicações tratarem diretamente desta questão. Para isso projetou uma API de forma a fornecer as principais funcionalidades de segurança necessárias as aplicações de sistemas críticos. Para avaliar a solução proposta, implementou-se o middleware SECOM (*Secure Communications Middleware*) de forma a implementar as principais funções previstas nesta arquitetura. Os testes realizados demonstram sua viabilidade o que a torna uma solução a ser avaliada para aplicações de sistemas críticos.

Palavras-chave: Segurança em Redes. Middleware de Segurança. Segurança. Criptografia. SECOM.

ABSTRACT

Monography
Superior Course of Technology in Computer Networks
Federal University of Santa Maria

A middleware for providing safe communication between devices

AUTHOR: BOLÍVAR MENEZES DA SILVA
ADVISER: TIAGO ANTÔNIO RIZZETTI
Defense Place and Date: Santa Maria, 25th June 2015

The applications using data networks to automate existing tasks or even allow for new services have become growing. The concept of Smart Grids, for example, depends intensively of reliable bidirectional communications network to monitor and manage the electricity distribution system in real time. However, to become possible achieve the requirements for this kind of applications it necessary provide a communication network that could provide high reliability in communications. In that way, it is necessary to ensure the integrity, confidentiality and authenticity of transmitted data. These requirements can be achieved through the use of encryption techniques. Therefore, this work proposes a communication architecture which uses cryptographic keys managed by a local certifying unity. It aims to ensure security by abstracting all responsibility of the applications treat this question. In these case, was designed an API in order to provide key security features for the critical systems applications. To solve this question, was implemented SECOM (Secure Communications Middleware) middleware in order to implement the main functions of this architecture. The tests demonstrate viability of this system, making it a solution to be evaluated for critical system applications.

Key words: Networks Security. Security Middleware. Security. Encryption. SECOM.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

SECOM	*Security Communication Middleware*
WSN	*Wireless Sensor Network*
API	*Application programming Interface*
TSL	*Transport Security Layer*
SSL	*Socket Security Layer*
MAC	*Message Authentication Code*
IDE	*Integrated Development Environment*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento da tecnologia, os preços acessíveis e a necessidade de manter uma empresa competitiva em um mercado cada vez mais tecnológico fez com que o uso de redes de dados aumentasse exponencialmente nos últimos anos. Dessa forma, a preocupação dos pesquisadores com a segurança da informação teve de ser redobrada, tornando esse, um dos assuntos mais discutidos atualmente.

Os riscos de segurança na comunicação em redes de computadores surgiram no momento em que as primeiras redes foram implementadas. Algumas atividades popularizadas na Internet, como: redes sociais, operações bancárias, compras e armazenamento de dados em nuvem, são apenas alguns exemplos de aplicações utilizadas através da internet. Dessa forma, fica evidenciado o quão critico é a segurança para Internet nos dias atuais.

Uma aplicação bastante promissora para o uso de redes de dados são as chamadas *Smart Grids* ou Redes Elétricas Inteligentes. Estas se caracterizam pelo uso intensivo de equipamentos digitais e de telecomunicação, aliados a dispositivos sensores remotos que fornecem informações da rede em tempo real, aumentando a qualidade e a quantidade de informações recebidas. As informações coletadas podem ser desde status da rede de distribuição de energia e dispositivos conectados a ela, até informações do próprio dispositivo como: temperatura, tensão, integridade, sobrecarga de dados, pressão, som, luminosidade, entre outras.

As informações capturadas pelos dispositivos sensores normalmente são enviadas por redes sem fio em malha auto-configuráveis, conhecidas como redes *Mesh*, e que nesse contexto podem ser referidas como *WSN* ou Rede de Sensores sem fio. Dito isso, é importante salientar a criticidade da entrega segura deste tipo de informação. Algumas informações, como por exemplo, a tensão elétrica de um dispositivo, em alguns casos podem ser críticas. Visto que a alteração dessa informação de forma inapropriada por algum agente externo a rede, seja ele um agente malicioso ou mesmo por um defeito ocorrido na transmissão dos dados, pode ser mais prejudicial ao sistema, do que a perda total dessa informação. Se uma informação, como a da tensão elétrica em um dispositivo da rede for modificada, passando a informar que o dispositivo está trabalhando sob uma tensão mais baixa do que seria a ideal, por exemplo. Essa informação

pode ser enviada a um dispositivo atuador responsável pela regulagem da tensão. É possível que esse atuador aumente a tensão do dispositivo, buscando o que seria a tensão ideal, ocasionando na queima e/ou indisponibilidade do equipamento. Dependendo da criticidade do mesmo, é possível que haja uma indisponibilidade parcial ou mesmo total ao Sistema Elétrico de Potência (SEP).

Além da integridade das mensagens, outros desafios de segurança que são bastante comuns em redes de computadores, como ataques de negação de serviço, ou acesso externo não autorizado a dispositivos internos a rede, são mais difíceis de serem prevenidos ou detectados em redes *Mesh*. O fato desse tipo de rede ser caracterizado por não possuir uma infraestrutura fixa, com dispositivo central para verificação da troca dos pacotes entre rede interna e a externa, faz com que todos os dispositivos necessitem de uma proteção individual extra. Além disso, a troca de pacotes é realizada diretamente entre nós ou os pacotes são repassados pelos próprios nós até chegarem ao destino. Tal característica torna a rede bastante tolerante a falhas, já que se um dos nós falhar, a comunicação pode continuar sendo realizada por outro caminho. Porém esse fato também torna a rede sujeita a ataques de inserção de informações falsas, através de nós mal-intencionados inseridos inadequadamente na rede a fim de tornar indisponível um serviço ou a própria rede.

Visando uma forma de tornar as redes de dados mais seguras, esse trabalho tem como objetivo principal propor e desenvolver uma ferramenta que execute independente das aplicações que a utilizarão e que dê suporte a troca segura de dados entre os dispositivos. Para isso, será necessário realizar um estudo sobre as principais vulnerabilidades de uma rede de comunicação, verificando as técnicas disponíveis para minimizar os problemas de segurança comuns em redes de dados. Após a apresentação da aplicação proposta, serão demonstradas algumas das etapas do desenvolvimento da mesma, bem como os testes e resultados obtidos.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma breve descrição sobre algumas técnicas e protocolos citados e utilizados para o desenvolvimento deste trabalho.

2.1 Socket Security Layer – SSL

Segundo a RFC 6101 (FREIER et al., 2011), o protocolo SSL objetiva proporcionar privacidade e confiabilidade na comunicação entre duas aplicações.

O protocolo é dividido em duas camadas. A mais inferior é a camada do protocolo de registro, responsável pelo encapsulamento dos protocolos de nível superior. A segunda camada é a denominada de protocolo de *handshake* (“aperto de mãos”), responsável por prover uma forma de autenticação entre o cliente e o servidor.

O protocolo SSL estabelece que:

- Após o *handshake* toda a comunicação é criptografada com chave simétrica;
- A chave simétrica utilizada na comunicação é definida no *handshake*;
- Os pacotes têm sua integridade verificada a partir de um código de autenticação de mensagens (MAC). Além disso, funções de *hash* segura como MD5 e SHA são utilizadas em conjunto com a função MAC.

O SSL pode ser utilizado para diversos fins, porém é amplamente utilizado e popularizado devido ao seu uso junto a *browsers*, buscando prover segurança da navegação na Internet. A utilização do protocolo SSL não é uma regra para navegação, porém uma grande parcela dos sites atualmente utiliza-o. Principalmente em aplicações como: acesso a e-mails, redes sociais, acesso a aplicações bancárias e etc (DAHAB, 2003).

Sempre que um usuário desejar realizar uma conexão protegida pelo protocolo SSL, é necessário assegurar que tanto o *browser* do cliente, quanto o do servidor, suportam o uso deste protocolo. Após isso, é necessário ainda, que ambos entrem em consenso sobre qual será o tipo de criptografia e o tipo de compactação dos dados a ser utilizada na sessão. Esse acordo é estabelecido no processo de *handshake*. Dessa forma é necessário que haja pelo menos um algoritmo de criptografia e um método compactação de dados em comum, entre as partes. Além disso, no *handshake* também será estabelecida o ID da sessão, que será válido apenas até o fim da sessão corrente (DAHAB, 2003).

As comunicações utilizando protocolo SSL exigem que toda a troca de informação seja criptografada. Por questões de desempenho, a comunicação não é criptografada sempre utilizando algoritmos assimétricos. Dessa forma, após o cliente e o servidor terem realizado o *handshake* com sucesso, e ambos já possuírem a chave simétrica da sessão, a comunicação passa a ser realizada com o algoritmo de criptografia simétrica definido (DAHAB, 2003).

2.2 Transport Layer Security – TLS

Segundo a RFC 5246 (DIERKS e RESCORLA, 2008), o protocolo TLS, tem como principais objetivos garantir a privacidade e integridade dos dados. Para tal, o protocolo tem como prioridades principais, necessariamente nessa ordem: segurança criptográfica, interoperabilidade, extensibilidade, e mais eficiência nas operações de criptografia.

Note que o protocolo TLS estabelece como base metas bastante semelhantes com o próprio protocolo SSL. O motivo é explicado pelo fato de que a versão do protocolo TLS 1.0 é baseada na versão 3.0 do protocolo SSL. Quanto à segurança criptográfica, pode se dizer que é também a mesma base entre os dois protocolos, ou seja, é utilizado criptografia assimétrica em conjunto com a simétrica em ambos os casos. A principal diferença fica na capacidade do protocolo TLS utilizar novos métodos criptográfico como extensão, sempre que desejar.

Do ponto de vista de interoperabilidade, o TLS visa possibilitar a utilização por parte de programadores independentes, ou seja, busca prover uma ferramenta para uso de qualquer programador que desejar utilizá-lo para qualquer projeto, sem que haja a necessidade de conhecimento aprofundado do código (DIERKS e RESCORLA, 2008).

Quanto a extensibilidade, o protocolo TLS busca fornecer um framework capaz de aceitar a utilização de novos métodos de criptografia, sempre que necessário. Com objetivo de minimizar a frequência de criação de novos protocolos, bastando apenas adicionar extensões, sempre que necessário (DIERKS e RESCORLA, 2008).

