CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Desenvolvimento de API Criptográfica para Comunicação Segura entre Aplicações Web: Estudo de Caso Aplicado em Web Service REST

DRIELY DA SILVA AOYAMA

Marília, 2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Desenvolvimento de API Criptográfica para Comunicação Segura entre Aplicações Web: Estudo de Caso Aplicado em Web Service REST

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação
Orientador: Prof. Fábio Dacêncio Pereira

Marília, 2016

Aoyama, Driely da Silva

Ficha catalográfica

62 f.

Trabalho de Curso (Graduação em Ciência da Computação) - Curso de Ciência da Computação, Fundação de Ensino “Eurípides Soares da Rocha”, mantenedora do Centro Universitário Eurípides de Marília –UNIVEM, Marília, 2016.

1. Criptografia 2. Segurança 3. Comunicação 4. Web

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Driely da Silva Aoyama

Desenvolvimento de API Criptográfica para Comunicação Segura entre Aplicações Web: Estudo de Caso Aplicado em Web Service REST

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,0 (nove)

Orientador:Fábio Dacêncio Pereira

1°.Examinador:Emerson Alberto Marconato

2°.Examinador:Fabio Lucio Meira

Marília, 05 de dezembro de 2016.

DEDICATÓRIA

Oferecer, empenhar-se, homenagear, dedicar!

Oferecer à Ele, Deus de promessas. À Quem se fez cumprir palavra e sabedoria para os momentos em que desistir era motivo para entregar-se.

Empenhar-se. Investir tempo de estudo contínuo, abster-se dos finais de semana de lazer, substituir horas de sono pelas extensas pesquisas. Qualquer ação que levasse ao alcance do tão esperado objetivo: a graduação!

Homenagear. Àqueles que durante os cinco extensos anos proporcionaram as melhores experiências acadêmicas, transferindo conhecimento e oferecendo oportunidades para crescimento pessoal e profissional, queridos professores.

Enfim, **dedicar** todo o esforço empreendido e a conquista alcançada àqueles que foram o suporte, o incentivo, que se fizeram presentes e compreensivos no decorrer desta longa caminhada: minha família!

AGRADECIMENTOS

Agradecer é o ato mais nobre que podemos oferecer aos que sempre estiveram torcendo para o sucesso de mais uma etapa concluída. E, portanto, nada mais justo do que demonstrar o quão grata sou àqueles que me apoiaram, me deram forças e coragem para encarar o medo para continuar quando eu nem sabia que elas existiam mais.

Reconhecer todo o mérito e honra de estudar em instituições na qual os educadores são os maiores incentivadores à busca de novos conhecimentos, novos horizontes. Horizontes que me fizeram romper as fronteiras do medo do desconhecido e as fronteiras geográficas, proporcionando uma das maiores experiências já vividas por mim: conhecer o Canadá. Obrigada, Fundação Bradesco de Marília e Fundação de Ensino "Eurípedes Soares da Rocha" – UNIVEM.

Retribuir toda dedicação que recebi dos meus familiares, desde o investimento financeiro até a compreensão aos dias difíceis e agradecer, principalmente, a ele que trocou suas noites, para ser minha companhia de estudo, que foi o meu ombro amigo quando o desespero batia à porta, que foi meu professor de plantão as 24 horas do dia. Que foi paciente, flexível, dedicado, parceiro: Marcus Vinicius Bassalobre de Assis. O que seria da minha jornada se não tivesse você para segurar a minha mão? É gratificante reconhecer que tive você como o meu ponto de equilíbrio.

Obrigada! Obrigada a todos que me fizeram chegar até aqui com sabedoria e perseverança.

"Todo caminho da gente é resvaloso.
Mas também, cair não prejudica demais
A gente levanta, a gente sobe, a gente volta!...
O correr da vida embrulha tudo, a vida é assim:
Esquenta e esfria, aperta e daí afrouxa,
Sossega e depois desinquieta.
O que ela quer da gente é coragem.
Ser capaz de ficar alegre e mais alegre no meio da alegria,
E ainda mais alegre no meio da tristeza..."

Rosa, J. G. Grande Sertão: Veredas. Rio de Janeiro: Nova Fronteira, 2005.

RESUMO

Satisfazer os critérios de segurança de um projeto de software em uma fase tardia do processo de desenvolvimento, pode ser uma decisão arriscada, resultando em impactos na produção que podem comprometer toda a estrutura do projeto. O trabalho apresentado é um estudo sistemático para desenvolvimento de uma API de apoio aos desenvolvedores de aplicações que não possuem conhecimentos específicos em segurança da informação e, desejam que as informações produzidas pelos seus softwares permaneçam seguras durante a transmissão ocorrida entre aplicações web, além de permitir uma definição dinâmica, em tempo de execução, para o sistema desenvolvido.

O desenvolvimento da API visa incluir um selecionador automático de algoritmos criptográficos, encolhendo a cifra mais indicada para criptografar cada cenário, com base em um critério de parâmetros que identifica os cenários apresentados pelo cliente da API. Dessa forma, utilizando uma inteligência escalável, a API passa a oferecer ganhos no desempenho, no que se refere a escolha de cifras adequadas para uma performance considerável, e na segurança da transmissão de dados.

Palavras-Chave: API, Criptografia, Comunicação, Web Service, Segurança, Automatização, Tomada de decisão.

ABSTRACT

To satisfy the security criteria of a software project at a late stage of development can be a risky decision and resulting in impacts on productivity that may compromising all the Project structure. So, the work presented is a systematic study to develop an API to support application developers' that are not known specific security information and want the information produced by the software remain secure during transmission occurred between web applications, in addition to enabling a dynamic setting in runtime for the system developed.

The development of API is an automatic feature of cryptographic algorithms that have been chosen according to criterion of some parameters that identify the most suitable cipher for each scenario and then it will perform an encryption of information. Thus, using a scalable intelligence, the API providing gains in performance, with regard to proper choice for the good performance, and security of data transmission.

Keywords: API, Cryptography, Communication, Web Service, Security, Automation, Decision-making.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

W3C	*World Wide Web Consortium*
API	*Application Programming Interface*
REST	*Representational State Transfer*
IBM	*International Business Machines*
TI	Technology Information
WSDL	Web Services Description Language
XML	Extensible Markup Language
UDDI	*Universal Description, Discovery and Integration*
SOAP	*Simple Object Access Protocol*
HTTP	*Hyper Text Transfer Protocol*
RPC	*Remote Procedure Call*
SMTP	*Simple Mail Transfer Protocol*
BEEP	*Blocks Extensible Exchange Protocol*
JSON	*JavaScript Object Notation*
MIME	*Multipurpose Internet Mail Extensions*
URL	*Uniform Resource Locator*
OWASP	*Open Web Application Security Project*
SQL	*Structured Query Language*
SO	Sistema Operacional
LDAP	*Lightweight Directory Access Protocol*
XSS	*Cross-site scripting*
CSRF	*Cross-site request forgery*
DES	*Data Encryption Standard*
AES	*Advanced Encryption Standard*
IDEA	*International Data Encryption Algorithm*
RFID	*Radio-Frequency Identification*
RSA	Ronald Rivest, Adi Shamir e Leonard Adleman (autores do algoritmo RSA).
MIT	*Massachusetts Institute of Technology*
MD5	*Message-Digest algorithm 5*

SHA	*Secure Hash Algorithm*
NIST	*National Institute of Standards and Technology*
RC	*Round Constant*
PHP	*Personal Home Page*
TLS	*Transport Layer Security*
SSL	*Secure Sockets Layer*
ECC	*Elliptic Curve Cryptography*
RAM	*Random-Access Memory*
MB	*Megabyte*
GB	*Gigabyte*
URI	*Uniform Resource Identifier*
XKMS	*XML Key Management Specification*
TI	*Tecnologia da Informação*

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O primeiro capítulo contextualiza o projeto, além de explicar os objetivos gerais e específicos, a problemática que foi solucionada, a justificativa que estimulou o desenvolvimento do trabalho e a metodologia para construção do mesmo.

1.1 Contextualização

A crescente demanda dos serviços disponibilizados pela indústria de software na Internet e a necessidade de troca de contexto entre tais aplicações, descrevem o cenário atual do mercado de software que vêm seguindo a tendência de utilização da Internet como principal meio de interatividade global.

A integração entre estes serviços oferecidos de forma transparente e consideravelmente "instantânea", tornou-se o principal motivo para que se definisse um mecanismo para apoiar interações entre máquinas através de uma rede. A este mecanismo, deu-se o nome de Web Service, segundo a W3C (*World Wide Web Consortium*), (Christensen, 2001).

Apesar de facilitar a troca de informações entre aplicações, os Web Services ainda enfrentam dificuldades em oferecer um ambiente protegido e totalmente confiável. Neste contexto, a criptografia é inserida, como sendo uma das melhores alternativas para criar um espaço seguro para transferência de informações relevantes.

Alguns trabalhos correlatos têm soluções para processos de comunicação segura entre cliente e Web Service, os quais pode-se destacar a dissertação de mestrado apresentada ao Instituto de Ciências Matemáticas e Computação da USP pelo autor Douglas Rodrigues que propõe

> "...uma avaliação e comparação de diretrizes e a adequação de técnicas que permitam não somente a criação de Web services seguros, mas também a validação dos serviços utilizados para determinar se a aplicação possui as características almejadas [sic] relacionadas ao desempenho e à segurança" (Rodrigues, 2011).

Outro trabalho que apresenta semelhanças com o proposto neste projeto é um artigo elaborado pelos autores Reginaldo F. Silva e José A. Cunha que “aborda conceitos e aspectos de segurança relativos aos Web Services” (Silva e Cunha, 2005).

Ambas correlações apontam estudos e análises relacionadas à segurança da informação trafegada entre Web Services e aplicações, no entanto, nenhum deles apresentam uma implementação automatizada de um selecionador de criptografias, o qual é proposto aqui.

1.2 Problemática

Os Web Services apresentam uma proposta de automatizar a comunicação entre aplicações executadas em diferentes plataformas através da Internet, permitindo o tráfego de informações relevantes entre emissor e o receptor. No entanto, tal comunicação, por utilizar a web como ponte de transição de dados, torna-se sujeita à ataques e outras atividades ilícitas do mundo virtual que compromete a segurança dos dados.

Assim, ao levar em consideração que as principais limitações que hoje atingem os Web Services estão relacionadas a segurança e privacidade, visualizou-se a possibilidade de desenvolvimento de uma API criptográfica capaz de consumir serviços web e permitir a transição de dados de forma segura, selecionando o melhor algoritmo criptográfico para determinada situação (Júnior, 2016).

1.3 Justificativa

É notável que o acesso à Internet tem apresentado um crescimento exponencial nos últimos anos; observa-se que, junto ao crescimento, os crimes virtuais têm tomado seu espaço de forma proporcional. Interceptação de informações sigilosas, furto de identidade e e-commerces fraudulentos são exemplos de atividades maliciosas cujo o objetivo é aplicar golpes no usuário desinformado.

Portanto, a fim de mitigar as vulnerabilidades às quais toda e qualquer aplicação web está exposta, protocolos e técnicas criptográficas começaram a ser exploradas acerca das medidas de segurança necessárias para determinadas aplicações. Todavia, incluir seções seguras em projetos de desenvolvimento de software para diminuir vulnerabilidades, tornou-se uma

atividade um tanto quanto complexa à medida que os problemas de segurança foram surgindo, já que para solucioná-los o desenvolvedor deveria dispor de conhecimentos específicos na área de segurança para produzir um software significativamente seguro.

Assim, levando em consideração as dificuldades atuais enfrentadas pelos desenvolvedores desprovidos de conhecimentos específicos relacionados a segurança da informação, este projeto vem a ser um facilitador para os programadores que possuem aplicações web e que estão suscetíveis aos ataques desferidos na web, proporcionando ao software um suporte à decisão do algoritmo de criptografia mais indicado para prover a segurança de determinado processamento.

1.4 Objetivos Gerais

O objetivo principal deste projeto é o desenvolvimento de uma API criptográfica para comunicação segura entre aplicações Web, apresentando uma forma automatizada de escolher os melhores algoritmos de criptografia adequados a determinados cenários nos quais haja consumo de serviço web.

Este trabalho contempla também os três requisitos básicos de manipulação segura de dados, os quais cito: confidencialidade, integridade e autenticidade, buscando o equilíbrio entre os requisitos de segurança e desempenho, conforme o cenário descrito pelo desenvolvedor usuário da API.

1.5 Objetivo Específicos

O projeto, em sua totalidade, atendeu aos seguintes requisitos:

• Estudo de criptografia com cifras simétricas, assimétricas, hash e leves (*lightweight*).

• Classificação das criptografias estudadas por níveis de capacidade de segurança.

• Comparação, avaliação e escolha das melhores criptografias para criação do tráfego de informações cifradas.

• Desenvolvimento da API automatizada para seleção de cifras criptográficas.

• Estudo de padrões de Web Service que permitem expansão e escolha do padrão REST, adotado como estudo de caso do projeto.

• Teste de desempenho dos algoritmos implementados, apresentando resultados que mostraram o nível de performance do padrão executado com diferentes tamanhos de arquivos.

• Teste da API finalizada, mostrando o funcionamento da mesma.

1.6 Metodologia

Para o desenvolvimento deste trabalho foi realizado um estudo de criptografia simétrica, assimétrica, hashes e blocos de cifras leves, analisando especificamente o comportamento das mesmas ao criptografar mensagens de diferentes tamanhos. Todas as informações de relevância retiradas desta análise, proporcionou a criação de uma tabela condicional para tomadas de decisões no que se refere a escolha dos algoritmos que compõem a implementação da API.

Todos os algoritmos criptográficos estudados foram classificados por níveis de capacidade de segurança. Ou seja, algumas variáveis foram comparadas, a fim de obter resultados que apontem as principais vulnerabilidades das criptografias, para que fosse possível escolher os algoritmos que melhor se encaixam na execução deste projeto.

Uma revisão de padrões de Web Services existentes ofereceu a base de apoio para definição do padrão a ser adotado no estudo de caso do projeto. O critério de seleção do Web Service obedeceu a condição para permissão de expansão do modelo escolhido, uma vez que a proposta de implementação é baseada na possibilidade de expandir um Web Service que atenda as rotinas de tratamento de segurança propostas.

O estudo de caso para comunicação entre cliente e Web Service possibilitou a automatização das rotinas de segurança, através um "handshake[1]" entre o cliente e o Web Service, utilizando a API desenvolvida para avaliação do melhor algoritmo criptográfico a ser utilizado em determinada situação. Após a avaliação, uma seção segura deve ser criada, autorizando o tráfego de informações cifradas com o algoritmo selecionado.

Os testes para validação do projeto foram executados por meio da avaliação de desempenho dos algoritmos (tamanho de mensagem x tempo de processamento). Trata-se de uma avaliação de grande importância na tomada de decisão, já que a característica principal deste desenvolvimento é a automatização da escolha do algoritmo criptográfico que melhor

1 Handshake: é o processo pelo qual duas máquinas afirmam uma a outra que a reconheceu e que está pronta para iniciar a comunicação.

atende um determinado cenário, proporcionando desempenho considerável e segurança adequada.

1.7 Organização do Trabalho

Esta monografia foi organizada, conforme a relação a seguir:

No capítulo 2, apresenta-se uma revisão de conceitos de Web Services, especificamente SOAP e REST.

No capítulo 3, são relacionados alguns aspectos no que diz respeito a segurança em sistemas web, apresentando algumas das principais vulnerabilidades deste ambiente, bem como os conceitos de criptografia e suas vertentes.

No capítulo 4 é exposta a estrutura da arquitetura da API desenvolvida, além de estudos correlatos à proposta deste trabalho.

No capítulo 5, são representadas as etapas da implementação da API, além de apontar um estudo de caso de utilização da API.

No capítulo 6, os resultados do desenvolvimento da API são relatados, evidenciando questões relacionadas ao desempenho e segurança de algoritmos criptográficos e ao funcionamento e execução da API proposta.

No capítulo 7, são apresentadas as conclusões do trabalho, listando dificuldades enfrentadas e contribuições oferecidas para uma comunicação criptografada.

2 WEB SERVICES: ASPECTOS CONCEITUAIS

Neste capítulo são apresentados os aspectos conceituais de uma tecnologia que permite a interação de diversas aplicações desenvolvidas em plataformas distintas, tornando-as compatíveis entre si através da comunicação realizada pela utilização de uma linguagem padrão conhecida pelos sistemas. A esta tecnologia chamamos de Web Service.

2.1 Web Services: Aspectos Conceituais

A diversidade entre aplicações desenvolvidas atualmente, provocou a necessidade do desenvolvimento de um serviço capaz de permitir a interação entre plataformas e sistemas operacionais diferentes. Assim os Web Services surgem com o principal objetivo de proporcionar a viabilidade de comunicação entre estas aplicações.

Portanto, de acordo com a IBM *developWorks* "um Web Service é um conjunto de padrões emergentes que permitem a integração interoperável entre os processos e sistemas de TI heterogêneos. Ou seja, é possível descrever Web Service como uma aplicação web autossuficiente e autossustentável que fornece meios de comunicação entre básico até o mais complicado serviço aplicação existente" (tradução nossa) (Colan, 2004).

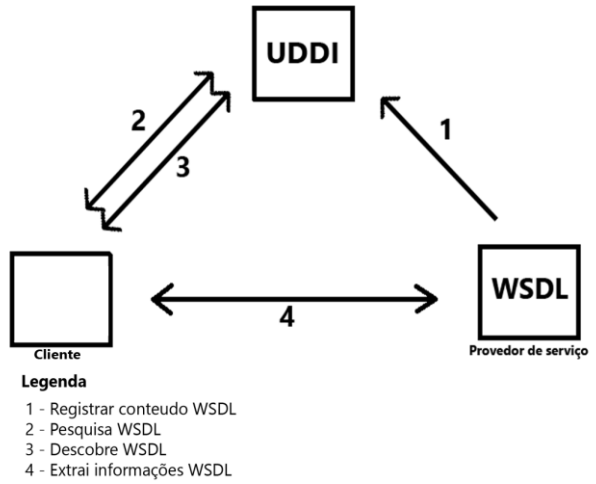
Para que os Web Services possam ser integrados com outros serviços e aplicações, é necessário que um determinado padrão seja reconhecido entre as aplicações/serviços e que possua uma interface compatível. Deste modo, a WSDL (*Web Services Description Languages*) tem um importante papel na geração dos tipos de descrições padronizadas. Tais descrições são tipicamente definidas em formato XML, o qual é requerido como uma linguagem comum para transmissão de dados entre aplicações que fazem o uso de Web Services.

A WSDL trabalha como um índice onde estão relacionadas todas as informações necessárias para que determinada aplicação faça suas requisições. Esta documentação possui vários tipos de parâmetros necessários para estabelecer a comunicação entre os softwares, entre os quais podemos citar tipos de dados, parâmetros de entrada e saída de serviço, o protocolo a ser utilizado para realizar a troca de informação com o serviço, entre outros.

Uma vez que os parâmetros para requisição de serviços web são definidos pela WSDL, é preciso que os mesmos se tornem conhecidos para serem utilizados pelas aplicações. Para isso, o registro UDDI (*Universal Description Discovery and Integration)* é aplicado no processo, procurando fornecer os métodos para publicação e localização das descrições dos Web Services (Brittenham, 2001).

Dessa forma, fica estabelecido o caminho pelo qual uma aplicação deve seguir quando quiser realizar comunicação com outro serviço: consulta o UDDI para descobrir o local onde está armazenado o documento WSDL da aplicação com a qual deseja conversar, ao encontrá-lo, extrai todas as informações de como deverá se comunicar com o Web Service e, por fim, envia a mensagem utilizando um dos padrões de protocolo projetados para invocar a troca de mensagens com o destinatário, o SOAP (*Simple Object Access Protocol*) por exemplo. A figura 1 ilustra o procedimento descrito.

Figura 1- Modelo de Comunicação entre Cliente e Web Service (autoria própria, 2016)



As bases para a construção de um Web Service são os padrões XML e SOAP, enquanto o transporte dos dados é realizado, normalmente, via protocolo HTTP.

Neste processo de construção, os dados são transferidos para o formato XML e encapsulados pelo protocolo SOAP. Desta forma, as aplicações não precisam estar necessariamente na mesma linguagem ou utilizando o mesmo sistema operacional, uma vez

que a comunicação estabelecida passa a utilizar um padrão comum de transmissão de mensagens, e portanto, os arquivos recepcionados provenientes de qualquer aplicação é claramente compreendido e interpretado sem distinção.

Dentre as principais padronizações de comunicação com Web Services, as duas de maior destaque são o protocolo SOAP e o modelo de design REST (Fidel, 2015).

2.2 Protocolo SOAP

Conforme a W3C (BOX et al, 2000, tradução nossa), "o SOAP é um protocolo leve utilizado para troca de informações em ambiente descentralizado e distribuído". Este protocolo pode ser dividido em três partes: SOAP Envelope que define um *framework* para descrever o que é a mensagem e como processá-la, SOAP *Encoding* que é um conjunto de regras de codificação utilizado para expressar instâncias de tipos de dados definidos pela aplicação e uma convenção para representar chamadas de procedimentos remotos (RPC).

No geral, uma mensagem SOAP representa a informação necessária para invocar um serviço ou analisar o resultado de uma chamada e contém informações específicas da definição da interface do serviço. Desse modo, SOAP pode ser facilmente combinado diversos protocolos, sendo o HTTP o mais utilizado atualmente.

Segundo Suda (2003), o protocolo SOAP possui diversas vantagens entre as quais estão:

• Possibilidade de trabalhar com qualquer plataforma, sistema operacional, ambiente computacional, linguagem de computação e qualquer protocolo.

• Possui protocolo baseado em XML e, por este motivo, inclui boa formação na estrutura dos documentos, texto legível e consumível em máquinas.

• Independência de transporte, o que significa que mensagens SOAP podem ser transportadas por qualquer tipo de protocolo, sendo ele síncrono ou assíncrono, dentre os quais podemos citar o HTTP, SMTP, BEEP, etc. Desse modo, o SOAP é flexível e estará sempre se adaptando às mudanças dos protocolos para encontrar novas maneiras de continuar enviando as mensagens.

•Possui grande capacidade de interoperabilidade, por conter dados codificados em XML. Ou seja, SOAP tem habilidade de trabalhar com interfaces de aplicações que nunca operaram entre si previamente, mas que agora conseguem se comunicarem por meio do código padrão adotado, o XML.

No entanto, ainda de acordocom Suda (2003), há também desvantagens a serem consideradas, como:

• Possui alguns problemas comuns de inclusão de tipos de dados como string, integer, date, etc. Estes tipos de dados não são os mesmos entre diferentes linguagens de programas, havendo necessidade de estabelecer a necessidade de utilização de um padrão (XML SCHEMAS) para implementação dos códigos.

• Problemas de segurança: SOAP sozinho não suporta todas as práticas de segurança necessárias em um ambiente computacional como a acessibilidade, confidencialidade, autenticação, autorização e integridade, por exemplo.

Assim, conforme as vantagens citadas, embora haja protocolos de segurança que possam auxiliar na proteção dos dados durante o transporte do mesmo para um Web Service SOAP, a segurança apenas em nível de rede não é o suficiente para garantir esta proteção já que, sozinho, o SOAP não suporta todas as práticas de segurança necessárias.

2.3 Estilo de Arquitetura REST

Enquanto isso, o REST (*Representational State Transfer*) é denominado um estilo de arquitetura idealizado por Roy Fielding no ano de 2000 em sua tese de doutorado, com base nos melhores estilos de arquiteturas existentes na época (Lima, 2012).

Inicialmente, o protocolo HTTP possuía a única função de receber informação de um servidor localizado através de uma URL e enviar uma informação ao mesmo. Para este tipo de procedimento, utilizava-se apenas o GET e o POST, apesar de possuir outros métodos de requisição disponíveis. No entanto, a partir da definição de REST de Fielding, quatro principais métodos tornaram-se explicito nos serviços web do tipo REST: GET, POST, PUT e DELETE (Rodriguez, 2008).

• GET é utilizado para recuperar recursos, buscar dados em um servidor web ou ainda executar consulta para receber como resposta um conjunto de recursos correspondentes.

• Para criar um recurso no servidor, utiliza-se o POST.

• Para mudar o estado dos recursos ou altera-lo, utiliza-se PUT.

• Para remover ou deletar os recursos, utiliza-se DELETE.

Tais métodos juntos são considerados um dos quatro princípios básicos de REST sugeridos pelo grupo de autores da IBM *developWorks* no artigo **RESTful Web services: The basics**, os quais são relacionados a seguir*:*

• forma de representação dos recursos no momento em que uma determinada aplicação o solicita. Neste princípio, as informações da aplicação são transferidas para um modelo em formato XML ou JSON, onde os registros do banco de dados são mapeados e, através das colunas e linhas, os atributos são criados para inserção no modelo XML/JSON. Uma característica muito interessante do HTTP para o tipo MIME (*Multipurpose Internet Mail Extension*) é que existe a possibilidade do cliente e servidor negociarem o formato dos dados que os mesmos irão trocar, podendo ser diferente das representações de XML ou JSON.

• Servidor *Statelles*. Outro princípio diz que servidor não deve armazenar informações do cliente para facilitar a escalabilidade e desempenho do serviço, já que a atividade do servidor é responder às solicitações dos clientes sem depender de informações realizadas por requisições anteriores.

• Por fim, as URLs de serviços web do tipo REST devem ser intuitivas, de modo que não necessitem de muitas informações para um desenvolvedor entender que àquele endereço aponta para um determinado recurso. Desse modo, a estrutura de uma URL deve ser considerada como uma interface simples, auto documentada, previsível e fácil de ser compreendida. Uma forma de alcançar este nível de usabilidade é definir estruturas de diretórios de URLs do tipo hierarquias de ramificações, a partir de um caminho principal, por exemplo:

http://www.myservice.org/discussion/topics/{subtopic}

A raíz /discussion possui um nó /topics abaixo dela. Desse modo, qualquer sub tópico que tiver após o nó tópico, poderá ser adicionado à URL, acrescentando /{subtopic} (Rodriguez, 2008).

2.4 Considerações Finais do Capítulo

Neste capítulo foram abordados os principais conceitos sobre a tecnologia do Web Service, bem como as principais funcionalidades exercidas por ele nos principais padrões existentes atualmente: Protocolo SOAP e Arquitetura REST.

Embora os Web Services já possuam alguns níveis de proteção de dados, entre as quais podemos citar a conexão ponto a ponto que garante a confidencialidade dos dados durante o transporte, ao passar por terminais intermediários, antes de chegar ao usuário final, a segurança já não é totalmente garantida (Pinho, 2008).

A seguir, são descritos alguns aspectos que contribuem para a formação da segurança fim a fim, a qual foi escolhida como forma mais eficaz para o projeto aqui proposto. Este contexto de segurança visa proteger a troca de mensagens entre clientes e servidores, cifrando a informação da origem até o destinatário, garantindo a integridade e autenticidade da mesma durante a transmissão.

3 ASPECTOS DE SEGURANÇA EM SISTEMAS WEB

O Capítulo que segue, contempla uma área da segurança, mais especificamente em sistemas web, apontando as principais fragilidades que atualmente a área enfrenta, em se tratando de proteção de dados e informações que trafegam diariamente em um ambiente aberto como a Internet.

Em contrapartida, são relacionadas também algumas das principais medidas que podem ser utilizadas como práticas de segurança para garantir a privacidade da comunicação entre sistemas web.

3.1 Sistema Web: O Lado da Vulnerabilidades

Os Web Services surgiram como elementos importantes para o sucesso do funcionamento da comunicação entre aplicações web, trazendo consigo uma ampla gama de vulnerabilidades de segurança que ainda estão sendo descobertas e exploradas a medida em que as novas formas de utilização dos Web Services são aprimoradas. Desse modo, investir em técnicas para segurança em sistemas web tornou-se uma das principais áreas de pesquisas no ambiente da tecnologia de informação (Prass, 2011).

Ainda que durante o processo de desenvolvimento de software, o atendimento aos requisitos que satisfaçam as necessidades do cliente seja o foco principal de um desenvolvedor, os critérios de segurança de tal aplicação devem ser identificados e incluídos na implementação do sistema, de modo que permita o alcance dos princípios de confidencialidade, integridade e disponibilidade.

Segundo Abreu (2011), entende-se por confidencialidade toda informação que pode ser acessada somente por usuários permitidos. Já integridade representa o ato de permitir que tais informações sejam alteradas somente por usuários autorizados. Disponibilidade consiste na garantia de que todas as informações estarão disponíveis para os usuários quando necessário.