Por fim, porém sendo uma contribuição quase tão importante quanto as anteriores, o *TLS* ainda implementa a capacidade de armazenamento de sessões opcional, visando diminuir o número de conexões estabelecidas partindo do início, além de diminuir o tráfego na rede. É importante observar que a utilização das sessões armazenadas, só será realizada se tanto o cliente como o servidor, concordarem de que a sessão é confiável. Uma sessão é dita confiável se sua *hash* permanecer inalterada e se o seu certificado não for expirado. Além disso, por questões de segurança, é sugerido que as sessões possuam um tempo limite de no máximo 24 horas, antes de expirarem (DIERKS e RESCORLA, 2008).

2.3 Certification Authority - CA

Para que o protocolo SSL seja viabilizado, do ponto de vista da confiança entre as partes envolvidas da comunicação, é necessário que haja uma terceira entidade, a qual tanto o cliente, quanto o servidor a tenham como confiável. As CAs (Autoridade Certificadora), são instituições públicas ou privadas, que possuem uma estrutura hierárquica responsável pela emissão e controle do uso de certificados digitais. Os certificados digitais funcionam como uma assinatura com validade jurídica, utilizadas para prover autenticidade em serviços disponibilizados na Internet (DAHAB, 2003) (COOPER et al., 2008).

Sites com serviço de e-mail como o *Gmail* e o *hotmail*, por exemplo, possuem certificados digitais devidamente assinadas por uma determinada autoridade certificadora. O método mais simples de saber se um site utiliza o protocolo SSL é verificar o protocolo mostrado na URL do mesmo. Se estiver sendo utilizado o protocolo *https*, significa que a comunicação está utilizando um certificado digital, onde o "s", significa *security*. A validade do certificado também é informada pelo navegador. Sempre que um certificado é auto assinado, ou não consta nos registros de CAs do navegador, o usuário é informado sobre os riscos. Dessa a forma, a continuidade ou não do acesso ao site é definida pelo próprio usuário (MOZILLA.ORG, 20--) (GOOGLE.COM, 20--).

Para a identificação de uma entidade, que pode ser uma pessoa, um processo ou servidor, por exemplo, é necessário algumas informações do mesmo, como e-mail, chave pública, CPF, assinatura e nome da CA que o emitiu. Para melhor entendimento, podemos entender o uso de uma assinatura digital como uma carteira de identidade virtual (SCDP.GOV, 20--).

2.4 Sockets

Socket, ou em português soquete, consiste em uma API para prover uma interface entre a camada de aplicação e a camada de transporte (KUROSE e ROSS, 2010).

Segundo Tanenbaum (2003), *Sockets* ou *Sockets de Berkeley*, como ficou conhecido, nada mais é do que um conjunto de primitivas que possibilita a utilização dos serviços providos pela camada de transporte para as aplicações e protocolos que necessitarem de comunicação através da rede. Também pode ser referida como uma API genérica para programação de protocolos de comunicação. Nesta seção iremos apenas falar sobre o uso de *sockets* utilizado o protocolo TCP. Dada à aplicação proposta neste trabalho, é necessária a garantia de confirmação do envio e recebimento dos dados. Abaixo é apresentado o conjunto de primitivas de soquetes para TCP e suas respectivas funções:

- SOCKET: Cria o ponto final de comunicação
- BIND: Anexa o endereço local a um socket
- LISTEN: Anuncia a disposição para aceitar conexões; mostra o tamanho da fila
- ACCEPT: Bloquear o responsável pela chamada até uma tentativa de conexão ser recebida
- CONNECT: Tenta estabelecer uma conexão ativamente
- SEND: Envia alguns dados através da conexão
- RECEIV: Recebe alguns dados da conexão
- CLOSE: Encerra a conexão

Do lado do servidor deve ser utilizada a primeira primitiva *SOCKET*, para estabelecer um ponto final e alocar espaço na entidade de transporte para o *socket* que está sendo criado, além de passar como parâmetro o formato de endereçamento a ser utilizado, o tipo de serviço e o protocolo. Feito isso, é a vez de utilizar a primitiva BIND, que é responsável por atribuir um endereço de rede ao *socket* criado. Em seguida é utilizada a chamada LISTEN, responsável por alocar espaço para a fila de chamadas recebidas.

Realizada a primeira etapa de criação e escolha das diretivas de comunicação através do *socket*, deve ser utilizada a primitiva ACCEPT, que tem função bloqueante, ou seja, fica esperando por conexões e só é liberada quando um novo cliente se conectar. Sempre que um cliente receber uma conexão, a entidade de transporte deve criar um novo *socket*, com as mesmas características e propriedades do *socket* original, para que comunicação de *socket* recém-criada seja tratada por um novo processo ou thread, dessa forma liberando o *socket* original para novas conexões. A primitiva ACCEPT retorna um descritor de arquivo para ser utilizado para ler ou gravar de maneira padrão. O número de conexões possíveis de um servidor irá depender das configurações do mesmo.

Do lado dos clientes, também é utilizada á primitiva SOCKET, porém a BIND não é necessária, já que o seu endereço não é importante para o servidor. Como mencionado anteriormente, o CONNECT é bloqueante, e do lado do cliente serve para aguardar a resposta

do servidor, para que então as primitivas SEND e RECV possam ser utilizadas para envio e recebimento de dados (TANENBAUM, 2003).

2.5 Hash

Hash é um método criptográfico unidirecional ou de via única, que com base em cálculos matemáticos realizados sob um dado ou informação recebido como parâmetro, obtém como resposta um dado criptografado de tamanho fixo. Independentemente do tamanho do dado ou informação original, o resultado sempre apresentará o mesmo número de caracteres. É dito unidirecional, porque após ser realizada a criptografia, utilizado algum algoritmo de *hash* como o MD5 (RIVEST, 1992) ou o SHA-1 (EASTLAKE e JONES, 2001), não há como retornar ao dado original a partir da *hash* resultante (STALLINGS, 2005).

Basicamente a utilização de funções *hash*, é voltada para garantir a integridade da mensagem. Considerando que cada mensagem gera uma *hash* única, sempre que uma mesma mensagem passar pelo processo utilizando o mesmo algoritmo, então é dito que a *hash* resultante sempre deve ser a mesma (UFRJ.BR, 20--). Dessa forma, para testar se uma mensagem não foi alterada no caminho entre a origem e o destino, basta aplicar uma nova *hash* passando como parâmetro a mensagem recebida e testar se a resultante é igual à recebida. Se este for o caso, é considerado que a mensagem não foi alterada da origem ao destino. Dito isto, é importante ressaltar que está técnica não é suficientemente segura, visto que, em um cenário onde o atacante intercepta a mensagem e a altera, poderá criar uma nova *hash* a partir da mensagem alterada e então enviá-la ao destinatário. O teste de *hash* não apresentará nenhuma anomalia e a mensagem vai ser considerada como não alterada. Para resolver este tipo de vulnerabilidade, normalmente esta técnica é utilizada em conjunto com outras técnicas, como assinatura digital, por exemplo, que é uma técnica que será abordada no subcapítulo 2.8 (NÉTO, 2004).

2.6 Criptografia Simétrica

Criptografia simétrica ou de chave privada, consiste em um método criptográfico que utiliza a mesma chave secreta, tanto para criptografar quanto para descriptografar uma mensagem. Criptografar uma mensagem consiste em torná-la ilegível para todos que não possuírem a chave secreta para descriptografá-la. Descriptografar uma mensagem, consiste em converter uma mensagem ilegível, para texto plano, ou seja, torná-la legível novamente, através do mesmo algoritmo e chave que a criptografou (BURNETT e PAINE, 2002) (OLIVEIRA, 2012).

2.7 Criptografia Assimétrica

A Criptografia assimétrica também conhecida como criptografia de chave pública se diferencia da criptografia simétrica por possuir um par de chaves. Uma denominada pública para criptografar os dados e verificar a validade de assinaturas digitais e outra denominada privada, para decifrar os dados e realizar assinaturas digitais. O conceito baseia-se no princípio de que todo e qualquer dispositivo poderá conhecer a chave pública, já que após a informação ser criptografada pela chave pública, só será descriptografada pela chave privada que por sua vez, é de conhecimento apenas do próprio dispositivo dono do par de chaves. Nem mesmo o dispositivo que criptografou a mensagem poderá convertê-la novamente em texto plano, se este não possuir a chave privada correspondente a chave pública utilizada na criptografia (STALLINGS, 2005).

2.8 Assinatura Digital

A técnica de assinatura digital consiste em criptografar a *hash* da mensagem a ser enviada, com a chave privada do emissor. A ideia por trás da assinatura digital é garantir a identidade do emissor da mensagem, dessa forma, quando a assinatura é realizada a partir da chave privada, que como dito anteriormente, apenas o dono do par a possui, pode ser dito então que quem assinou a mensagem foi de fato quem diz ser (GANDINI et al., 2001). É perfeitamente possível que a mensagem propriamente dita seja assinada, porém a utilização da *hash* da mensagem como parâmetro para a assinatura é justificada. Uma vez que, independentemente do tamanho da mensagem, a *hash* resultante sempre terá o mesmo número de caracteres. Além disso, normalmente essa *hash* é apresentada por um número menor de caracteres, em relação a mensagem original. Uma *string* de tamanho fixo e menor, faz com que o poder de processamento, bem como o tempo gasto com a criptografia dela seja reduzido. Por fim, é importante frisar que a utilização da *hash*, como abordado anteriormente na seção 3.1, provê uma forma de garantir a integridade da mensagem, ou seja, não há necessidade de criptografar a mensagem inteira para que sua integridade seja garantida.

A utilização da chave privada para a assinatura digital se justifica pelo fato dessa chave ser conhecida apenas pelo dono do par. Dessa forma, apenas ele poderia possuir a chave para assinar a mensagem. Por outro lado, para verificar se a assinatura confere é necessário que o dispositivo receptor possua a chave pública do emissor, que como o próprio nome sugere, é uma informação pública, divulgada livremente.