De acordo com OWASP Top 10 (2013), que tem como princípio fundamental educar desenvolvedores, designers, arquitetos, gestores e organizações sobre a as consequências relacionadas às mais importantes vulnerabilidades de segurança em aplicações web, o cenário de ameaças para a segurança têm mudado constantemente. O lançamento de novas tecnologias

e a implantação de sistemas cada vez mais complexos são os principais fatores da evolução dessas vulnerabilidades. A última atualização da relação de vulnerabilidades divulgada apresenta a ordem descrita na tabela 1:

Tabela 1 - Top 10 Vulnerabilidades (OWASP Top 10, 2013)

Ataques	**Descrição**
A1 – Injeção	As falhas de Injeção, tais como injeção de SQL, de SO (Sistema Operacional) e de LDAP, ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados manipulados pelo atacante podem iludir o interpretador para que este execute comandos indesejados ou permita o acesso a dados não autorizados.
A2 – Quebra de autenticação e gerenciamento de sessão	As funções da aplicação relacionadas com autenticação e gerenciamento de sessão geralmente são implementadas de forma incorreta, permitindo que os atacantes comprometam senhas, chaves e tokens de sessão ou, ainda, explorem outra falha da implementação para assumir a identidade de outros usuários.
A3 – *Cross–Site Scripting* (XSS)	Falhas XSS ocorrem sempre que uma aplicação recebe dados não confiáveis e os envia ao navegador sem validação ou filtro adequados. XSS permite aos atacantes executarem scripts no navegador da vítima que podem "sequestrar" sessões do usuário, desfigurar sites, ou redirecionar o usuário para sites maliciosos.
A4 – Referência Insegura e Direta a Objetos	Uma referência insegura e direta a um objeto ocorre quando um programador expõe uma referência à implementação interna de um objeto, como um arquivo, diretório, ou registro da base de dados. Sem a verificação do controle de acesso ou outra proteção, os atacantes podem manipular estas referências para acessar dados não-autorizados.
A5– Configuração Incorreta de Segurança	Uma boa segurança exige a definição de uma configuração segura e implementada na aplicação, frameworks, servidor de aplicação, servidor web, banco de dados e plataforma. Todas essas configurações devem ser

	definidas, implementadas e mantidas, já que geralmente a configuração padrão é insegura. Adicionalmente, o software deve ser mantido atualizado.
A6- Exposição de dados sensíveis:	Muitas aplicações web não protegem devidamente os dados sensíveis, tais como cartões de crédito, IDs fiscais e credenciais de autenticação. Os atacantes podem roubar ou modificar esses dados desprotegidos com o propósito de realizar fraudes de cartões de crédito, roubo de identidade, ou outros crimes. Os dados sensíveis merecem proteção extra como criptografia no armazenamento ou em trânsito, bem como precauções especiais quando trafegadas pelo navegador.
A7 – Falta de função para Controle do Nível de Acesso	A maioria das aplicações web verificam os direitos de acesso em nível de função antes de tornar essa funcionalidade visível na interface do usuário. No entanto, as aplicações precisam executar as mesmas verificações de controle de acesso no servidor quando cada função é invocada. Se estas requisições não forem verificadas, os atacantes serão capazes de forjar as requisições, com o propósito de acessar a funcionalidade sem autorização adequada.
A8 – *Cross-Site Request Forgery* (CSRF)	Um ataque CSRF força a vítima que possui uma sessão ativa em um navegador a enviar uma requisição HTTP forjada, incluindo o cookie da sessão da vítima e qualquer outra informação de autenticação incluída na sessão, a uma aplicação web vulnerável. Esta falha permite ao atacante forçar o navegador da vítima a criar requisições que a aplicação vulnerável aceite como requisições legítimas realizadas pela vítima.
A9- Utilização de Componentes Vulneráveis Conhecidos:	Componentes, tais como bibliotecas, frameworks, e outros módulos de software quase sempre são executados com privilégios elevados. Se um componente vulnerável é explorado, um ataque pode causar sérias perdas de dados ou o comprometimento do servidor. As aplicações que utilizam componentes com vulnerabilidades conhecidas podem minar as suas defesas e permitir uma gama de possíveis ataques e impactos.
A10 – Redirecionamentos e	Aplicações web frequentemente redirecionam e encaminham usuários para outras páginas e sites, e usam dados não confiáveis para determinar as páginas de destino. Sem uma validação adequada, os atacantes

encaminhamentos inválidos	podem redirecionar as vítimas para sites de *phishing* ou *malware*, ou usar encaminhamentos para acessar páginas não autorizadas.

Assim, a fim de garantir que os princípios de segurança sejam aplicados para reduzir as vulnerabilidades advindas do processo de comunicação na web, diversas técnicas e algoritmos de segurança são implementadas para que as transmissões se tornem seguras e confiáveis. Dentre as técnicas mais utilizadas atualmente estão a criptografia, a assinatura e certificados digitais, protocolos de transmissão, entre outros.

3.2 Conceitos Básicos de Criptografia

Ao longo de toda a história cronológica da Criptografia, diversos povos têm procurado esconder informações através do uso da substituição de palavras por símbolos, números ou letras. Dentre as várias razões definidas para segurança da informação estavam a necessidade proteger as informações comerciais dos demais concorrentes (segredo chinês) e a necessidade de proteger os segredos militares (segredo alemão) através do uso da famosa máquina Enigma G (Freire, 2007).

De acordo com Burnett (2002), o principal objetivo da criptografia é manter a segurança das informações que trafegam em um canal de comunicações que está sujeito a interceptação por usuários não autorizados. Criptografia representa, portanto, uma ciência que utiliza funções matemáticas para gerar uma ou várias chaves combinadas por números, letras e/ou símbolos utilizados para criptografar um texto original legível. Este texto pode utilizar uma ou mais chaves para ser criptografado e a segurança desta criptografia está totalmente relacionada a complexidade da chave gerada. Esta técnica de segurança é capaz de permitir a transmissão de informação entre redes abertas, dificultando a interceptação destes por usuários não autorizados.

Atualmente, podemos definir como sendo legível algo “que pela clareza e nitidez caligráfica ou tipográfica, pode-se ler com facilidade” (recurso dicionário do Google). Assim, toda e qualquer forma de mascarar, um determinado dado ou informação, de modo que estes permaneçam impossibilitados de compreensão, podem ser considerados textos criptografados.

Segundo Trinta (1998), existem duas maneiras de criptografar: através de códigos ou cifras. A primeira maneira procura esconder o conteúdo da mensagem através de códigos pré-definidos entre as partes envolvidas. Já a cifra é uma técnica na qual o conteúdo da mensagem é cifrado através da transposição ou substituição das letras da mensagem original. O processo para decifrar as mensagens nesta técnica é realizado através da inversão do procedimento de cifração.

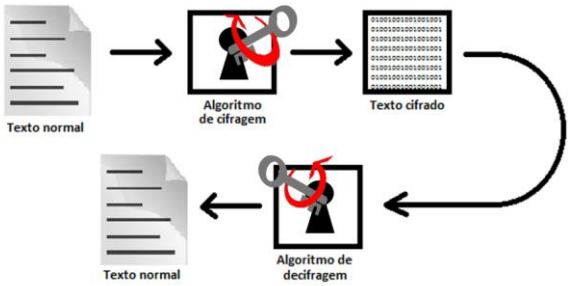
3.3 Classificação dos Algoritmos Criptográficos

Para garantir a proteção das informações, é possível utilizar outras vertentes na área da análise criptográfica as quais podemos citar a criptografia simétrica, assimétrica e as funções Hash.

3.3.1 Criptografia Simétrica

A criptografia simétrica, também chamada criptografia convencional, utiliza uma única chave para cifrar e decifrar uma mensagem. Na criptografia simétrica, cada algoritmo pode utiliza diferentes tamanhos de blocos, de chaves e de procedimentos para execução do método de cifração e decifração de dados (Saranya K et al, 2014). A figura 2 representa o processo da criptografia simétrica.

Figura 2 - Representação Criptografia Simétrica. Analogia baseada no livro Criptografia e segurança: O guia Oficial RSA (autoria própria, 2016)



A cifra de substituição é normalmente a mais utilizada entre os algoritmos desta vertente, por ser mais simples de manipular. Além disso, neste tipo de algoritmo, somente valores numéricos, letras do alfabeto e símbolos especiais são aceitos como textos legíveis, enquanto para o texto já cifrado são permitidos somente o alfabeto e/ou combinação de caracteres ou números (Saranya K et al, 2014).

Um dos grandes problemas da simetria é a distribuição e gerenciamento das chaves entre o emissor e o receptor, uma vez que este dever permanecer em segredo durante a transmissão entre as partes envolvidas, de modo que a chave não seja capturada por uma pessoa não autorizada.

Existem diversos algoritmos que representam a criptografia simétrica, entre os quais citamos DES, AES, IDEA, Twofish, RC6, CAST, a maioria destes não mais utilizados com frequência. De qualquer forma, o AES destaca-se entre esta relação, como sendo um dos principais algoritmos nesta área da criptografia.

3.3.1.1 Padrão de Criptografia Avançada – *Advanced Encryption Standard* (AES)

Este algoritmo criado em substituição ao DES, devido ao tamanho da chave do DES que era considerado muito pequeno suscetível, portanto, a quebra da chave. O AES é também uma cifra de blocos que possuem tamanho fixo de 128 bits, chamados de *State*, os quais são organizados em uma matriz 4x4. O tamanho das chaves criptográficas pode variar entre 128, 192 e até 256 bits para cifrar e decifrar informações (Stallings, 2011).

De acordo com Feldhofer et al. (2004) e Hodjat et al. (2005), para a implementação do AES a mesma função criptográfica é aplicada sobre cada bloco através de iterações que variam entre 10, 12 ou 14 vezes por bloco, as quais alteram a matriz *State* por meio de transformações não lineares, lineares e baseadas em chave. Cada bloco altera para outro de mesmo tamanho e cada byte da matriz *State* são afetados pelos seguintes procedimentos:

- *SubBytes:* Momento em que ocorre a transformação dos bytes do estado (blocos de dados de entrada) por bytes de S-Box (tabelas de substituição estática).
- *ShiftRows:* Ocorre a rotação de forma cíclica das linhas do estado para a segunda em uma casa, a terceira em 2 casas e a quarta em 3 casas.
- *MixColumns*: Nesta operação, acontece a transformação dos dados das colunas do estado multiplicando por um polinômio irredutível fixado.
- *AddRoundKey:* "Embaralha" as colunas com uma das chaves geradas na rotina de expansão.

No procedimento de decodificação de informação, transformações equivalentes ao processo de codificação podem ser realizadas a fim de chegar a informação decifrada, bastando realizar, por exemplo, o processo de *SubBytes* inverso, *ShiftRows* inverso, etc.

Advanced Encryption Standard (AES) é uma cifra de bloco que já foi amplamente analisada para garantir a segurança contra vários ataques existentes. No entanto, este tipo de criptografia exige uma quantidade significativa de energia e de recursos. Desse modo, para alguns casos como dispositivos RFID, isto é, tecnologias que utilizam a radio frequência para captura de dados, utilizar algoritmos criptográficos tradicionais que exigem grande capacidade de memória e poder de processamento torna-se inviável, já que estes tipos de dispositivos dispõem de recursos limitados (Arora, 2012).

Assim, para atender a necessidade de dispositivos que dispõem de recursos limitados para grandes processamentos, uma série de cifras de blocos leves têm sido propostas, as quais são definidas a seguir.

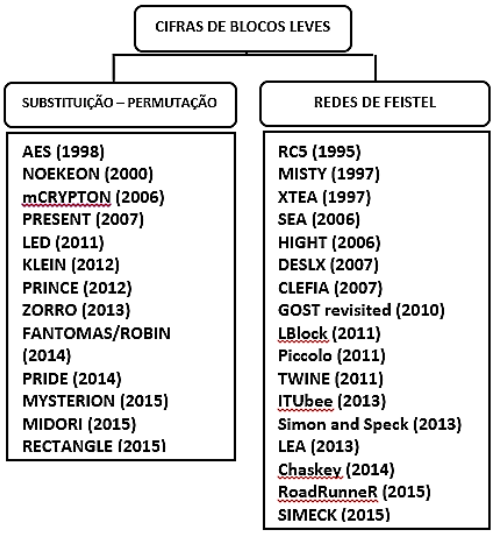
3.3.1.2 Cifras de Blocos Leves

A fim de abordar uma área da criptografia que abrangesse algumas necessidades da computação mais restritas, as quais o baixo consumo de energia e limitações da área de hardware são requisitos para implementação do algoritmo proposto, surge as cifras de blocos

leves (*lightweight block ciphers*), normalmente utilizadas em aplicações que exigem uma combinação ideal entre eficiência computacional e segurança para uma boa transmissão de informações (Arora et al, 2012).

A figura 3 ilustra algoritmos classificados como cifras de blocos leves. Estas cifras atuam em blocos de tamanho fixo, com variação de tamanhos entre 32, 64 e 128 bits e são categorizadas em dois subgrupos: Redes Substituição – Permutação e Redes de Feistel.

Figura 3 - Cifras de Blocos Leves (Costa et al., 2016)

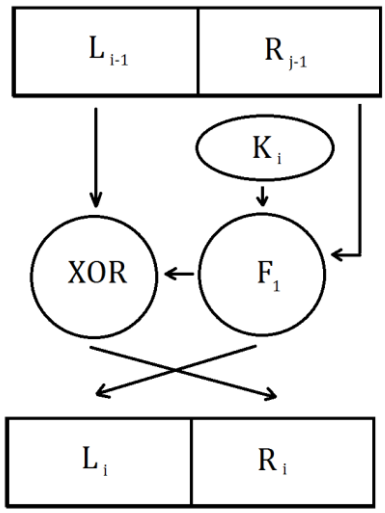


As cifras de blocos são uma função que opera sobre blocos de bits de tamanho fixo, transformando um bloco de texto não cifrado em um bloco de texto cifrado, através uma chave simétrica de tamanho k-bits.

A rede de Substituição – Permutação visa proporcionar tanto a confusão quanto a difusão da mensagem, utilizando operações distintas. Na etapa de "confusão" a relação entre texto não cifrado, chave e texto cifrado torna-se mais complexa, de modo que seja difícil descobrir as propriedades da chave k a partir do texto cifrado. Já na etapa "difusão", os bits do texto legível são embaralhados para que as redundâncias sejam eliminadas no texto cifrado (Biryukov et al, 2016).

Na rede Feistel, ilustrada na figura 4, a estrutura ordena a saída de texto cifrado como *Right-Round* e *Left-Round*. Os blocos são divididos em duas partes iguais (*Left e Right*) e para gerar o texto cifrado, as duas partes do bloco sofrem processamento de n iterações. Cada iteração recebe Left-$_{i-1}$ e Right $_{i-1}$, as quais são derivadas da iteração anterior. Através de um algoritmo gerador de chaves, sub chaves distintas também são geradas e nomeadas como K_n. A cada *round* (rodada) são realizadas operações de XOR, bit a bit. A decodificação é realizada usando o mesmo processo de *Right-Round* e *Left-Round*, mas com sub chaves utilizadas na ordem inversa (Costa et al, 2016).

Figura 4 - Processo de Iteração da Rede Feistel (autoria própria, 2016)

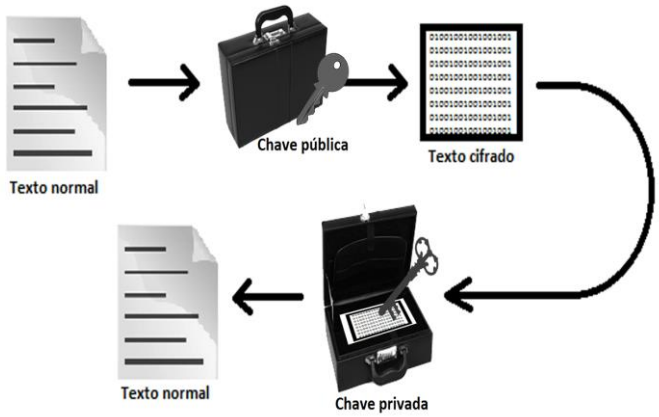


Assim, ao escolher algoritmos de segurança para implementação em aplicações, os custos devem ser levados em conta de forma a satisfazer as necessidades de eficiência computacional e segurança na transmissão fim a fim, as quais os algoritmos de chave simétricas, especialmente as cifras de blocos leves desempenham importante papel para garantir a confidencialidade nas referidas aplicações.

3.3.2 Criptografia Assimétrica

A criptografia assimétrica precisa de um par de chaves para efetuar o procedimento de codificação e decodificação de informações. Nesse método uma chave deve ser criada para codificar a informação, a qual denominamos chave pública, e outra chave é criada para decodificar a informação, chamada chave privada. A figura 5 ilustra o processo realizado para obtençãoo de criptografia assimétrica.

Figura 5 - Representação Criptografia Assimétrica (autoria própria, 2016)



Na criptografia assimétrica o algoritmo é uma função de via única, onde a chave trata todos os dados como números e neles realiza operações matemáticas, ao contrário do que ocorre com a criptografia simétrica que opera nos dados apenas como bits e realiza operações do computador para criptografar determinado dado (Burnett,2002).

Os algoritmos RSA, ElGamal e Curvas Elípticas são alguns exemplos de criptografias que utilizam o modelo assimétrico em sua estrutura. No entanto, o mais utilizado atualmente é o RSA.

3.3.2.1 Algoritmo RSA

O algoritmo Rivest-Shamir-Adleman (RSA), foi um dos sucessos descobertos no desafio desenvolvido pela MIT (*Massachussets Institute of Technology*). O esquema RSA foi inicialmente implementado com o propósito geral de estabelecer uma abordagem para criptografia de chave pública (Burnett, 2002).

Este esquema pertence ao grupo dos algoritmos de bloco de cifras, nos quais o texto original e o texto cifrado são representados por números inteiros entre 0 e n – 1 para qualquer n. O tamanho normal para n varia entre 1024 bit ou 309 dígitos decimais. Assim, o RSA faz uso

de uma expressão com exponenciais. O texto original é cifrado em blocos e cada bloco possui um valor binário menor que o determinado número n. Desse modo, o tamanho do bloco deve ser menor ou igual a $\log_2 (n) + 1$ ou ainda, o tamanho do bloco pode ser representado como i bits, onde $2^i < n <= 2^{i+1}$. Para criptografar textos originais, são utilizadas as seguintes notações:

C (texto cifrado) = M^e mod n

M (texto original) = M^{ed} mod n

De acordo com a notação, o emissor e o receptor devem conhecer o valor de n. O emissor conhece o valor de e, enquanto o receptor conhece somente o valor de d. Dessa forma, é gerado um algoritmo de criptografia pública com chave pública PU ={e,n} e chave privada PR{d,n}.

De forma mais detalhada a figura 6 ilustra os passos necessários para a realização do cálculo do RSA:

• Escolher dois números primos (p,q);

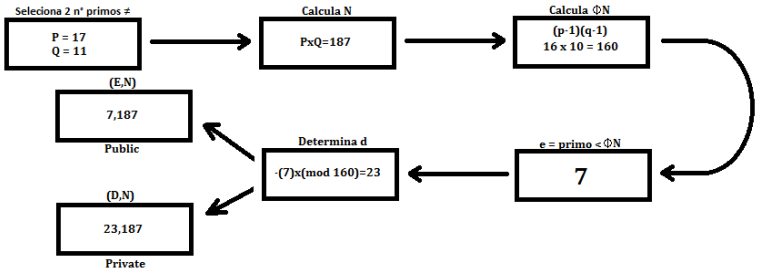
• Calcular n = pq;

• Calcular f(n)=(p-1)(q-1)

• Selecionar e que seja primo para f (n) = 160 e menor que f (n);

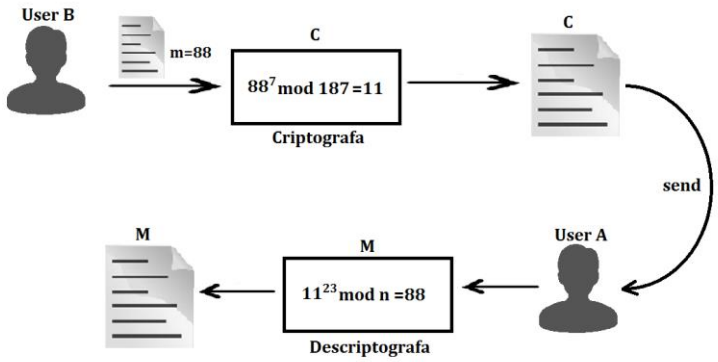
• Determinar d tal que de = 1(mod160) e *d*<160.

Figura 6 - Criando Chaves Pública/Privada RSA (autoria própria, 2016)



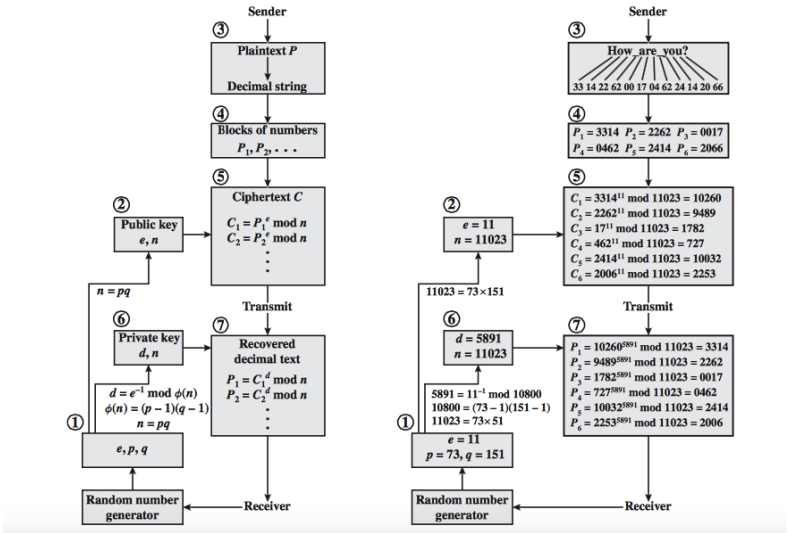
Assim, de acordo com a figura 7, supondo que o usuário A publicou uma chave pública e o usuário B deseja enviar uma mensagem M para A. Para este procedimento, é necessário que B calcule C = M^e mod n e transmita C. No recebimento desta mensagem cifrada, o usuário A decifra-a calculando M = M^{ed} mod n.

Figura 7- Utilizando RSA para Criptografar e Descriptografar Mensagem



Já em um processo de múltiplos blocos de dados, o RSA trata cada bloco do texto original com um código único de quatro dígitos decimais ou dois caracteres alfanuméricos. A figura 8 ilustra a sequência de eventos que criptografa a mensagem de múltiplos blocos, baseado no livro *Cryptography and Network Security Principles and Practice 5th Edition* (Stallings, 2011).

Figura 8 - Representação de Criptografia de Mensagens em Blocos Múltiplos (Stallings,2011)

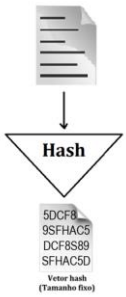


3.3.3 Função HASH

Uma função de Hash é uma função matemática que transforma uma mensagem (cifrada ou não) com entrada de comprimento finito em uma saída de tamanho fixo, mais especificamente um código denominado Hash. Este código tem a responsabilidade de identificar uma informação de maneira única, onde qualquer alteração realizada nesta informação pode causar variação no resultado do valor do Hash (Marques, 2011). Assim, funções Hash são operações unidirecionais, as quais não permite obter informações a partir de um valor de Hash gerado, conforme ilustrado na figura 9.

Figura 9- Processo de Geração de HASH

(autoria própria, 2016)



De acordo com Alfred J. Menezes et al (1996), funções do tipo Hash são comumente utilizadas em assinaturas digitais, para garantir a integridade dos dados. Quando a mensagem é longa, geralmente a mesma é criptografada utilizando uma função Hash pública, no entanto, somente o valor Hash é assinado. A mensagem é enviada ao receptor e este, por sua vez, deve utilizar a função Hash para descobrir o valor do Hash para esta mensagem e se a assinatura recebida está correta para este valor. Dessa forma, economiza-se tempo e espaço quando comparado com assinaturas de mensagens diretas, que normalmente envolvem a divisão da mensagem em blocos de tamanhos adequados e assinatura de cada um deles, individualmente.

Funções Hash podem ser utilizadas para a integridade de dados pois o Hash de valor correspondente a uma determinada entrada é um cálculo. A qualquer momento é possível verificar se os dados não foram alterados, recalculando o valor de Hash e comparando com o valor original. Isso porque, de acordo com as características do algoritmo da função de Hash, é computacionalmente inviável encontrar um objeto de dados com um resultado Hash pré-especificado, ou ainda, encontrar dois objetos de dados que mapeiam o mesmo resultado Hash.

Funções Hash são normalmente utilizadas em aplicações de segurança e protocolos de Internet, as quais podemos citar:

Autenticação de Mensagem: Utilizada para verificar integridade de uma mensagem, de modo que os dados recebidos estejam exatamente como foram enviados (sem modificações, inserções ou exclusões), assegurando a validade da identidade do remetente. Neste caso, o valor da função Hash é utilizado como um resumo da mensagem.

Assinatura Digital: O valor de Hash é utilizado juntamente com uma chave privada.

Qualquer pessoa que conheça a chave pública poderá verificar a integridade da mensagem que estiver associada com aquela assinatura digital. Assim, para alterar a mensagem, o interceptador deverá conhecer a chave pública.

Arquivo de senha unidirecional: Funções de Hash são comumente utilizadas para armazenamento de um valor de Hash para uma senha, em vez de uma palavra-passe em si. Desse modo, o invasor que obtiver acesso ao arquivo de senhas não conseguirá recuperar as senhas salvas. Basicamente, nesta aplicação, quando um usuário digita a senha, o Hash da senha é comparado com o valor Hash armazenado para verificação, não havendo, portanto, dados arquivados que revelem a senha original do usuário (Stallings, 2011).

Os algoritmos de Hash utilizados com mais frequências nas aplicações e protocolos de Internet são *Message-Digest Algorithm* 5 (MD5) e o *Secure Hash Algorithm* (SHA). O MD5 é uma função de 128 bits desenvolvido pelos mesmos autores do RSA Data Security muito utilizado por softwares ponto-a-ponto na verificação de integridade de arquivos e logins, enquanto o SHA é bastante aplicado em aplicações que exijam alta segurança da integridade (Marques, 2011). O SHA-3 foi o último padrão de Hash selecionado durante uma competição organizada pela *National Institute of Standards and Technology* (NIST), a fim de aprimorar o nível de segurança da família SHA. O algoritmo Keccak foi escolhido como vencedor da competição.

3.3.3.1 Keccak (SHA – 3)

O Keccak pertence a família das funções esponja. A função esponja é uma generalização do conceito de função Hash criptográfica que utiliza a construção em esponja para a permutação de tamanho fixo para uma entrada de qualquer tamanho que gere saídas de tamanhos arbitrários (Souza, 2013, apud Daemen et al., 2009).

Este algoritmo consiste em duas etapas: a função Keccak-f[b] (A,RC), que realiza substituições e operações lógicas sobre os dados e a função de esponja Keccak[r,c](M), que organiza e prepara os dados de entrada para realizar a manipulação desses dados pela função de permutação e arranja os valores de saída para gerar o Hash (Souza, 2013).

No item seguinte, algumas APIs de segurança são apresentadas de modo a acrescentar outras maneiras de proteção de dados já existentes.

3.4 APIs de Segurança Prontas

Logo, o conjunto de todos os tipos de algoritmos criptográficos existentes como principal técnica de proteção de informações deram origem a diversas APIs de segurança, as quais são divididas e implementadas de acordo com a tecnologia utilizada. Para Java, por exemplo, *Bouncy Castle* compõe uma legião de APIs seguras que implementam algoritmos criptográficos, enquanto em PHP, há diversas classes nativas de segurança, bem como classes externas criptográficas que executam algoritmos como RSA, Rijndael, AES, RC4, entre outros.

As figuras 10, 11, 12 e 13 mostram os exemplos de alguns algoritmos retirados da PHP *Security Library*, implementados em linguagem simplificada para melhor compreensão.