2.9 Criptografia RSA

O algoritmo de criptografia RSA recebeu esse nome devido ao nome dos seus criadores Ron Rivest, Adi Shamir, e Len Adleman. Foi publicado pela primeira vez em 1978 e tornou-se um dos algoritmos mais conhecidos e utilizados desde então (MOLLIN, 2002) (NÉTO, 2004).

O algoritmo baseia-se na premissa de que a criptografia e a descriptografia dos dados são realizadas por um par de chaves, uma denominada pública, que como o nome sugere pode ser de domínio público e uma denominada privada, que é de conhecimento apenas do dispositivo, entidade ou pessoa ao qual o par de chaves pertence. Os dados criptografados com determinada chave pública, só poderão ser descriptografados pela respectiva chave privada correspondente. Sendo assim, garantindo que apenas quem possuir a chave privada poderá obter os dados em forma legível (STALLINGS, 2005).

A segurança em RSA baseia-se na dificuldade em fatorar um número grande em seus componentes primos. É importante lembrar que segundo o teorema fundamental da matemática, todo o número inteiro, positivo e maior do que um, pode ser decomposto em fatores primos (GUTIÉRREZ, 2004), porém não existe uma forma simplificada de realizar esta operação, tornando bastante custoso em termos de processamento computacional este cálculo. Dito isto, o algoritmo RSA cria o par de chaves a partir da multiplicação de dois números primos, os quais resultam em um número grande o suficiente para tornar inviável a quebra da criptografia através de cálculos. A criptografia é quebrada no momento em que são descobertos os dois números primos utilizados na multiplicação. A explicação de como é realizado todo o processo de criação de chaves, bem como a criptografia e descriptografia não serão abordadas, por não ser o foco principal do trabalho (STALLINGS, 2005).

3 DESCRIÇÃO DA PROPOSTA DE UM MIDDLEWARE SEGURO PARA COMUNICAÇÃO

A preocupação com a autenticidade dos dispositivos de uma rede, bem como a garantia de que os dados trafegados entre estes não foram modificados sempre é uma preocupação, independente da aplicação em que a rede de dados será utilizada. Porém se observamos algumas aplicações específicas, a criticidade da informação pode ser vital para o correto funcionamento dos dispositivos, tornando imprescindível que a autenticidade dos dados seja garantida, sob pena de tornar todo o sistema inapropriado para tal utilização.

Um exemplo de cenário onde manter a garantia de autenticidade dos dispositivos é um desafio, está na utilização de redes *mesh*, que são redes sem infraestrutura definidas, onde cada dispositivo conectado é um roteador em potencial. Além disso, as redes *mesh* possuem como uma de suas principais características a facilidade de inserção de novos nós na rede, o que é muitas vezes visto como uma característica positiva do ponto de vista de disponibilidade de serviço, também é uma vulnerabilidade sob a ótica de segurança de comunicação.

O *middleware* de segurança SECOM (*Security Communication Middleware*) busca prover autenticidade, integridade e confiabilidade ao tráfego de dados entre dispositivos, principalmente em sistemas cujo os dados possuem uma maior sensibilidade, como é o caso de dispositivos sensores, servidores de e-mails ou mesmo agências bancárias. Além disso, é importante frisar, que como se trata de um sistema instalado em todos os nós da rede, seu funcionamento é totalmente independente do tipo ou organização física da rede, ou seja, o *middleware* SECOM, exercerá a mesma ação tanto em redes *mesh*¸ como redes com infraestrutura definida, por exemplo.

1.1 Estrutura do sistema de segurança

Para o funcionamento correto do *middleware* SECOM, será necessário o uso de um servidor de chaves que deve conhecer todos os dispositivos conectados à rede, além de todos os que serão conectados futuramente. Este servidor será responsável por autenticar os dispositivos da rede, bem como armazenar e distribuir chaves conforme necessário. Os dispositivos da rede apenas irão se comunicar com outros dispositivos que já estiverem devidamente autenticados pelo servidor. O servidor de chaves possuirá um par de chaves assimétricas para utilizá-las na comunicação com os dispositivos. Cada dispositivo cliente deve ter posse de um par de chaves, uma pública e outra privada, além de já possuir a chave pública do servidor. Para essa implementação será utilizado o algoritmo assimétrico mais popular atualmente conhecido como RSA (OLIVEIRA, 2012), abordado na seção 2.9. Além das chaves já citadas, os dispositivos devem possuir também um identificador único, que será criado a partir de um identificador também único de seu hardware, como o MAC por exemplo. Esse identificador (ID) gerado deve ser de conhecimento exclusivo apenas do dispositivo e do servidor de chaves.

3.1 Autenticação

Esse subcapítulo descreverá as etapas da autenticação de um novo nó na rede. A autenticação será realizada utilizando criptografia assimétrica e assinatura digital para garantir a confidencialidade e a autenticidade das mensagens trocadas.

A primeira etapa é iniciada com o dispositivo cliente enviando uma requisição de autenticação para o servidor de chaves. Essa requisição deve conter a chave pública provisória do dispositivo e seu identificador (ID), criptografados com a chave pública do servidor. Na

segunda etapa, com o servidor já de posse do ID do dispositivo, é realizada uma busca por alguma referência ao dispositivo solicitante, em seu banco de chaves. Caso o ID não corresponda a nenhum dos dispositivos cadastrado, a conexão é encerrada e o dispositivo não é autenticado. Já em caso de o servidor encontrar o identificador do dispositivo solicitante em seu banco, o mesmo deve gerar um novo par de chaves para este dispositivo, salvar e relacioná-lo ao ID encontrado. Feito isso, este par de chaves deve ser criptografado com a chave pública provisória do dispositivo cliente e, por fim, enviado a ele. Dessa forma, é iniciada a terceira e última etapa da autenticação, que consiste no envio da confirmação de entrega bem-sucedida do par de chaves ou de erro no envio.

A Figura 1 ilustra as trocas de mensagens necessárias para autenticação de um novo nó na rede, ou seja, a comunicação entre o nó cliente e o servidor de chaves.

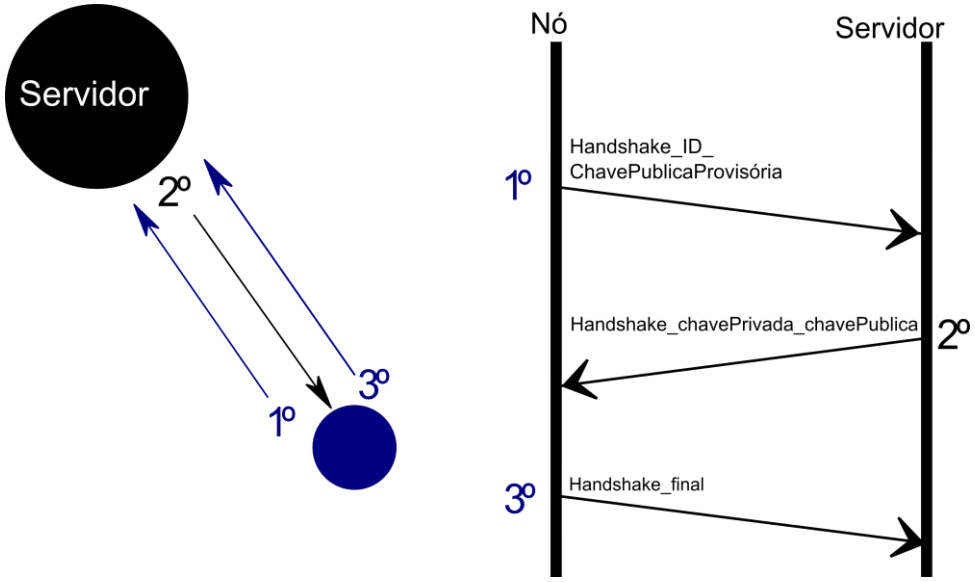

Figura 1: Comunicação Cliente/Servidor

Fonte: Acervo pessoal.

1º: Envio da requisição de autenticação do cliente, contendo: o código identificador do dispositivo e sua respectiva chave pública provisória, criptografados com a chave pública do servidor;

2º: Envio do novo par de chaves do cliente, que a partir de agora serão as permanentes, criptografada com a chave pública provisória desse cliente.

3º: Mensagem de confirmação ou erro, no recebimento do par de chaves, devidamente assinadas pelo cliente, utilizando ainda a chave privada referente ao par provisório.

3.2 Servidor de chaves

O papel do servidor de chaves na implementação desse *middleware* de segurança, é o de criar e armazenar as chaves dos dispositivos da rede. Porém, além disso, também é responsável pelo controle centralizado da entrada de dispositivos na rede, dessa forma, todos os dispositivos que farão parte da comunicação na rede, devem antes ser autenticados pelo servidor. As configurações que ditam como será realizada a comunicação após a autenticação dos dispositivos, dependerá da aplicação que o utilizará, ou seja, se as mensagens serão criptografadas de forma simétrica ou assimétrica ou então, se serão assinadas. Configurações como o tempo máximo para a revogação de chaves, e outros aspectos característicos da rede, que podem ser configurados conforme forem necessários, serão realizadas no servidor de chaves.

Todos os itens característicos do servidor serão abordados mais especificamente em itens no decorrer desta seção.

3.2.1 Armazenamento de informações dos clientes no Servidor

Como mencionado anteriormente, o servidor armazenará localmente as informações de todos os clientes que já constarem autenticados na rede. Além disso, também armazenará a informação dos que ainda serão adicionados a ela, ou seja, ao menos o ID desses dispositivos, já que apenas serão aceitos na rede dispositivos que possuírem o ID previamente cadastrados. Novos dispositivos podem ter seus IDs cadastrados no servidor, no entanto, como critério de segurança, essa operação poderá ser realizada apenas por pessoas com acesso ao servidor, dessa forma, os dispositivos cliente não poderão requisitar o cadastramento de seus IDs, em caso dos mesmos não constarem no armazenamento do servidor.

3.2.2 Revogação de chaves

Como critério de segurança é de se esperar que as chaves dos dispositivos tenham um tempo máximo de utilização, ou seja, que sejam atualizadas de tempos em tempos. Pensando nisso, será adicionada a funcionalidade de revogação de chaves dos dispositivos da rede, dessa forma, sempre que um dispositivo for autenticado e adicionado ao servidor de chaves, juntamente com as informações de ID e seu respectivo par de chaves será adicionada também a data e hora em que o mesmo foi aceito na rede. Dessa forma, este dispositivo poderá ter a chave atualizada tomando como parâmetro o tempo salvo no banco no momento da autenticação.

Além da revogação de chaves síncrona, que poderá ter um tempo pré-definido, ou mesmo editado pelo administrador da rede, também haverá a opção de revogação de chaves solicitada assincronamente. Dessa forma, se por algum motivo o administrador da rede desejar

que algum dos dispositivos atualize seu par de chaves, forçando-o a realizar uma nova autenticação, ou mesmo se desejar que todos os dispositivos sejam autenticados novamente, a operação poderá ser realizada a qualquer momento.

3.2.3 Comunicação entre dispositivo

Sempre que um dispositivo da rede for estabelecer uma comunicação com outro, o mesmo deverá ter efetuado previamente o processo de autenticação com o servidor. Devidamente autenticado, se o nó ainda não possuir a chave do dispositivo cliente a ser acessado, deverá enviar uma requisição de chaves para o servidor. Esta requisição deverá conter o endereço IP do dispositivo desejado, além de estar devidamente assinada, para provar a identidade do mesmo. A Figura 2 ilustra a comunicação entre dois nós da rede. Note que o dispositivo A já possuía a chave do dispositivo B, porém o B precisou solicitar a chave pública de A, ao servidor de chaves, para que fosse possível enviar mensagens criptografadas a ele.

Ainda na Figura 2, as setas azuis ilustram a troca de mensagens entre o SECOM e a aplicação que está o utilizando. Dessa forma todas as mensagens que devem ser criptografadas ou assinadas, descriptografadas ou validadas, antes do envio ou após o recebimento, passarão antes pelo SECOM.

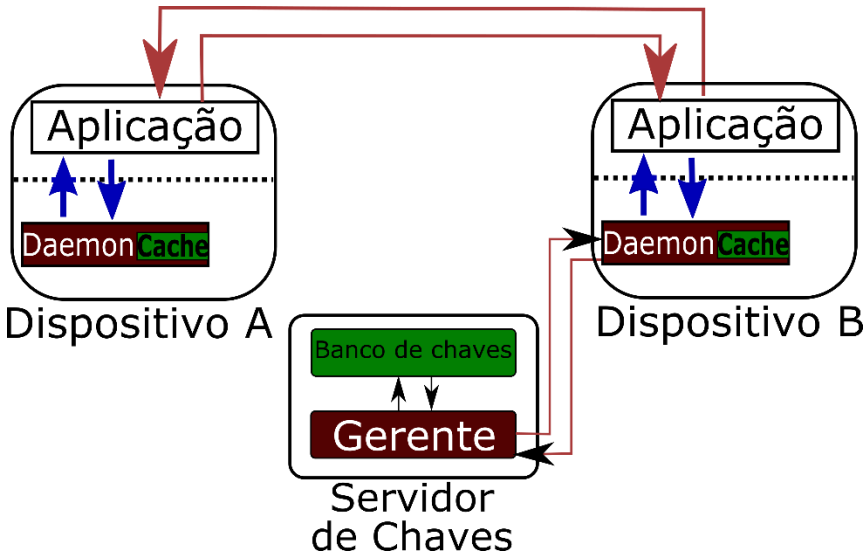

Figura 2: Comunicação entre clientes
Fonte: Acervo pessoal.

A Figura 3 ilustra um cenário onde ambos os dispositivos já possuem a chave pública um do outro, ou seja, já foram devidamente autenticados ao servidor de chaves. A aplicação A deseja enviar uma mensagem criptografada para a aplicação B, para isso, é necessário que esta aplicação utilize o *daemon* SECOM, para que seja realizada a criptografia utilizado a chave pública do dispositivo B, ou a chave secreta, no caso de a comunicação utilizar criptografia simétrica. Dessa forma a aplicação envia uma solicitação, contendo a ação a ser tomada, com a mensagem que sofrerá a ação. Neste exemplo a ação é representada pela parte inicial da *string*, onde a palavra "Requisicao" seguida da palavra "cifra", informam que a ação resulta na criptografia de uma *string.* Além disso, a palavra "assimetrica", informa o tipo de criptografia realizada. Neste caso a *string* criptografada será "Redes de Computadores". A última parte da *string* da Aplicação A é "IPdestino", que em um cenário real, consiste no endereço de IP da Aplicação B, para que o SECOM possa tomar como parâmetro sobre qual chave pública de seu cache, será utilizada para criptografar a mensagem.

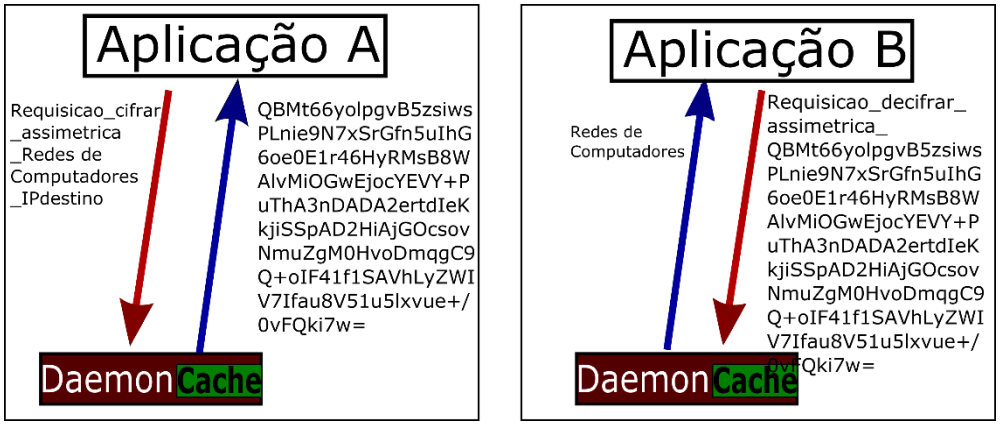

Figura 3: Comunicação Aplicação/Daemon
Fonte: Acervo pessoal.

A descriptografia realizada no dispositivo B, é realizado com a chave privada do mesmo. Sendo assim, não há necessidade de passar como parâmetro o IP, já que será utilizada sempre a chave privada do próprio dispositivo. Observe que na aplicação B, a mensagem enviada para o *daemon* consiste do tipo de ação, representados pelas palavras “Requisicao_decifrar”, além do tipo de chave que será utilizada, representada pela palavra “assimetrica”.

Na representação ilustrada de ambas as aplicações, as setas vermelhas representam a primeira mensagem trocada e as azuis representam a resposta obtida. Além disso, o retângulo que as envolve, ilustra o escopo dos respectivos dispositivos.

3.2.4 Armazenamento das chaves dos dispositivos vizinhos

O objetivo do armazenamento de chaves consiste em diminuir o número de consultas ao servidor. Visto que será estabelecido um limite de tempo pré-definido, para que seja realizada novamente a consulta de chaves ao servidor, sempre que este limite for alcançado.

Para tornar possível essa funcionalidade é necessário manter em cada dispositivo, algumas informações do dispositivo que realizou a comunicação com ele, como: o endereço IP, sua chave pública e o horário em que a chave foi adicionada.

3.2.5 Autenticidade do Servidor de Chaves

A garantia de que apenas quem tem a chave privada de um dispositivo poderá descriptografar as mensagens direcionadas a ele confere um bom nível de segurança à transferência de dados na rede, visto que a confiabilidade mencionada é proporcional a confiabilidade do algoritmo RSA, descrito no subcapítulo 2.9. No entanto, é necessário que os dispositivos que receberem os dados endereçados a eles, confiem em sua procedência.

Para que os nós da rede confiem de que o servidor de chaves é mesmo quem diz ser e não um nó malicioso se passando por ele, todos os dispositivos possuirão a chave pública do servidor, armazenada em seu banco. Toda a mensagem que supostamente for assinada pela chave privada do servidor, deverá ter sua autenticidade confirmada através da chave pública correspondente, que como dito anteriormente, estará em todos os dispositivos da rede por padrão. Além disso é importante ressaltar, que a chave pública de todos os dispositivos nesta rede, é apenas pública para os nós dispositivos que já estiverem sido autenticados. As chaves públicas nunca serão divulgadas para dispositivos externos a rede. Sendo assim, é como se a chave pública também fosse uma chave secreta, ou seja, tanto a chave privada quanto a chave pública de um dispositivo, serão chaves secretas, para todos os dispositivos que não fizerem parte da rede.

3.3 Interface de mensagens

Este subcapítulo descreve os comandos enviados para o *middleware* SECOM e suas respectivas respostas, ou seja, descreve os comandos que serão utilizados como interface entre o SECOM e as aplicações que o utilizarão.

Tipo de operação	Comando	Resposta retornada
Criptografia simétrica	Cifrar_simetrica_mensagem	*String* criptografada
Descriptografia simétrica	Decifrar_simetrica_mensagem	Texto plano em formato *String*
Criptografia assimétrica	Cifrar_assimetrica_ip_mensagem	*String* criptografada
Descriptografia assimétrica	Decifrar_assimetrica_ip_mensagem	Texto plano em formato *String*
Assinatura digital	Assinatura_assinar_mensagem	Assinatura referente a mensagem. (Apenas a assinatura)
Validação da assinatura	Assinatura_verificar_ip_	Retorna um valor booleano: true se a assinatura confere e; false se a assinatura não confere;

Quadro 1: Interface de comandos
Fonte: Acervo pessoal.

O Quadro 1 representa respectivamente os tipos de operações, os comandos que devem ser passados e as devidas respostas retornadas. Esses comandos, servirão de interface entre o *middleware* e a aplicação. Analogamente pode-se dizer que é a única linguagem em que tanto o SECOM, como as aplicações entenderão, possibilitando a comunicação entre as partes.

Qualquer comando originado na aplicação que não corresponda a nenhum dos comandos citados, será descartado pelo SECOM e retornada uma mensagem de erro. Além disso, esses são apenas os comandos de interface, não incluindo os comandos trocados com o servidor de chaves, os quais, por sua vez, são totalmente independentes da aplicação que o utilizará.