Figura 10 - Código-Fonte RC4 *(PHP Security Library)*

```
incluir(Crypt/RC4);
objeto rc4;
rc4 = novo Crypt_RC4();
rc4.setarChave('abcdefghijklmnopqrstuvxyz');

inteiro tamanho = 10*1024;
string texto = '';

para(inteiro i = 0; i<tamanho; i = i+1)
{
    texto = texto+'a';
}

escreva 'criptografado:' + rc4.criptografar(texto);
escreva 'descriptografado:' + rc4.descriptografar(texto);
```

Figura 11 - Código-Fonte HASH *(PHP Security Library)*

```
incluir(Crypt/Hash);
objeto hash;
hash = novo Crypt_Hash('sha1');
string texto = 'abcdefg';
texto = binario_para_hexadecimal(texto);
texto = hash.hash(texto);
escreva 'criptografado:' + rc4.criptografar(texto);
```

Figura 12 - Código – Fonte AES *(PHP Security Library)*

```
incluir(Crypt/AES);
objeto aes;
aes = novo Crypt_AES();
aes.setarChave('abcdefghijklmnop');

inteiro tamanho = 10*1024;
string texto = '';

para(inteiro i = 0; i<tamanho; i = i+1)
{
    texto = texto+'a';
}

escreva 'criptografado:' + aes.criptografar(texto);
escreva 'descriptografado:' + aes.descriptografar(texto);
```

Figura 13 - Código - Fonte RSA *(PHP Security Library)*

```
incluir(Crypt/RSA);
objeto rsa;
rsa = novo Crypt_RSA();
extrair(rsa.criarChave());
string texto = 'texto que será criptografado';
string chavePublica = rsa.pegarChave('publica');
string chavePrivada = rsa.pegarChave('privada');
rsa.carregarChavePublica(chavePublica);
rsa.carregarChavePrivada(chavePrivada);
escreva 'criptografado:' + rsa.criptografar(texto);
escreva 'descriptografado:' + rsa.descriptografar(texto);
```

As características destes algoritmos podem ser implementadas tanto através de extensões opcionais existentes em classes nativas da linguagem trabalhada, como podem também ser executadas por meio de bibliotecas externas, inseridas dentro da aplicação como uma extensão da classe, usando para isso, estruturas semelhantes às apresentadas nos exemplos de códigos anteriores.

3.5 Considerações Finais do Capítulo

Durante o desenvolvimento deste capítulo foram evidenciadas as vulnerabilidades que mais afetam os sistemas web atualmente, entre os quais estão as falhas na injeção de SQL, quebra de autenticação e gerenciamento de sessão, exposição de dados ~~sensíveis~~ de maneira incorreta, etc.

Além disso, foram exploradas também as principais tecnologias de proteção que podem ser utilizadas para que o nível de insegurança na rede aberta, no caso a Internet, seja diminuído, dispondo das criptografias apresentadas como meio para alcance do nível de privacidade desejado.

Na sequência, servindo-se com as informações adquiridas até o momento sobre Web Service e criptografia, é apresentada uma arquitetura referente a visão geral que abrange o objetivo do projeto aqui proposto, unindo os conceitos de comunicação de Web Service e cliente com a garantia de segurança por meio de criptografia implementada em um contexto fim a fim.

4 ARQUITETURA GERAL DA API

Neste capítulo são relacionados os trabalhos correlatos que fazem referência a ideia deste projeto, bem como os componentes e o funcionamento da arquitetura proposta para o cenário de comunicação entre cliente e Web Services.

4.1 Trabalhos Correlatos

Qualquer comunicação entre Web Services é realizada por meio de troca de informações utilizando como formato principal, o XML. No entanto, alguns protocolos de Web Services, como o SOAP, não implementam segurança por questões de portabilidade. Assim, se faz necessário que a proteção da comunicação seja realizada, também, em nível de aplicação por meio de configurações personalizadas e utilização de alguns padrões de segurança (Rodrigues, 2011).

Levando isso em consideração, o trabalho de Rodrigues (2011) apresenta uma avaliação e comparação de adequações técnicas que permitam a criação de Web Services seguros, além de validar se os serviços de uma aplicação possuem características desejadas relacionadas ao desempenho e segurança dos mesmos. No projeto, o autor determina também "quais algoritmos criptográficos impõe menor impacto em termos de desempenho no contexto de Web Services". Estes algoritmos ou a combinação destes, visam garantir a segurança fim-a-fim e com desempenho aceitável.

Em Chiaramonte et al (2006), os autores abordam um sistema focado em oportunidades de otimização de conexões seguras em ambientes dinâmicos e, através de alguns parâmetros extraídos de cada cenário de comunicação é definido o algoritmo ou conjunto de algoritmos, bem como o protocolo mais indicado para preservar as informações trafegadas pela rede naquele ambiente, naquele momento. Dentre as principais características avaliadas para escolher os algoritmos e chaves adequadas para criação de conexão segura estão: desempenho do algoritmo/chave, nível de segurança desejado, velocidade da rede utilizada, tempo de processamento (Chiaramonte et al, 2006). Tais características motivaram o levantamento dos parâmetros adaptados para esta dissertação.

Ambas correlações apontam estudos e análises relacionadas à segurança da informação trafegada entre aplicações, no entanto, nenhum deles apresenta uma implementação automatizada de uma API de seleção de criptografias baseada em comunicação entre cliente e Web Services qual é proposta aqui, focando em uma arquitetura que pretende unir a segurança que a criptografia promove, junto ao fluxo de execução da informação que parte do cliente e percorre um ambiente suscetível a vulnerabilidades para chegar ao serviço web.

4.2 Desenvolvimento e Implementação da Arquitetura

Por se tratar de uma tecnologia independente de plataforma e linguagem de programação, os Web Services deixam a desejar com relação as ameaças de segurança as quais estão suscetíveis durante a troca de informações. Dessa forma, o desafio em prover segurança na comunicação entre cliente e Web Services consiste em analisar e compreender os riscos atuais e contorna-los utilizando técnicas de criptografia.

Para desenvolvimento da arquitetura do projeto foi necessário definir os principais métodos de comunicação utilizados por um Web Service para envio e requisição de informações. A partir daí o desenvolvimento da API proposta, conta com a inclusão da reescrita destes métodos, acrescentando os parâmetros que definirão cada cenário de avaliação, quando estes necessitarem da utilização da API para a tomada de decisões referente a criptografia adequada para cada comunicação estabelecida.

Na figura 14 são apresentados os componentes propostos para esta API em um modelo geral de arquitetura. Por meio desta representação, são destacados alguns métodos que foram reescritos, bem como a extensão da API para bibliotecas criptográficas, mais especificamente a PHPSECLIB, que são compostas por alguns algoritmos implementados no projeto.

Figura 14- Modelo de Arquitetura da API (autoria própria, 2016)

- **A) Algoritmos Criptográficos:** Um conjunto de algoritmos, selecionados de acordo com a classificação realizada no estudo das criptografias mais utilizadas para cifrar informações durante a troca de informações. No modelo consta exemplos de algoritmos simétricos, assimétricos, cifras de blocos leves, além de citar a família SHA, bastante utilizada também em casos onde há necessidade de criação de assinaturas digitais.
- **B) Biblioteca Criptográfica:** Um conjunto de bibliotecas que apresentam classes específicas, que visam oferecer a segurança necessária para determinada situação de transmissão de dados. Foram citados exemplos de bibliotecas utilizadas em Java e, principalmente em PHP, linguagem a qual o projeto foi direcionado. A partir dessas bibliotecas, foi feita a extensão de alguns algoritmos que complementam o nível de segurança criado pela API desenvolvida.
- **C) Protocolo/Arquitetura de Web Service REST:** A partir da definição do REST como estilo de arquitetura de Web Service utilizado no projeto, foram selecionadas algumas micro-frameworks permitidas para utilização neste estilo de arquitetura para que alguns métodos destas fossem reescritos e incorporados à API proposta, incluindo novos parâmetros.
- **D) Classes de criação do cenário:** Determinadas classes foram implementadas de modo a permitir a configuração e parametrização nas chamadas realizadas pelo cliente e pela

API. Para isso o ambiente é identificado e caracterizado como um cenário específico, para que então, haja a tomada de decisão sobre qual algoritmo será utilizado para cifrar informação. Para tanto, se faz necessário que o cliente encaminhe a requisição, de acordo com os parâmetros exigidos, classificando o ambiente com precisão. A tabela 2 apresenta os parâmetros preliminares para identificação do cenário.

Tabela 2- - Parâmetros para Identificação de Cenários (autoria própria, 2016)

Parâmetros para Definição de Cenário	**Descrição**
Volume de dados a ser cifrado (cliente / serviço web)	Cenários críticos que exigem espaço considerável no local de destino para ser armazenado.
Desempenho do algoritmo	Cenários que exigem processamento tempo real.
Nível de Segurança desejado	Cenários que necessitam de muita segurança durante o envio e armazenamento das informações.
Velocidade da rede utilizada	Cenários que em que a velocidade da rede é limitada ou não.
Tempo de processamento nas estações Cliente / serviço web	Cenários em que o tempo de processamento em cada computador no cliente é uma variável, além de verificar o mesmo tempo de processamento gasto no serviço web.
Frequência de requisição informação cifrada	Se a frequência da requisição de informação for constante ou não, deve ser informado.
Deseja garantir integridade dos dados?	Cenários em que as informações não podem sofrer alterações e/ou modificações.

Documento devera ser assinado digitalmente?	Casos de cenários que necessitem o documento enviado necessite de uma validade digital.
Tipo de arquivo cifrado	O tipo de arquivo pode demandar um algoritmo criptográfico diferente para o cenário.

- **E) Classes de seleção de algoritmos criptográficos:** Conta com métodos para classificação do algoritmo a ser escolhido para criptografar a informação enviada pela aplicação (cliente), estendendo as classes de criação do cenário. Os parâmetros são recebidos nos métodos de criação de cenário, juntamente com a informação que o cliente deseja enviar para o Web Service. Com cenário identificado na etapa anterior, o mesmo é utilizado para escolher o algoritmo mais adequado para cifrar a informação recebida, enviando para a próxima etapa, **Classes de response e request seguras para REST.**
- **F) Classes de response e request seguras para REST:** Nesta etapa, alguns métodos do micro-framework SLIM foram reescritos, de modo a inserir a informação já criptografada, para envio ao cliente.

4.3 Considerações Finais do Capítulo

Nesta seção foram relacionados os principais trabalhos correlatos à proposta desta monografia, além de descrever os principais componentes do modelo de arquitetura a ser implementada para a construção da API. A arquitetura contempla uma técnica de construção em blocos, de modo que o desenvolvimento seja facilitado e conexo.

5 Implementação da API

A arquitetura projetada no capítulo anterior deu origem a uma implementação em linguagem PHP, na qual validou-se as funcionalidades propostas no modelo sugerido. Assim, este capítulo apresenta os principais detalhes deste desenvolvimento, bem como o embasamento teórico de um estudo de caso para Web Services REST.

5.1 Codificação do Modelo de API

Durante a etapa de codificação, optou-se por utilizar a linguagem PHP, versão 5.1, e arquitetura MVC (Model - View - Controller) como forma de garantir as boas práticas de programação em uma estrutura organizada, bem como um *framework* com adaptações próprias, unindo as qualidades dos *frameworks* SLIM e Laravel e mantendo uma estrutura mais leve para implementação do código.

A divisão da arquitetura em blocos, conforme modelado no capítulo 4, permitiu que o desenvolvimento da implementação fosse realizado em etapas e sem a necessidade de seguir a sequência proposta no desenho modelo. Isso porque, a criação das classes foi projetada de modo a serem independentes, mas estendidas entre si. Dessa forma, iniciou-se o código pela etapa de parametrização do ambiente, no qual são relacionados parâmetros fixos e dinâmicos do ambiente do cliente. As informações recebidas por esta classe, são salvas em banco de dados para posterior utilização na etapa de seleção do algoritmo.

5.1.1 Etapa de Criação do Cenário

Nesta etapa, o projeto recebe as primeiras classes responsáveis por garantir que os parâmetros de definição do ambiente sejam solicitados ao cliente.

Nesta etapa, quanto mais detalhada for a classificação do ambiente, mais precisa será a tomada de decisão na escolha do algoritmo adequado para determinados cenários. Isso porque,

a avaliação de determinados parâmetros relacionados na tabela 2, podem direcionar a escolha do algoritmo com relação ao desempenho ou segurança que estes apresentarem. No entanto, optou-se por priorizar os seguintes parâmetros:

Fixos (em toda requisição, o cliente enviará a mesma informação):

• Token de identificação do cliente.

• ID de identificação do cliente.

Dinâmicos (a cada nova requisição, os parâmetros podem variar):

• Volume de dados, escolhido por ser um fator determinante quando criptografa-se dados.

• Tempo médio de processamento dos algoritmos, outro fator essencial para determinados cenários.

• Nível de segurança, definidos com base na literatura dos algoritmos.

Cada algoritmo implementado passou por uma classificação de nível de segurança, baseado na literatura destes. Isto é, foi realizada uma criptoanálise dos algoritmos selecionados para definir qual o nível de segurança cada algoritmo poderia receber, sendo 1 para o algoritmo mais seguro (ainda não sofreu ataques publicados oficialmente), 2 para algoritmos que foram parcialmente quebrados (apenas algumas rodadas) mas ainda são utilizados como criptografia segura e 3 para o algoritmo menos seguro (já foi quebrado por meio de ataques). Conforme a tabela 3, é possível observar os principais pontos responsáveis por complementar a escolha do nível de segurança para cada algoritmo implementado.

Tabela 3 - Tabela de Nivel de Segurança (autoria própria, 2016)

Algoritmos	**Data de criação**	**Ataques**	**NIVEL**
AES	2002	Não publicado oficialmente	1
DES	1976	Ataque de força bruta, 1997 - (Vogt, 2003)	3
3DES	1993	Não publicado oficialmente	2
SHA1	1995	Ataque de colisão, 2005 - (Sotirov et al,2008)	3
SHA256	2001	Ataques de colisão, 2008 - (Sotirov et al,2008)	2
MD5	1991	Ataque de colisão, 2004 - (Sotirov et al,2008)	3
RSA	1978	Chave 512 bits fatorada, 1999 - (Valenta et al, 2015)	1
ECC	1985	Birthday Attack, 2003 - (Sangalli, 2012)	1
SIMON	2013	Criptoanálise diferencial em algumas rodadas (Abed et al, 2014)	2
SPECK	2013	Criptoanálise diferencial em algumas rodadas(Abed et al, 2014)	2
SIMECK	2015	Não publicado oficialmente	1

Assim, de acordo com Vogt (2003), o Projeto DESCHALL por meio do DES Challenge, conseguiu quebrar o algoritmo DES, através de um ataque de força bruta e, por este motivo, o algoritmo recebe o nível 3 como qualificação da sua segurança. Outros algoritmos que receberam o nível 3 de segurança são MD5 e SHA-1, os dois primeiros quebrados em sua forma

completa, por meio de ataques de colisão, onde um conjunto de dados possuem o mesmo valor de Hash.

Receberam o nível 2 de qualificação de segurança os algoritmos 3DES, SHA256, SIMON e SPECK por apresentarem relatos de ataques contra algumas rodadas dos algoritmos (Ataque diferencial e Ataque de colisão), ou por ainda serem utilizados em diversos cenários que necessitam de segurança criptográfica implementada, como é o caso do 3DES empregado em aplicações financeiras (Corrêa,2009) e o SHA256, que ainda é utilizado em protocolos de segurança como TLS, SSL e Ipsec.