4 IMPLEMENTAÇÃO DA PROPOSTA

Esta seção descreve a implementação da proposta deste trabalho, informando sobre os métodos, linguagem de programação e ferramentas utilizadas. Além disso, demonstra o funcionamento de algumas classes utilizadas, bem como a estrutura básica dessas classes, no cliente e no servidor.

4.1 Ferramentas e linguagem de programação

A linguagem de programação escolhida para a implementação da arquitetura proposta foi o Java (ORACLE.COM, 20--). Essa escolha é justificada devido à facilidade de programação proporcionada pela linguagem, além da grande variedade de documentação disponível na Internet e em livros, e por ser preferida pelo autor do trabalho.

Como IDE *(Integrated Development Environment)*, para auxiliar no desenvolvimento da aplicação, foi utilizado o NetBeans, por ser um projeto de código aberto, gratuito, multiplataforma, de fácil utilização e bastante difundido atualmente (NETBEANS.ORG, 200-).

A configuração do computador utilizado nos testes foi a seguinte: *Notebook Dell Inspiron Serie 3000; Processador i5-5200U com 2.2 GHz e cache de 3MB; memória 8GB, DDR3L, 1600 MHz.*

4.2 Implementação do servidor de chaves

Devido as características da aplicação, o servidor de chaves deverá estar sempre "escutando" em uma determinada porta, ou seja, sempre esperando por requisições de conexão vindas dos *hosts* cliente. Sendo assim, para viabilizar esta tarefa, é necessário a implementação de uma ferramenta que utilize as funcionalidades da camada de transporte, neste caso, foi utilizada a *API* de *sockets*. O diagrama *UML*, representado na Figura 4, contém apenas as classes e métodos principais da aplicação servidor, ou seja, os necessários para a explicação da estrutura da aplicação servidor. Além disso, esse subcapítulo irá apresentar uma breve explicação sobre as classes demonstradas no digrama.

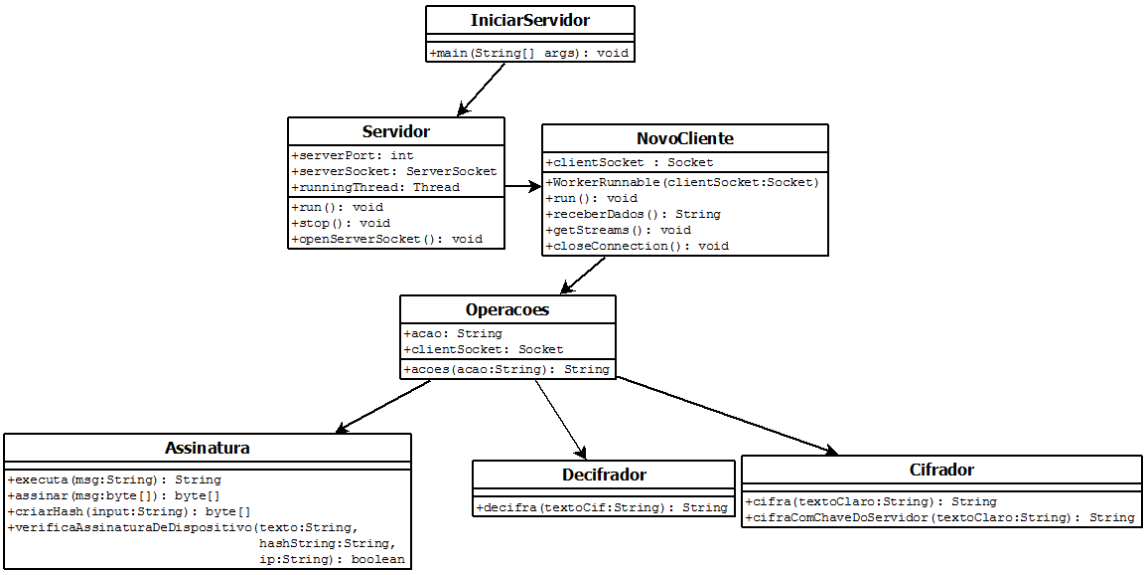

Figura 4: Diagrama UML do Servidor
Fonte: Acervo pessoal.

4.2.1 Classe Servidor

Esta classe é a responsável por executar o Servidor de *sockets*. O único parâmetro que deve ser passado no momento em que a classe Servidor for estanciada, é o número da porta lógica escolhida para as conexões dos clientes. Nesse caso, foi utilizada a porta lógica de número 1714. Após a execução do método *run()*, presente nesta classe, todos os outros passos necessários para tornar possível a conexão de clientes simultâneos são executados.

4.2.2 Classe *Thread*

Em Java, esta classe é utilizada para a criação de linhas de execução paralelas. Para isso, a classe recebe como parâmetro um objeto, contendo o código a ser executado em paralelo (ORACLE.COM, 20--) (CAELUM.COM, 20--). Para que o servidor possa aceitar múltiplas conexões de clientes simultaneamente, é necessário a utilização dessa classe, junto a classe Servidor. Dessa forma, toda a conexão recebida pelo servidor será tratada por uma nova *thread*, liberando-o para receber novas conexões. Note que esta classe não consta no diagrama, porém é estanciada e utilizada na classe Servidor.

4.2.3 Classe *NovoCliente*

A utilização dessa classe é dada da seguinte forma: após a conexão de um cliente, a classe *NovoCliente* é estanciada e recebe como parâmetro uma variável do tipo *SocketCliente*, a qual contém as informações da conexão realizada. Ao ser estanciada, o objeto criado é utilizado como parâmetro para a utilização na classe *Thread*, abordada anteriormente. Dessa forma, este cliente conectado passará a executar paralelamente em uma *thread*, liberando o acesso de novos clientes para novas conexões simultâneas, caso houverem.

4.2.4 Classe *Operacoes*

Essa é a classe responsável pela criação das mensagens a serem enviadas e pela análise das mensagens recebidas do cliente, ou seja, sempre que uma mensagem for recebida de um cliente, é essa classe que a tratará e identificará seu propósito. De forma semelhante, sempre que uma mensagem tiver de ser enviada para um cliente, ela passará por esta classe. Nela serão concatenadas as informações adicionais, como cabeçalho contendo o tipo de mensagem, e/ou a resposta de requisições, como a concatenação de um par de chaves. Todas as mensagens obedecem à sintaxe de mensagens descritas na seção 3. Dessa forma, todas as ações possíveis já possuem um script destinado a elas. Em caso de ser recebido uma mensagem que não corresponde a nenhuma requisição válida, o cliente é imediatamente desconectado.

Todas as ações realizadas no servidor de chaves têm sua execução partindo dessa classe. Sempre que um nó cliente requisitar um novo par de chaves, por exemplo, será da classe *Operacoes* que partirão os comandos necessários para a geração de um novo par de chaves. Além disso, o armazenamento no banco de chaves do servidor, com as informações do cliente devidamente associadas, a criptografia do par de chaves e por último o envio para o cliente, também são ações originadas nesta classe.

4.2.5 Demais classes utilizadas no servidor.

Para que todas as ações descritas na modelagem do projeto sejam viabilizadas foi necessário a criação de métodos que as executem. Por questão de organização e boas práticas, esses métodos foram separados em classes, conforme suas características, as quais possuem nomes mnemônicos para facilitar o entendimento

4.3 Implementação da aplicação cliente

Da mesma forma que foram abordadas algumas classes e métodos quanto a implementação da aplicação servidor, no subcapítulo 4.2, neste subcapítulo será abordado algumas das classes principais, utilizadas na implementação da aplicação cliente.

A Figura 5 representa o diagrama *UML* dos clientes. Note que somente algumas classes foram apresentadas. O objetivo da imagem é demonstrar a estrutura dos clientes. Observe que as operações foram organizadas em classes, conforme o tipo. Além disso, a maioria das ações são chamadas diretamente pela classe *Operacoes*. A ideia de estrutura tanto dos clientes como do servidor é relativamente simples, porém não foram mostradas todas as classes utilizadas, para a visualização da imagem não ser prejudicada.

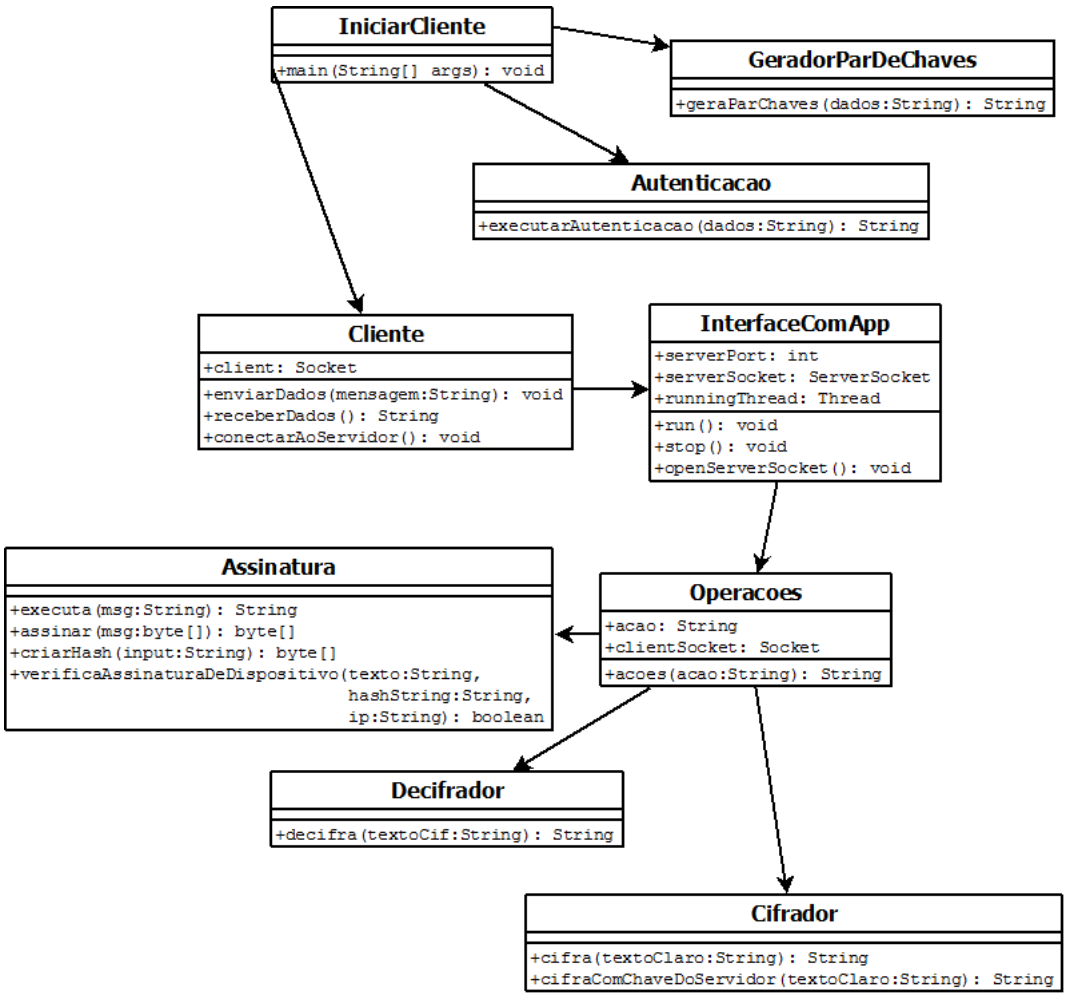

Figura 5: Diagrama UML dos Clientes
Fonte: Acervo Pessoal

4.3.1 Classe *IniciarCliente*

Esta é a primeira classe a ser executada pela aplicação cliente e partindo dela, serão estanciadas as demais classes. Quando o cliente é executado, a primeira ação executada por ele é a criação de um par de chaves provisórias. É criada uma estância da classe *GeradorParDeChaves*, e a partir do método *geraParChaves*, é gerado um par de chaves, onde

uma será pública e a outra será a privada e armazenadas logo em seguida. Essas serão as chaves utilizadas para autenticação junto ao servidor. Feito isso, o próximo passo é a autenticação com o servidor de chaves. A autenticação é realizada utilizando o método *executarAutenticacao*, da classe *Autenticacao*, que será abordada na sequência.

4.3.2 Classe Autenticacao

A classe *Autenticacao*, como o nome sugere, é responsável pela autenticação do dispositivo junto ao servidor. O método *executarAutenticacao*, realiza a autenticação com o servidor, sempre que o dispositivo for conectado na rede. Os passos executados para a autenticação são os seguintes: em primeiro lugar, tanto o identificador do dispositivo, quanto sua chave pública provisória, são criptografados e enviados ao servidor. Esses dados criptografados servirão de requisição para o início da autenticação, além de servirem para autenticar o dispositivo, através do identificador. Lembrando que a chave utilizada na criptografia dessa primeira mensagem é a chave pública do servidor, que já está no dispositivo.

Após o envio da primeira mensagem, o cliente fica aguardando a resposta do servidor, a qual será o novo par de chaves gerado pelo servidor e criptografado com a chave pública provisória, enviada na primeira mensagem de requisição, pelo cliente. Após descriptografar a mensagem, o par de chaves é salvo no dispositivo.

4.3.3 Classe *InterfaceComAPP*

Embora estejamos falando do cliente, é necessária a criação de um socket servidor, para que sejam recebidas as requisições dos clientes. Esta classe, como já mencionado, é estanciada na classe *IniciarCliente*. Por definição a porta utilizada para esta execução é a porta 7777.

4.3.4 Classe *Operacoes*

A classe *Operacoes* tem objetivo semelhante, tanto no servidor, quanto nos clientes. A grande diferença está nas ações executadas em cada uma das requisições ou respostas. O método *acoes*, presente nessa classe, é responsável por receber e responder as requisições destinadas a aplicação. Tanto mensagens vindas do servidor, como vindas das aplicações que o estão utilizando.

5 TESTES E RESULTADOS

Para a realização dos testes de utilização do *middleware* SECOM, bem como sua eficácia na execução das operações, foi desenvolvido um sistema utilizando a linguagem de programação Java. O sistema consiste em uma aplicação simples que realiza a maioria das operações propostas pelo *middleware* SECOM. As operações testadas são as seguintes: assinatura digital, criptografia/descriptografia de chave pública e verificação da assinatura.

Para a realização dos testes foi utilizado um computado com Sistema Operacional Linux Mint (LINUXMINT.COM, 20--). As capturas de telas mostradas no decorrer desta seção, foram realizadas a partir da execução da aplicação de testes, diretamente na sob a IDE NetBeans versão 8.0.2 (NETBEANS.ORG, 20--).

Outro fator que deve ser observado sobre este teste é que é foi realizado em um único computador, ou seja, não trafegou na rede, tão pouco entre aplicações. Os testes foram realizados todos em uma única aplicação, com objetivo de verificar o tempo que será somado ao tempo de execução das aplicações que irão utilizá-lo. Visto que o tempo de execução irá variar de acordo com o dispositivo que está o executando, bem como seu poder de processamento e demais limitações de hardware.

Como definido no escopo deste trabalho, o objetivo é demonstrar uma proposta de aplicação de segurança para prover um método de autenticar os dispositivos da rede e as mensagens trocadas entres eles. A preocupação com consumo de memória ou processamento não foi observada no desenvolvimento desta aplicação. Entretanto, em um cenário real esta seria uma das principais preocupações dos programadores. Dessa forma, nessa seção serão apresentadas algumas imagens demonstrando testes de medição do tempo transcorrido em cada uma das operações, bem como a explicação do significado dos dados apresentados. Na Figura 6 é apresentada a imagem da tela, do menu da aplicação de testes:

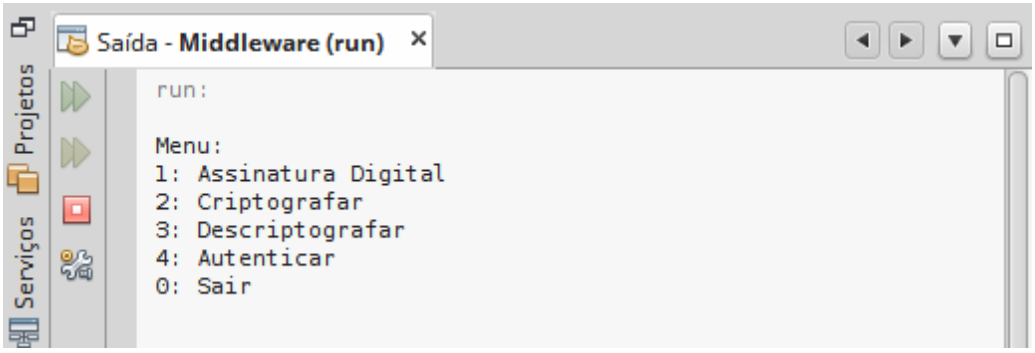

Figura 6: Menu de testes.
Fonte: Acervo pessoal.

5.1 Assinatura Digital

A primeira operação testada é a de Assinatura Digital, onde é realizada a assinatura da *string* “Redes 2015”, conforme pode ser visualizado na Figura 7:

Figura 7: Assinatura Digital.
Fonte: Acervo pessoal.

Note que o tempo demandado para a realização da assinatura na *string* digitada é de 54 milissegundos, conforme destacado pelo retângulo azul. A assinatura foi realizada logo após o início da execução do programa, ou seja, ainda não havia sido executada nenhuma das operações. O tempo mostrado antes e depois do resultado da assinatura (em vermelho), corresponde ao tempo atual do computador naquele exato momento, em milissegundo. Os dois retângulos amarelos representam, de cima para baixo, respectivamente, no primeiro a *string* a ser assinada e no segundo, a sua assinatura, precedida da mesma, ambas concatenadas com "_" (*underline*).

Realizando o mesmo procedimento a partir da *string* "Redes 2015", porém agora logo após a primeira execução, sem reiniciar a aplicação, foi verificado uma redução no tempo de operação. O tempo que era de 54 milissegundos, foi reduzido para 10 milissegundos. Além de que, a cada nova execução o tempo oscilou alguns milissegundos para mais ou para menos, porém manteve-se na média dos 10 milissegundos. Na Figura 8, as informações da segunda execução e com o novo tempo de execução destacado com o retângulo azul:

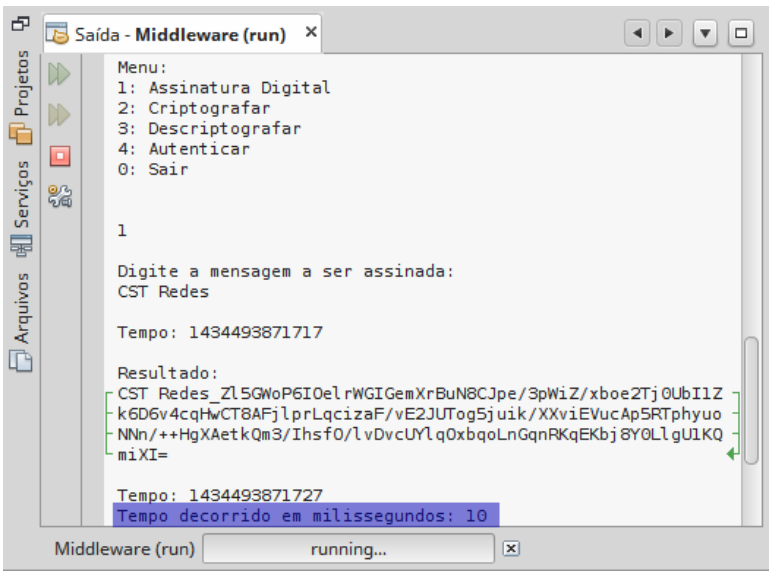

Figura 8: Assinatura Digital na segunda execução.
Fonte: Acervo pessoal.

Outro detalhe importante é saber algumas das operações que são executadas quando a opção de assinatura é solicitada. A partir da inserção da *string*, é gerado uma *hash* correspondente a ela, que por sua vez, é ainda criptografada utilizando a chave privada do dispositivo que está assinando. Chave esta, que nesse caso deve ser carregada do arquivo em que está salva. Todas essas operações demandam tempo, que se somam ao tempo total demonstrado em ambas as figuras.