O AES, ECC e o RSA embora tenham algumas teorias sobre possíveis ataques contra os mesmos, ainda assim, não há publicações oficiais referentes a quebra dos algoritmos em sua forma completa e, portanto, receberam nível 1 na classificação. O AES, por exemplo, continua sendo utilizado em *Smart Cards* e outros equipamentos que utilizam pouca memória RAM e necessitam poucos ciclos de processamento (Rinaldi, 2012). O RSA, por sua vez, ainda que tenha sido alvo de quebra de chave de 512 bits, possui uma significativa margem de segurança, se utilizada chaves com maior número de bits, perdendo apenas para o quesito de tempo de processamento, caso este seja um fator decisivo durante a utilização da criptografia para segurança de dados. Já o ECC, embora quebrado por uma chave de 109 bits, a sua segurança ainda é equivalente ao RSA ao usar chaves maiores de 160 bits. Dessa maneira, o ECC ainda compete com o RSA no quesito segurança, por apresentar tamanhos de chaves reduzido em comparação com RSA, que utiliza chaves relativamente grandes para adquirir uma segurança aceitável (Stallings, 2011).

5.1.2 Etapa de Seleção do Algoritmo Criptográfico

Nesta etapa, foi realizado a estrutura da classe de seleção do algoritmo, na qual os processamentos para escolha do algoritmo mais adequado são executados. Para a criação da tabela condicional que processa a tomada de decisão baseada na resposta dos parâmetros enviados pelo cliente, foi necessário atribuir pesos de 1(um) a 7(sete) para as faixas de tamanho e tempo de processamento, de modo que pudessem ser utilizadas de maneira equivalente dentro da tabela condicional.

Tabela 4 - Faixas de tamanho de arquivo em megabytes (autoria própria, 2016)

Pesos	Faixas de Tamanho (MB)	Faixas de tempo de processamento (seg)
1	0 - 9.99	0 – 10
2	10 – 19.99	10.01 – 20
3	20 – 29.99	20.01 – 30
4	30 – 39.99	30.01 – 40
5	40 – 49.99	40.01 – 50
6	50 – 59.99	50.01 – 60
7	60 ou mais	60.01 ou mais

A tabela 5 foi criada utilizando os pesos atribuídos a cada faixa de tamanho e tempo e processamento. Assim, cada algoritmo recebeu um peso referente ao tempo de processamento gasto durante a criptografia de uma mensagem com diversos tamanhos.

Tabela 5 - Tomada de Decisão (autoria própria, 2016)

Tamanho (pesos)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Algoritmos	**Tempo de processamento (pesos)**						
AES_128	1	1	2	3	4	5	5
AES_192	1	2	3	4	5	6	6
AES_256	1	2	3	4	6	7	7
MD5	1	1	1	1	1	1	1
3DES	1	1	2	2	3	4	4
RSA	5	7	7	7	7	7	7
ECC	7	7	7	7	7	7	7
SHA1	1	1	1	1	1	1	1
SHA256	1	1	1	1	1	1	1
SHA512	1	1	1	1	1	1	1
SIMON	1	1	1	2	2	2	2
SPECK	1	1	1	1	2	2	2
DES	1	1	2	2	3	3	3
SIMECK	1	1	1	1	1	1	1

Assim, por meio da tabela condicional é realizada a tomada de decisão sobre a escolha do algoritmo mais adequado para determinado cenário, executando os seguintes passos:

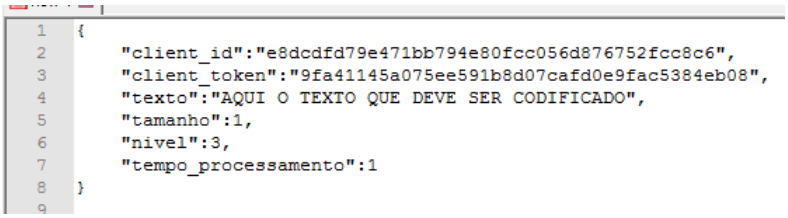
Passo 1 – Realiza-se uma busca pelos algoritmos que estão de acordo com o tamanho e o tempo de processamento enviados por meio dos parâmetros do cliente.

Passo 2 - Os algoritmos selecionados no passo 1, consulta a tabela de níveis de segurança, verificando quais destes estão em conformidade com o nível solicitado pelo cliente.

Passo 3 - Se houver algoritmo com mesmo nível solicitado pelo cliente, este é escolhido para a criptografia da mensagem. Caso contrário, a classe implementada selecionará o algoritmo com maior nível definido, dentre os estabelecidos no passo 2.

Como um exemplo, suponhamos que o cliente tenha enviado as seguintes informações:

Figura 15 - JSON do Cliente (autoria própria, 2016)



```
{
    "client_id":"e8dcdfd79e471bb794e80fcc056d876752fcc8c6",
    "client_token":"9fa41145a075ee591b8d07cafd0e9fac5384eb08",
    "texto":"AQUI O TEXTO QUE DEVE SER CODIFICADO",
    "tamanho":1,
    "nivel":3,
    "tempo_processamento":1
}
```


Desse modo, baseado na tabela de apoio (tabela 5), o tamanho da mensagem do cliente é definido pelo peso 1 (faixa de 0 a 9.99 MB) e o tempo de processamento desejado para criptografia/descriptografia da informação possui tempo de processamento 1 (faixa de 0 a 10 segundos).

A partir dessas informações, a tabela 6 é consultada e informa que os algoritmos AES_128, AES_192, AES_256, MD5,3DES, SHA1, SHA256, SHA512, SIMON, SPECK, DES e SIMECK possuem as características de tamanho e tempo definidas pelo cliente.

Tabela 6 - Consulta Exemplo na Tabela condicional (autoria própria, 2016)

Tamanho (pesos)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Algoritmos	**Tempo de processamento (pesos)**						
AES_128	1	1	2	3	4	5	5
AES_192	1	2	3	4	5	6	6
AES_256	1	2	3	4	6	7	7
MD5	1	1	1	1	1	1	1
3DES	1	1	2	2	3	4	4
RSA	5	7	7	7	7	7	7
ECC	7	7	7	7	7	7	7
SHA1	1	1	1	1	1	1	1
SHA256	1	1	1	1	1	1	1
SHA512	1	1	1	1	1	1	1
SIMON	1	1	1	2	2	2	2
SPECK	1	1	1	1	2	2	2
DES	1	1	2	2	3	3	3
SIMECK	1	1	1	1	1	1	1

Na sequência, os algoritmos selecionados consultam a tabela de níveis (tabela 3) para conferir qual destes fará a criptografia da mensagem do cliente. Neste caso, os algoritmos escolhidos seriam DES, SHA1 e MD5. Assim, para criptografia desta mensagem, qualquer desses algoritmos poderiam ser escolhidos, pois estariam codificando com as informações enviadas pelo cliente.

Portanto, ainda na classe de seleção do algoritmo, a mensagem do cliente é codificada com um dos algoritmos selecionados e gravada para utilização no método de response que realiza o envio da mensagem criptografada ao cliente.

5.1.3 Etapa F – Response e Request Seguras para REST

Por fim, classe APP do micro framework SLIM foi implementada para utilização de métodos específicos os quais cito os métodos **request() e response()**, reescritos de modo que este ultimo pudesse preparar a mensagem já criptografada, para envio ao cliente que fará o uso desta mensagem em uma comunicação com Web Service REST, descrita no estudo de caso apresentado como sendo uma possível extensão deste projeto).

5.2 Estudo de Caso: Utilização da API para Comunicação entre Cliente e Web Service REST

A utilização da web como meio de comunicação entre aplicações tem exigido que a pronta resposta seja um dos principais requisitos para que haja um feedback em que, pelo menos, o tempo de resposta seja reduzido. Isso porque maioria das aplicações utilizadas no dia-a-dia têm solicitado que o retorno das informações solicitadas seja o mais rápido possível. Por este motivo, o estudo de caso apresentado pretende mostrar que a comunicação realizada por meio do estilo REST pode oferecer facilidade de integração e alta performance por meio da exposição de recursos utilizando URIs e métodos HTTP, além de apresentar um bom potencial de escalabilidade que está relacionada a sua comunicação sem estados ou repositórios replicados.

Antes de estabelecer o canal de comunicação entre cliente e Web Service deve-se levar em consideração que a adaptação desta API precisa ocorrer de maneira protegida para que durante a sua utilização, o cliente esteja seguro para enviar suas informações para a API. Assim, torna-se relevante estipular alguns critérios que podem auxiliar a manter a segurança durante a transmissão de dados, os quais são citados a seguir.

• Definir quais são os clientes da API, conhecê-los e classificá-los conforme opções da tabela 7:

Tabela 7- Classificação dos Clientes da API (Bacili, 2016)

Básico (que não traz muitos riscos)	Clientes (apps) de um grupo restrito e local.
Intermediário	clientes acessam a API externamente mas com boa reputação.
Crítico	Clientes públicos, no qual qualquer destes podem acessar a API.

• Verificar se a API oferece acesso a informações relevantes, sendo estas classificadas como relaciona a tabela 8:

Tabela 8- Classificação das informações relevantes que serão criptografadas pela API(Bacili, 2016)

Acessórias	Não relacionadas a usuário
Sociais	Relacionada a Internet das Coisas.
Vitais	Dados que podem comprometer os negócios ou vida pessoal.

• Verificar se API faz alteração de dados importantes, isto é, se a API utiliza somente GETs (básico), GETs/POSTs/ PUTs em informações não vitais (intermediário) ou se a API faz uso de todos os métodos em recursos vitais.

Assim, com os critérios acima classificados, é possível estabelecer o quão exposta a API está com relação aos riscos de segurança e escolher um protocolo de comunicação que abranja as principais ameaças relacionadas a autenticação e autorização, privacidade, integridade, disponibilidade e auditoria que possam atingir a informação trafegada.

Para garantir que pelo menos os atributos acima citados sejam assegurados pela API, é sugerido algumas soluções na tabela 9.

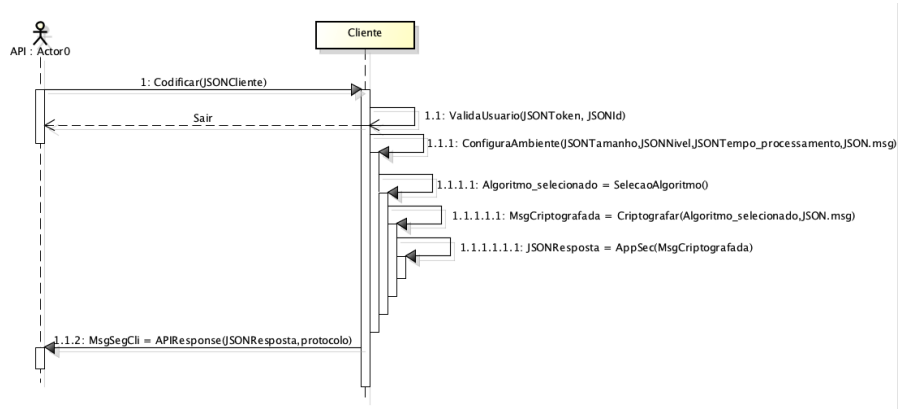
Tabela 9 - Atributos de Segurança (Bacili, 2016)

Atributos de segurança	**Soluções**
Autenticação e autorização	Confirmar que o usuário é realmente proprietário daquela requisição e garantir que ele terá acesso às informações pretendidas por meio da utilização de App tokens, caso a identificação do usuário não seja necessária e somente a autorização `a API deve ser detectada ou utilizando OAuth2, que provê acessos seguros a recursos disponibilizados pelo próprio dono da API, onde o cliente deve registrar-se no *Authorization server* para receber sua *Client* ID, *Client Secret* e uma URL de *redirect* pela qual será feita a trocas de tokens para posterior envio de informações. Dessa forma, o cliente sempre irá se comunicar com um token diferente a cada requisição que fizer para a API.
Privacidade	Garantir que somente o cliente autorizado pode ver a informação trafegada, por meio de protocolos SSL/TLS ou ainda ocultando informações relevantes, através de criptografia.
Disponibilidade	Conferir se a API está acessível ao cliente e se a mesma possui um desempenho satisfatório, sem que haja grande volume de requisição com objetivo de derrubar o serviço ou ainda sem que haja possibilidade de ataques do tipo INJECTION. Para isso, é sugerido que o monitoramento de tráfego da API seja realizado para possível identificação de ataques maliciosos, além de limitar o uso dos clientes, visando restringir o volume de requisições desnecessárias.

Integridade	Garantir de a API não estará acessível para alteração e/ou manipulação de informações por clientes não autorizados, evitando a exposição de recursos que não são usados e utilizando chaves opacas, isto é, chaves que não são sequenciais que pode impedir a dedução das chaves por parte de um cliente malicioso.
Auditoria	Comprovar que as requisições foram ou não executadas, tomando bastante cuidado com relação aos logs criados e tempo de armazenamento dos mesmos para que a informação não se transforme em ponto negativo para API.

Dessa maneira, levando como base os conceitos de segurança expostos, o estudo caso direcionado mostra que uma comunicação entre cliente e Service REST com a utilização da API pode ser mais segura caso os clientes sejam reconhecidos pela API, por meio de token e ID de identificação e as informações relevantes dos clientes sejam asseguradas pela API, por meio da codificação da mensagem, conforme representado na figura 16.

Figura 16- Comunicação Cliente X API (autoria própria, 2016)

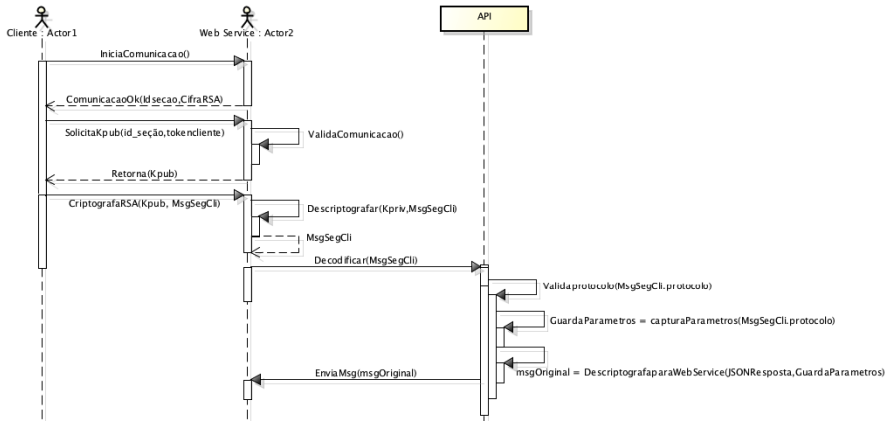


Após a definição dos atributos de segurança para comunicação entre cliente e API, propõe-se, conforme a figura 17, que o cliente se comunique com Web Service por meio do estilo REST que permite *handshake* utilizando requisições de pronta resposta para garantir que o cliente conectado possua acesso autorizado ao Web Service desejado e que, portanto, a comunicação a nível de transporte, estará segura para envio das informações criptografadas pela API.