5.2 Criptografando com chave pública.

A próxima operação testada foi a de criptografar dados utilizando criptografia assimétrica, que conforme o menu é a opção 2. Como estes testes foram realizados localmente, sem envio e recebimento de dados pela rede e com apenas um dispositivo, foi utilizado apenas um par de chaves. Dessa forma, os dados foram criptografados com a chave pública pertencente a esse par.

No Figura 9, a *string* destacada pelo retângulo amarelo, corresponde a entrada de dados, que será passada como parâmetro para a função de criptografia realizar a operação sobre ela e retorná-la criptografada, conforme mostrada no retângulo azul. Da mesma forma que foi explicado na operação anterior (Assinatura Digital), o mesmo ocorreu nesta operação, onde o tempo transcorrido foi maior na primeira execução da operação, logo após o programa ser executado, conforme mostra os retângulos vermelhos. Os dois primeiros retângulos correspondem ao tempo antes e depois da realização da criptografia e o último corresponde ao tempo total de criptografia.

Figura 9: Criptografia Assimétrica
Fonte: Acervo pessoal

Ao realizar pela segunda vez consecutiva a criptografia da mesma *string*, o tempo de criptografia foi reduzido, assim como ocorreu na opção de Assinatura digital e também como ocorreu nas outras operações e será demonstrado mais adiante. Como relatado anteriormente, este é um teste realizado localmente, com apenas um par de chaves e em um único dispositivo. Dessa forma, não foi necessário passar o IP como parâmetro, já que a chave utilizada é a do próprio dispositivo. A redução do tempo de criptografia pode ser verificada na Figura 10:

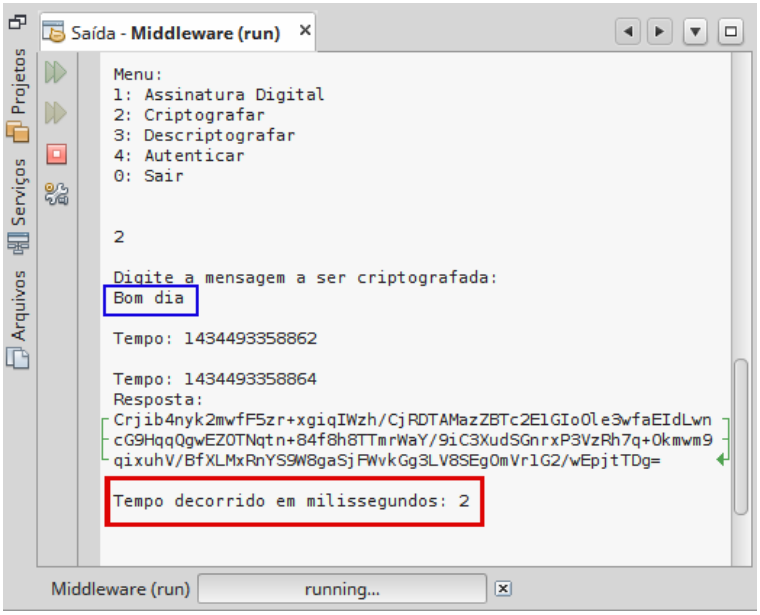

Figura 10: Criptografia Assimétrica na segunda execução
Fonte: Acervo pessoal.

Na Figura 10, o retângulo azul é a *string* que foi utilizada como parâmetro para criptografia. Como pode ser visto, neste teste foi utilizado o mesmo texto da primeira vez que foi realizada a criptografia. O retângulo vermelho representa o novo tempo de criptografia, dessa vez reduzido. Além disso, não só o tempo mudou como o resultado da criptografia também mudou, como já era esperado, a cada nova criptografia, os dados criptografados são representados por um conjunto de caracteres diferentes.

5.3 Decriptografando com chave privada

Para testar a operação de descriptografia, foi utilizada a *string* resultante do último teste de criptografia, demonstrado na Figura 10.

Na Figura 11, pode ser observado, destacado em vermelho a mensagem criptografada, passada como parâmentro para o middleware decriptografar e devolver a *string* em texto plano,

apresentada no retângulo verde, ou seja, todos os caracteres destacados no retângulo vermelho representam a forma criptografada da string “Bom dia”.

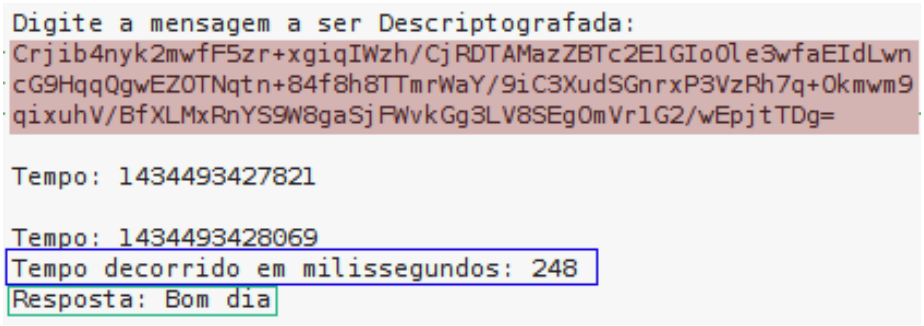

Figura 11: Descriptografia
Fonte: Acervo pessoal.

Tanto na Figura 11 quando na Figura 12, os retângulos azuis representam o tempo de criptografia. Perceba que a diferença foi bastante significativa, porém como mencionado, o tempo transcorrido em cada uma das vezes é bastante variável, além de que, após a primeira execução o tempo se mantem constante em todos os testes, com apenas pequenas variações. Em todos os testes, sempre há uma grande disparidade de tempo entre a primeira e as demais execuções consecutivas.

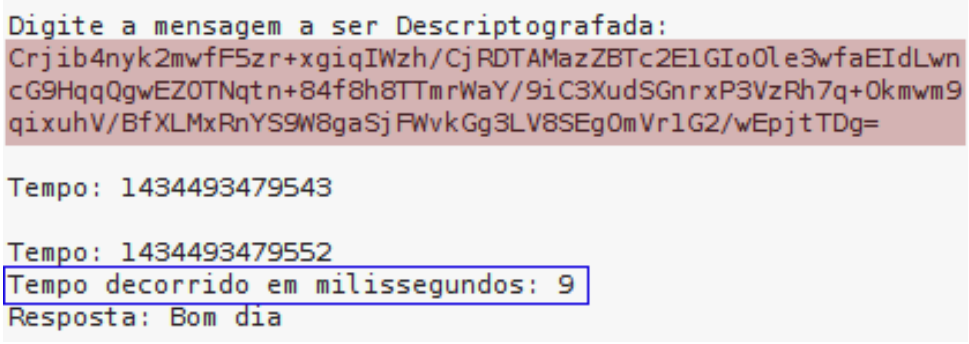

Figura 12: Descriptografia na segunda execução
Fonte: Acervo pessoal.

5.4 Verificando Assinatura Digital

A Figura 13 apresenta destacada em vermelho, a assinatura realizada no segundo teste de realização de assinatura digital, apresentado no subcapítulo 5.1 Figura 8. Além disso, note que antes e depois da assinatura, são mostradas respectivamente a mensagem propriamente dita, e o endereço de IP do dispositivo que supostamente assinou a mensagem. Basicamente, todos os caracteres envoltos pelo retângulo verde, são os caracteres correspondente a mensagem que teria sido recebida por um dos nós da rede.

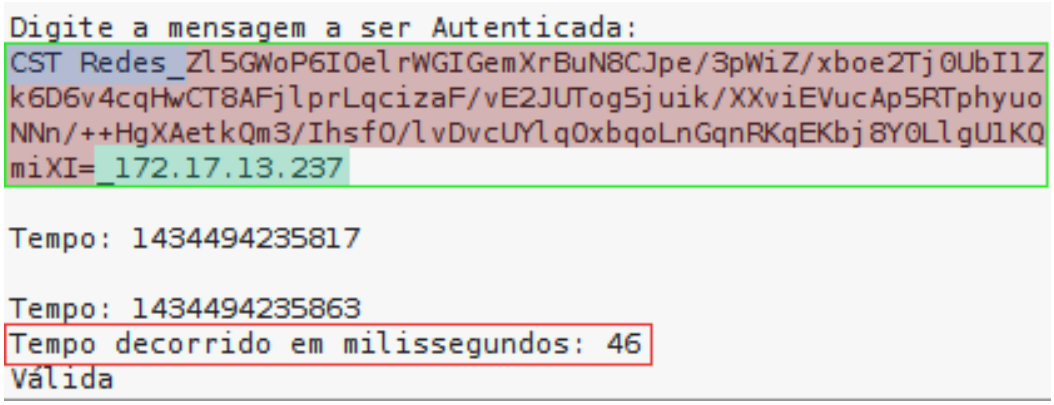

Figura 13: Teste de autenticação da mensagem.
Fonte: Acervo pessoal.

O retângulo vermelho, na parte inferior da imagem demonstra o tempo demandado para a realização da verificação da assinatura. Além disso, a mensagem de “Válida” abaixo do retângulo indica que a verificação da mensagem recebida resultou em resposta positiva quanto a validade da mensagem.

Como esperado, o tempo de demandado da segunda execução seguida da verificação da mesma assinatura foi menor. Como demonstrado na Figura 14, pelo retângulo vermelho

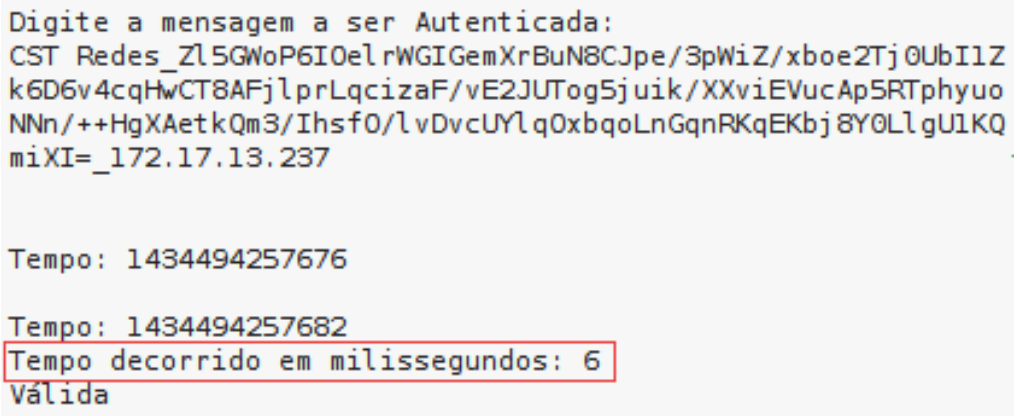

Figura 14: Teste de autenticação da mensagem na segunda execução.
Fonte: Acervo pessoal.

Uma observação importante diz respeito as operações executadas sempre que esta ação é solicitada ao middleware. Quando uma mensagem é recebida de um cliente, ela é dividida em três partes. Na primeira parte está a mensagem propriamente dita, que neste caso corresponde a “CST Redes”, na segunda parte está a assinatura digital e na terceira parte o endereço de IP do dispositivo que enviou a mensagem, que no caso é o “172.17.13.237”, todas as partes separadas por *“underlines”*.

Para verificar a validade da mensagem, o middleware cria uma nova *hash* da primeira parte da mensagem e utiliza a chave pública correspondente ao endereço de IP da terceira parte da mensagem como parâmetro. É dito que, se a *hash* resultante corresponder a *hash* recebida, a mensagem permanece inalterada. Além disso, utilizando a chave pública correspondente ao endereço de IP, pode ser garantida que o dispositivo emissor possui a chave privada correspondente, como especificado na subcapítulo 2.8, ou seja, é de fato quem afirma ser.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Por vários anos, diversos grupos de pesquisadores buscaram, sem sucesso, encontrar algumas vulnerabilidades no algoritmo de criptografia RSA. Algo que tornasse viável computacionalmente falando, quebrar a segurança proporcionada pela criptografia utilizando chaves assimétricas. O insucesso desses pesquisadores prova a eficiência da utilização de chaves assimétricas na segurança da informação (OLIVEIRA, 2012) (OKUMURA, 2014).

Porém, se compararmos as criptografias simétrica e a assimétrica, percebemos uma grande diferença, em termos do uso de processamento em cada uma delas, bem como de nível de segurança obtido. Dessa forma, o *hardware* do equipamento se torna um dos maiores obstáculos. Visto que, alguns dispositivos de redes WSN, por exemplo, possuem um *hardware* bastante modesto, buscando reduzir os gastos de bateria. Pensando nisso, várias tecnologias, assim como o *middleware* SECOM, implementam uma forma alternativa de criptografia além da criptografia assimétrica, ou seja, criptografia simétrica.

Como o SECOM é uma aplicação instalada em todos os dispositivos da rede a qual está configurada, algumas possibilidades de adição de novas funcionalidades se tornam possível. Dessa forma, como sugestão de trabalhos futuros, podem ser realizadas a adição de novas funcionalidades ao *middleware*, que por questão de tempo hábil não puderam ser implementas. Tais como:

- Descoberta de Vizinhança;
- *Time out* de armazenamento de chaves em cada dispositivo;
- Questões como formas para prover coerência nas informações dos dispositivos, ou seja, diminuir ou eliminar a possibilidade de haverem informações contraditórias a respeitos dos dispositivos, como duas chaves públicas diferentes para um mesmo dispositivo, ou endereços de IP associados incorretamente;

Outra sugestão bastante interessante seria o desenvolvimento de operações que possibilitem ao SECOM interagir com o *firewall* de cada dispositivo. Dessa forma, um nó central poderia enviar comandos para adição ou remoção de regras para todos os dispositivos da rede. Esse tipo de operação tornasse bastante interessante, principalmente para redes sem infraestrutura definida e/ou sem *firewall* centralizado.

6.1 Artigos publicados

Durante o desenvolvimento dessa monografia, foram submetidos, em um evento científico, dois artigos, os quais fazem uso da ferramenta SECOM. Um deles se intitula "*Cyber security and communications network on SCADA systems in the context of Smart Grids*", e descreve o uso do *middleware* de segurança SECOM, para tornar segura a comunicação entre os sistemas supervisórios e IEDs (Dispositivos Eletrônicos Inteligentes). O outro artigo, intitulado "*Security of communications on a high availability mesh network applied in Smart Grids*", descreve a utilização do *middleware* SECOM para garantir a autenticidade dos dispositivos participantes de uma rede *mesh*. É importante ressaltar que essa monografia foi desenvolvida como parte do trabalho de um grupo de pesquisa, coordenado pelo Prof. Tiago Antônio Rizzetti e as publicações citadas acima, fazem parte deste mesmo grupo.

Abaixo, seguem as referências, contendo os autores e o evento no qual os artigos foram publicado e submetidos:

RIZZETTI, TIAGO ANTONIO; WESSEL, PEDRO; RODRIGUES, ALEXANDRE SILVA; SILVA, BOLÍVAR MENEZES; MILBRADT, RAFAEL; CANHA, LUCIANE NEVES. *Cyber security and communications network on SCADA systems in the context of Smart Grids. In: 2015 50th International Universities Power Engineering Conference (UPEC), 2015, Staffordshire University. 2015 50th International Universities Power Engineering Conference (UPEC), 2015.*

RIZZETTI, TIAGO ANTONIO; RODRIGUES, ALEXANDRE SILVA; SILVA, BOLÍVAR MENEZES; RIZZETTI, BRUNO AUGUSTO; WESSEL, PEDRO; CANHA, LUCIANE NEVES. *Security of communications on a high availability mesh network applied in Smart Grids. In: 2015 50th International Universities Power Engineering Conference (UPEC), 2015, Staffordshire University. 2015 50th International Universities Power Engineering Conference (UPEC), 2015.*

7 REFERENCIAL BIBLIOGRÁFICO

BURNETT, S.; PAINE, S. **CRIPTOGRAFIA E SEGURANÇA:** o guia oficial RSA. Rio de Janeiro: Editora Campus, 2002.

CAELUM.COM. Caelum, 20--. Disponível em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/programacao-concorrente-e-threads/#17-1-threads>. Acesso em: 23 Junho 2015.

COOPER, D.; SANTESSON, S.; FARRELL, S; BOEYEN, S; HOUSLEY, R; POLK, W. INTERNET ENGINEERING TASK FORCE. **RFC** 5280, 2008. ISSN 5280. Disponível em: <https://www.ietf.org/rfc/rfc5280.txt>. Acesso em: 24 Junho 2015.

DAHAB, R. **O Protocolo Secure Sockets Layer (SSL)**. Universidade Estadual de Campinas Instituto de Computação. São Paulo, p. 34. 2003.

DIERKS, T.; RESCORLA, E. INTERNET ENGINEERING TASK FORCE. **RFC 5246**, 2008. ISSN 5246. Disponível em: <https://tools.ietf.org/html/rfc5246#section-1>. Acesso em: 21 Junho 2015.

EASTLAKE, D.; JONES, P. INTERNET ENGINEERING TASK FORCE. **RFC 3174**, 2001. ISSN 3174. Disponível em: <https://tools.ietf.org/html/rfc3174>. Acesso em: 18 junho 2015.

FREIER, A.; KARLTON, P.; KOCHER, P. INTERNET ENGINEERING TASK FORCE. **RFC 6101**, 2011. ISSN 6101. Disponível em: <tools.ietf.org/html/rfc6101>. Acesso em: 20 Junho 2015.

GANDINI, J. A.; SALOMÃO, ; JACOB,. A SEGURANÇA DOS DOCUMENTOS DIGITAIS, 2001. Disponível em: <http://egov.ufsc.br/portal/sites/default/files/anexos/27250-27260-1-PB.pdf>. Acesso em: 24 Junho 2015.

GOOGLE.COM. Google Chrome. **support.google**, 20--. Disponivel em: <https://support.google.com/chrome/answer/95572?hl=pt-BR>. Acesso em: 24 Junho 2015.

GUTIÉRREZ, F. J. G. Apuntes de Matemática Discreta 11. Teorema Fundamental de la Aritmética, 2004. 30.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet:** uma abordagem top-down. 5ª. ed. São Paulo: Pearson, 2010.

LINUXMINT.COM. Linux Mint, 20--. Disponivel em: <http://linuxmint.com/>. Acesso em: 23 Junho 2015.

MACHADO, ; MAIA,. **Arquitetura de Sistemas Operacionais**. 4ª. ed. Rio de Janeiro: LTC Editora, 2007.

MOLLIN, R. A. **Public-key Cryptography:** Theory and Practice. 1ª. ed. [S.l.]: Chapman & Hall/CRC, v. I, 2002.

MOZILLA.ORG. Mozilla Suport, 20--. Disponível em: <https://support.mozilla.org/pt-BR/kb/erro-esta-conexao-nao-e-confiavel>. Acesso em: 24 Junho 2015.

NETBEANS.ORG. NETBEANS, 20--. Disponível em: <https://netbeans.org/>. Acesso em: 23 Junho 2015.

NÉTO, C. J. **Segurança em Redes Móveis Ad Hoc**. Universidade de São Paulo. São Paulo, p. 30. 2004.

OKUMURA, M. K. Números primos e criptografia RSA. **Dissertação**, São Paulo, 21 Março 2014. 41.

OLIVEIRA, R. R. Criptografia simétrica e assimétrica - os principais algoritmos de cifragem, v. 31, p. 11-15, 2012.

ORACLE.COM. **Java**, 20--. Disponível em: <http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>. Acesso em: 23 Junho 2015.

RIVEST, R. L. Request for Comments. **INTERNET ENGINEERING TASK FORCE**, 1992. ISSN 1321. Disponível em: <https://www.ietf.org/rfc/rfc1321.txt>. Acesso em: 18 jun. 2015.

SCDP.GOV. SERPRO, 20--. Disponível em: <https://www.scdp.gov.br/Manual/CertificacaoDigital/Roteiro_Certificacao_Digital.htm>. Acesso em: 24 Junho 2015.

STALLINGS, W. **Cryptography and Network Security**. 4ª. ed. [S.l.]: Prentice Hall, 2005.

TANENBAUM, A. S. **Redes de computadores**. 4ª. ed. São Paulo: Campus, v. XVIII, 2003.

UFRJ.BR, 20--. Disponível em: <http://www.gta.ufrj.br/grad/09_1/versao-final/assinatura/hash.htm>. Acesso em: 18 Junho 2015.