Assim, por se tratar de uma requisição seguida de resposta, ao receber a requisição do cliente, Web Service precisará verificar se o cliente está autorizado a abrir a conexão, em caso positivo, o Web Service libera um token para que o cliente possa iniciar o envio da mensagem criptografada para o Web Service.

O Web Service, também deverá utilizar a API para entender a mensagem do cliente e, para isso, utilizará o protocolo utilizado pelo cliente para que, a partir deste protocolo, a API possa identificar quais foram os parâmetros utilizados para criptografar a mensagem do cliente e, então fazer o procedimento inverso para envio da mensagem pura ao Web Service, conforme descrito na figura 17.

Figura 17 - Comunicação Cliente x Web Service x API (autoria própria, 2016)



Assim, a segurança da comunicação entre cliente e Web Service REST, permanece resguardada pela proteção da camada de transporte, por meio dos *handshakes* cliente versus API e cliente versus Web Service, e também pela camada de aplicação, devido ao fato da mensagem estar protegida com criptografia executada pela API.

5.3 Considerações Finais do Capítulo

Dessa maneira, o capítulo 5 relata uma visão abrangente e detalhada do processo de desenvolvimento da API em cada uma das suas etapas de forma independente, além de incluir uma proposta de estudo de caso na qual a API desenvolvida deve realizar um *handshake* com o cliente para que este, por sua vez, possa criptografar sua mensagem e, estabelecer, na sequência, uma comunicação segura com o Web Service REST desejado.

6 RESULTADOS

Com base no desenvolvimento da API explicado no capítulo anterior, foi necessário realizar testes simulando o funcionamento da mesma. Assim, nesta seção, é apresentado os resultados analisando os seguintes itens: processo de classificação de níveis dos algoritmos criptográficos, desempenho dos algoritmos utilizados na implementação e execução da API em seu estado final, a fim de comprovar se os objetivos iniciais do projeto foram atingidos.

6.1 Sobre o Nível de Segurança dos Algoritmos

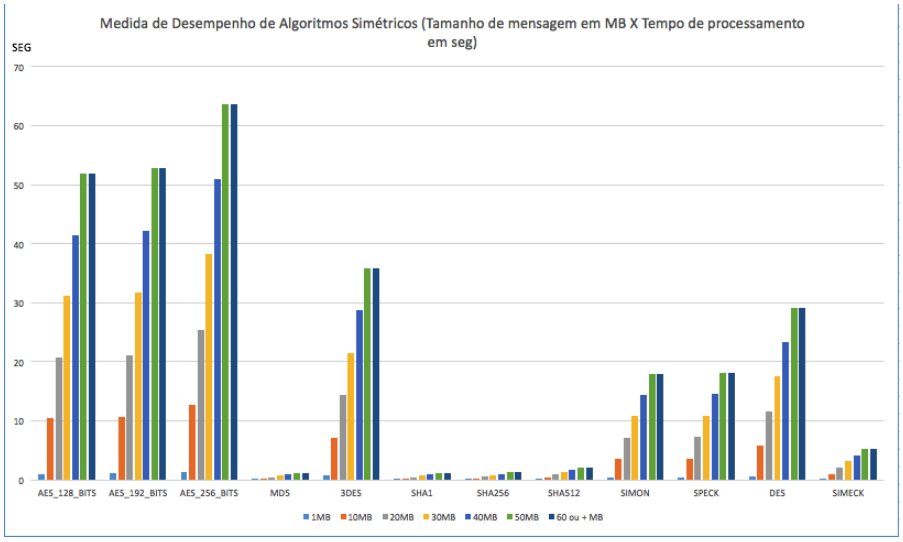
Apesar da etapa de classificação dos níveis de segurança apresentar informações que permitiria a exclusão de algoritmos criptográficos deste projeto, ainda assim optou-se por manter a implementação com os algoritmos apresentados, por levar em consideração que vários deles ainda são utilizados em alguns cenários distintos, proporcionando segurança. No caso do SHA1, por exemplo, o mesmo já foi parcialmente quebrado, mas ainda é utilizado para validar a integridade de conteúdos digitais em alguns sites, sendo gradativamente substituído pelo SHA-2 e o 3DES ainda bastante utilizado em operações bancárias.

6.2 Sobre o Teste de Desempenho dos Algoritmos

O teste de desempenho realizado nos algoritmos implementados, permitiu a avaliação destes com base em informações relevantes para o projeto, sendo elas tempo de processamento e tamanho de mensagem. Assim, de acordo com a figura 18, o tempo do processamento de dados combinados com tamanho da mensagem criptografada, medidos por meio de execução dos algoritmos para cada tamanho diferente de mensagem, possibilitou a visualização da variação que ocorre durante a criptografia de mensagens com tamanhos diferentes, indicando, portanto que os algoritmos de funções Hash e o SIMECK podem ser utilizados por qualquer tamanho de mensagem, já que estes apresentam um melhor desempenho durante a encriptação da mensagem, com variação de tempo de processamento definida entre 0 e 10 segundos (peso 1). Já os algoritmos simétricos AES, 3DES, DES, SIMON e SPECK mostram uma alteração de

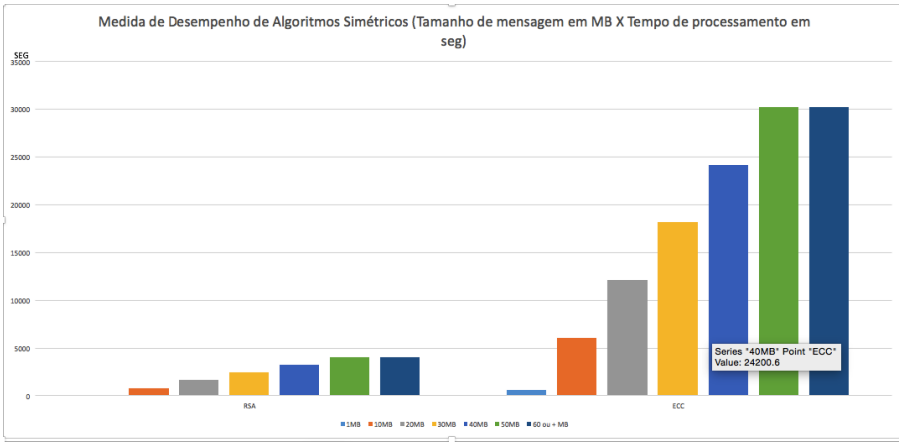
desempenho a partir de mensagens de 10 MB, as quais levam mais que 10 segundos para serem criptografadas.

Figura 18- Gráfico de desempenho dos algoritmos simétricos (autoria própria, 2016)



Por fim, RSA e ECC apontam um desempenho retardado, com uma disparidade visível maior que 50 segundos para criptografar mensagens de qualquer tamanho, conforme figura 19.

Figura 19 - Medida de desempenho dos algoritmos assimétricos (autoria própria, 2016)



Este resultado pode ser afirmado, também com base na literatura de ambos algoritmos, nos quais o autor Santos (2015) confirma que os algoritmos simétricos são os mais indicados para mensagens com grande volume de dados, devido ao desempenho comprovado, enquanto os assimétricos são recomendados para cenários em que se queira desconsiderar o uso de um canal seguro para compartilhar chaves, pois a criptografia assegura o canal por si só, ainda que perca em custo computacional.

Além disso, é possível comprovar através dos resultados reafirmados pela literatura, que apesar dos algoritmos simétricos apresentarem um bom desempenho, os assimétricos ainda vencem no quesito segurança, principalmente pelo fato de contar com duas chaves distintas e, muitas vezes extensas, para realizar a criptografia, o que dificulta possíveis ataques.

6.3 Sobre a Execução Completa da API

Finalmente, durante a execução completa da API, é possível identificar que o objetivo proposto pelo projeto é atingido, uma vez que a criptografia de mensagens enviadas para a API é realizada de forma automática por meio da seleção dos algoritmos implementados, conforme descrito no capítulo 5, item5.1.2, permitindo portanto, que a mensagem criptografada seja

utilizada como uma forma de proteger os dados transmitidos em cenários onde há consumo de Web Services.

Além disso, o resultado final deste desenvolvimento contribui com a criação de uma API capaz de simplificar a inclusão de técnicas de segurança de dados em aplicações e projetos que não possuem uma implementação mais avançada no que se refere a segurança. A figura 20 ilusta uma requisição à API realizada por uma aplicação web, comprovando, portanto, a simplicidade da inclusão desta técnica de segurança.

Figura 20 - Código exemplo de requisição da API em uma aplicação (autoria própria, 2016)

```
function Fazer_Request_API()
{
    //monta a url da API e o JSON do request
    var url_request = "http://localhost/service/codificar/";
    var json_request = '{'+
                            '"client_id":"e8dcdfd79e471bb794e80fcc056d876752fcc8c6",'+
                            '"client_token":"9fa41145a075ee591b8d07cafd0e9fac5384eb08",'+
                            '"texto":"AQUI O TEXTO QUE DEVE SER CODIFICADO",'+
                            '"tamanho":1,'+
                            '"nivel":3,'+
                            '"tempo_processamento":1'+
                        '}';
    var RESPOSTA_em_JSON = null;
    //efetua a requisicao retornando JSON_RESPOSTA como um objeto
    $.getJSON(url_request+json_request, function(JSON_REPOSTA)
    {
        RESPOSTA_em_JSON = JSON_REPOSTA;
    })
    //em caso de sucesso escreve o resultado na tela
    .done(function()
    {
        var JSON_REPOSTA_STRING = converter_json_para_string(RESPOSTA_em_JSON);
        escrever(JSON_REPOSTA_STRING);
    })
    //em caso de falha escreve a mensagem uma erro
    .fail(function()
    {
        escrever( "Erro ao efetuar requisição" );
    });
}

function converter_json_para_string(string)
{
    return JSON.stringify(string);
}

function escrever(string)
{
    alert(string);
}
```

6.4 Considerações Finais do Capítulo

O capitulo 6, discorre sobre os resultados finais do desenvolvimento proposto, mostrando que o objetivo inicial do trabalho é atingido em sua totalidade.

7 CONCLUSÃO

Nesta etapa, apresenta-se a conclusão do trabalho, bem como as as contribuições oferecidas durante o desenvolvimento do projeto.

7.1 Contribuições do Projeto

Ainda que a segurança seja uma das principais, senão a principal necessidade dentro de uma comunicação entre clientes e Web Services, os atuais protocolos de segurança disponíveis são, normalmente, representados a nível de transporte ou de XML. Os SSLs (*Secure Sockets Layer*), por exemplo, é um protocolo que visa proteger a rede utilizada pelos Web Services para se comunicar, “encriptando todo o tráfego de rede sobre o socket”(Silva, 2007) ou ainda autenticando Web Services por meio de PKIs, isto é, Infraestrutura de chaves públicas. Já a segurança a nível de XML, envolve a criptografia de documentos XMLs, como é o caso do XKMS (*XML Key Management Services*) que fornece um protocolo para gerenciamento de chaves públicas, com a possibilidade de obter informações complementares incluindo registros, verificação e revogação de permissões destas chaves e O WS- Security que propõe uma maneira segura de comunicação com Web Services, incluindo dados relativos a segurança no cabeçalho do envelope SOAP (Silva et al, 2005).Todos exemplos, embora ofereçam proteção durante a comunicação com Web Services, ainda não contemplam uma segurança fim-a-fim.

Assim, diante deste contexto, o objetivo deste trabalho é inserido, de modo a desenvolver uma API que além de permitir uma segurança fim-a-fim, possa complementar o nível de proteção disponibilizando para o cliente uma segurança a nível de aplicação. Portanto, se durante uma comunicação a segurança do cliente seja violada ainda na camada de transporte, haverá também uma proteção na camada de aplicação.

Com a implementação desta API, foi possível garantir que o algoritmo mais adequado fosse escolhido para realizar a criptografia de mensagens em determinados cenários, avaliando, para isso, o desempenho de cada algoritmo na relação tempo de execução *versus* tamanho da mensagem e definindo níveis de segurança aos mesmos com base na literatura. A confluência dessas variáveis possibilitou a consolidação da tomada de decisão, conforme descrito no capítulo 5, item 5.1.2. Isso porque desempenho e nível de segurança tendem a se complementar, pois alto desempenho sem segurança é tão ineficiente quanto nível de segurança elevado e baixo desempenho. Portanto, embora houvesse diversas variáveis a serem utilizadas para a escolha mais eficiente do algoritmo, na qual podemos citar velocidade da rede, tempo de processamento no cliente e no servidor, frequência de requisição da informação cifrada, tipo de arquivo cifrado, entre outros, optou-se por explorar aquelas que eram mais relevantes para o desenvolvimento da API, considerando a possibilidade de inclui-las em trabalhos futuros, completando a eficácia da API.

Além disso, embora o desempenho tenha sido estimado de forma simplificada (utilizando apenas comparação entre tempo e tamanho da mensagem a ser cifrada pelo algoritmo), este indicador foi suficiente para comprovar o funcionamento da API, não descartando a oportunidade de incluir no projeto novas maneiras de refinar a medida do desempenho, utilizando técnicas mais específicas.

Dessa forma, o objetivo inicialmente traçado foi atingido, mostrando por meio deste trabalho que a utilização da API desenvolvida contribui para a inclusão de uma medida de segurança em um nível acima da camada de transporte, facilitando o aperfeiçoamento de aplicações que não possuem em sua estrutura um nível de segurança adequado para transmitir suas informações para Web Services. Portanto, ainda que a camada de transporte tenha sua segurança violada ou inoperante (certificados SSLs e protocolos expirados, por exemplo), haverá na camada de aplicação uma maneira segura de impedir o acesso de terceiros a informação transportada.

No link https://github.com/DrielyAoyama/service é possível visualizar toda a implementação executada para o desenvolvimento do projeto proposto. A pasta APP é composta pelos códigos implementados, os quais foram subdivididos em:

- MVC: possui o conteúdo dos controllers, models e views do trabalho;
- algoritmos: contempla algumas bibliotecas e algoritmos utilizados na implementação;
- interno: contém os códigos internos dos micro-frameworks utilizados.

7.2 Dificuldades do Projeto

Durante a elaboração do projeto várias dificuldades foram encontradas, as quais podemos citar:

- Implementação dos algoritmos de blocos de cifra leve. Uma das grandes adversidades na implantação foi ter que reescrever os algoritmos SIMON, SIMECK e SPECK, baseados em implementações já prontas, mas em linguagens diferentes da utilizada no projeto. Todas as implementações desses algoritmos são encontradas em linguagem C, Phython ou Java, as quais demandam um conhecimento mais específico para compreensão dos códigos.
- Classificação do nível de segurança para cada algoritmo implementado. Com base em algumas informações específicas como datas de criação dos algoritmos e datas do primeiro ataque contra os algoritmos, foi definido um nível de segurança para cada uma das implementações. No entanto, houve bastante dificuldade em encontrar as datas dos primeiros ataques, havendo necessidade de dedicar boa parte do tempo do projeto para buscar as melhores fontes de informação.

7.3 Trabalhos Futuros

Considerando a possibilidade de estender o trabalho e, então permitir a inclusão de funções que possam enriquecer o projeto, propõe-se as seguintes ideias:

- Elaboração da documentação da API para utilização dos desenvolvedores.
- Inclusão de novos parâmetros para classificação do cenário do cliente (conforme visto na tabela 2), de modo que os novos atributos contribuam com o aprimoramento da seleção do algoritmo.
- Utilização de técnicas mais específicas para medir desempenho dos algoritmos e oferecer dados mais assertivos com relação a performance dos algoritmos.
- Implementação da comunicação *handshaking* para que cliente e servidor façam o uso da API desenvolvida.
- Estudo de caso com outras tecnologias de comunicação como o Websocket, por exemplo, para avaliação de qual tecnologia possui melhor eficácia com a utilização da API.

Portanto, as propostas acima visam aperfeiçoar o projeto, oferecendo uma nova solução que atinge todo cenário de estudo, desde o momento em que se estabelece a comunicação entre cliente e Web Service, até o momento em que a API é utilizada por ambos os lados

8 Referências Bibliográficas

ABED, F. et al. **Differential Cryptanalysis of Round-Reduced Simon and Speck.Bauhaus**. Universität Weimar, Germany. 2014. Disponível em : http://www.iacr.org/archive/fse2014/85400194/85400194.pdf . Acesso em: 30 set. 2016.

ABREU, L. F. S. **A Segurança da Informação nas Redes Sociais**. 2011. 55 f. Trabalho de Conclusão de Curso para Grau de Tecnólogo em Processamento de Dados - Faculdade de Tecnologia de São Paulo. São Paulo. 2011. Disponível em: http://www.fatecsp.br/dti/tcc/tcc0023.pdf .Acesso em: 26 mar. 2016.

ARORA, Anjali. et al. **A Survey of Cryptanalytic Attacks on Lightweight Block Ciphers**. International Journal of Computer Science and Information Technology & Security. Vol 2. 2012. Disponível em: http://ijcsits.org/papers/Vol2no22012/43vol2no2.pdf . Acesso em: 19 mai. 2016.

BACILI, Kleber. **Webnar RESTful API Design.** Webinar Sensedia. 2016. Disponível em : http://downloads.sensedia.com/webinar-design-de-apis-restful. Acesso em: 6 out. 2016.

BIRYUKOV, Alex et al. **Lightweght block Ciphers**. Université du Luxenbourg. 2016. Disponível em https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers .Acesso em: 4 jun. 2016.

BOX, D. et al. **Simple Object Access Protocol (SOAP) 1.1**. W3C. 2000. Disponível em: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Acesso em: 18 mar 2016.

BRITTENHAM, P. et al**. Understanding WSDL in a UDDI Registry**. IBM Software Group. Disponível em: http://www.ibm.com/developerworks/library/ws-wsdl/ Acesso em: 18 mar. 2016.

BURNETT, Steven. **Criptografia e segurança: o guia oficial RSA**. Editora Campus. 2002. Disponível em: https://books.google.com.br/books?id=DIskNhcTUNoC&pg=PA11&hl=pt-BR&source=gbs_toc_r&cad=3#v=onepage&q=editor&f=false. Acesso em: 18 mar. 2016.

CAZORLA, M. et al. **Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks***. Cryptology ePrint Archive. 2013. Disponível em: https://eprint.iacr.org/2013/295.pdf. Acesso em: 28 fev. 2016.

CHIARAMONTE, R. B. et al. **Aspectos de Desempenho do SICO- Sistema de Comunicação de Dados com Suporte Dinâmico a Segurança**. Workshop em Sistemas Computacionais de Alto Desempenho. 2006. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/wscad/2006/0014.pdf. Acesso em: 05 abr. 2016.

CHRISTENSEN, Erik et al. **Web Services Description Language**. W3C. 2001. Disponível em: https://www.w3.org/TR/wsdl. Acesso em: 18 mar 2016.

COLAN, M**. Standards and web services**. IBM Software Group. 2004. Disponível em: https://www.ibm.com/developerworks/webservices/standards/ Acesso em: 18 mar. 2016.

CORRÊA, A. S F M. -**Algoritmos Simétricos para Software Embarcado**. Programa de Pós-Graduação em Informática Segurança de Redes - Universidade Federal do Rio de Janeiro. 2009. Disponível em: http://equipe.nce.ufrj.br/rust/Mestrado%202009/SionSimetricAlg2009_SR.pdf . Acesso em: 15 out 2016.

COSTA, C. et al. **Análise de Metodologias de Implementação e Desempenho em FPGA dos Algoritmos Criptográficos Leves Simon e Speck**. Centro Universitário Eurípides de Marília (UNIVEM) - Computing and Information Systems Research Lab (COMPSI) - Marília-SP. 2016. Disponível em: http://sbrc2016.ufba.br/downloads/WoCCES/154894.pdf . Acesso em: 29 mai 2016.

DAEMEN, J. et al. **The Sponge Functions Corner.** 2013. Disponível em: http://sponge.noekeon.org/.Acesso em: 29 mai. 2016.

DAEMEN, J. et al. **The Keccak Reference**. Version 3. 2011b. Disponível em: http://keccak.noekeon.org/Keccak-reference-3.0.pdf.Acesso em: 29 mai. 2016.

EISENBARTH, T. et al. **A Survey of Lightweight-Cryptography Implementations**. IEEE Design & Test of Computers. 2007. Disponível em: https://securewww.esat.kuleuven.be/cosic/publications/article-1446.pdf. Acesso em: 28 fev. 2016.

FIDEL, B. **Web Services REST versus SOAP**. 2015. Disponível em : http://www.devmedia.com.br/web-services-rest-versus-soap/32451. Acesso em 04 abr. 2016.

FREIRE, P. B. et al. **A Matemática dos Códigos Criptográficos**. Curso de Matelática - Universidade Católica de Brasília. 2007. Disponível em: https://www.ucb.br/sites/100/103/TCC/12007/PalomaBarbosaFreire.pdf. Acesso em: 20 abr. 2016.

HINZ, M. A. M. **Um Estudo Descritivo de Novos Algoritmos de Criptografia.** Universidade Federal de Pelotas. 2000. Disponível em: http://www.jabour.com.br/ufjf/apa/Mono-MarcoAntonio.pdf . Acesso em: 28 fev. 2016.

HOLANDA, M. T. et al. **Segurança no Desenvolvimento de Aplicações**. Gestão da Segurança da Informação e Comunicações. Versão 1. 2009-2011. Disponível em: http://mauriciolyra.pro.br/site/wpcontent/uploads/2015/12/22Seguranca_Desenvolvimento_Aplicacoes.pdf. Acesso em: 03 mar. 2016.

JUNIOR, B. S. Web Services. 2016. Disponível em: http://www.criandobits.com.br/fs-programacao/fs_materias-webservice.php . Acesso em: 20 mar. 2016.

LIMA, J. C. R. **Web Services (SOAP X REST).** 2012. 41 f. Trabalho de Conclusão de Curso para Grau de Tecnólogo em Processamento de Dados - Faculdade de Tecnologia de São Paulo,

São Paulo. 2012. Disponível em: http://www.fatecsp.br/dti/tcc/tcc00056.pdf. Acesso em: 18 mar. 2016.

MARQUES, F. Y. Projeto, **Desenvolvimento e Análise de um dos Algoritmos SHA-3 Finalistas em Smart Cards**. Centro Universitário "Eurípedes de Marilia" – Univem Bacharelado em Ciência da Computação. 2011. Disponível em: http://aberto.univem.edu.br/bitstream/handle/11077/369/ . Acesso em: 20 mai. 2016.

MENEZES, A. J et al. **Handbook of Applied Cryptography**. CRC Press. 1996. Disponível em: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.99.2838&rep=rep1&type=pdf. Acesso em: 18 mar. 2016.

MORAES, J et al. **Web Services**. Artigos. Pontifica Universidade Católica do Rio Grande do Sul. 2010. Disponível em:http://www.inf.pucrs.br/~gustavo/disciplinas/sd/material/Artigo_WebServices_Conceitual.pdf . Acesso em: 27 fev. 2016.

PINHO, S. C. S. **Arquitectura de Segurança num ambiente SOA (*Service Oriented Architecture*).** Dissertação de Mestrado – Engenharia de Informática e Computação, Faculdade de Engenharia d Universidade do Porto. 2008. Disponível em: https://repositorio-aberto.up.pt/bitstream/10216/57684/2/Texto%20integral.pdf . Acesso em : 15 mai. 2016.

PRASS, F. **Aspectos com Padrões de Segurança.** Artigo Revista Java Magazine, 93. 2011. Disponível em: http://www.devmedia.com.br/aspectos-com-padroes-de-seguranca-artigo-revista-java-magazine-93/21654. Acesso em : 01 jun. 2016.

RINALDI, G. D. **Análise do AES e sua Criptoanálise Diferencial**. 2012. 71 fl. Trabalho de Graduação - Universidade Federal do Rio Grande do Sul. Porto Alegre. 2012. Disponível em: https://www.lume.ufrgs.br/bitstream/handle/10183/54139/000855639.pdf?sequence=1. Acessado em: 21 Out 2016.

RODRIGUES, D. **Um Estudo Comparativo das Especificações de Segurança Aplicadas a uma Arquitetura Orientada a Serviços**. Dissertação de Mestrado - Universidade de São Paulo. São Carlos. 2011. Disponível em: http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19072011-140007/pt-br.php. Acesso em: 27 fev. 2016.

RODRIGUEZ, A. **RESTful Web services: The basics**. IBM Software Group. 2008. Disponível em : http://www.ibm.com/developerworks/library/ws-restful/. Acesso em: 18 mar. 2016.

SANGALLI, L~~l~~. A. **Criptossistemas baseados em curvas elípticas e seus desafios**. Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas (Unicamp), 2012. Disponivel em: . Acesso em: 12 de set. 2016.

SANTOS, D. A~~a~~. et al. **Estudo sobre o desenvolvimento de aplicações VOIP para plataforma Windows com criptografia**. Trabalho de graduação - Universidade Federal do Estado do Rio De Janeiro Centro de Ciências Exatas e Tecnologia Escola de Informática Aplicada. 2015. Disponível em: http://bsi.uniriotec.br/tcc/201512AraujoRamos.pdf. Acesso em: 22 out. 2016.

SARANYA, K. et al. **A Review on Symmetric Key Encryption Techniques in Cryptography**. International Journal of Science, Engineering and Technology Research, Vol 3, 2014. Disponível em: http://ijsetr.org/wp-content/uploads/2014/03/IJSETR-VOL-3-ISSUE-3-539-544.pdf. Acesso em: 19 mar. 2016

SILVA, R. F. et al. **Arquitetura de Segurança em Aplicações Baseadas em Web Services.** Portal de Periódicos do IFRN – HOLOS, Instituto Federal do Rio Grande do Norte,v. 3,p.15-24, 2005. Disponível em: http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/77 . Acesso em: 27 fev. 2016.

Formatado: Recuo: Primeira linha: 0 cm, Espaço Antes: 6 pt, Depois de: 6 pt

Formatado: Internet Link, Fonte: Cor da fonte: Texto 1, Oculto

Formatado: Fonte: Cor da fonte: Texto 1

Formatado: Fonte: Cor da fonte: Texto 1

Formatado: Cor da fonte: Texto 1

Formatado: Cor da fonte: Texto 1

Formatado: Internet Link, Cor da fonte: Texto 1, Oculto

Formatado: Cor da fonte: Texto 1

SILVA, J. **Segurança em Web Services**. 2007. Disponível em: http://araticum.infonet.com.br/andres/apresentacoes/artigos/SegurancaWebServices%20-%20JandsonAlmeidaSilva.pdf. Acesso em: 29 mai. 2016.

SOTIROV, A. et al. **MD5 Considered Harmful Today**. Creating a roque CA Certificate. 2008. Disponível em : https://www.win.tue.nl/hashclash/rogue-ca/ . Acesso em: 11 Set. 2016.

SOUZA, S. C. **Segurança para Web Services com Criptografia Heterogênea Baseada em Proxy**. 2010. 87 f. Dissertação de Mestrado - Pontifica Universidade Católica do Rio Grande do Sul. Porto Alegre. 2010. Disponível em: http://tede.pucrs.br/tde_busca/arquivo.php?codArquivo=2872. Acesso em: 27 fev. 2016.

SOUZA, A. M. **Comparação de Desempenho entre Diferentes Implementações do Algoritmo Keccak para Plataformas GPGPUS Utilizando Opencl.** Centro Universitário "Eurípedes de Marilia" – Univem Bacharelado em Ciência da Computação. 2013. Disponível em: http://aberto.univem.edu.br/bitstream/handle/11077/968/. Acesso em: 29 mai. 2016.

STALLINGS, W. **Cryptography and Network Security Principles and Practice Fifth Edition**. Prentice Hall. 2011. Disponível em: http://faculty.mu.edu.sa/public/uploads/1360993259.0858Cryptography%20and%20Network%20Security%20Principles%20and%20Practice,%205th%20Edition.pdf . Acesso em: 18 mai. 2016.

SUDA, B. **SOAP Web Services**. 2003. 66 f. Dissertação de Mestrado em Ciência da Computação - School of Informatics University of Edinburgh, 2003. Disponível em: http://suda.co.uk/publications/MSc/brian.suda.thesis.pdf. Acesso em: 18 mar. 2016.

TORRES, I. E. et al. **Os Dez Riscos de Segurança mais Críticos em Aplicações Web**. OWASP Top Ten Project. 2013. Disponível em: https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013. Acesso em: 10 mar. 2016.

TRINTA, F. A. M. **Um Estudo sobre Criptografia e Assinatura Digital**. Departamento de Informática Universidade Federal de Pernambuco. 1998. Disponível em: http://www.di.ufpe.br/~flash/ais98/cripto/criptografia.htm. Acesso em: 07 abr. 2016.

VALENTA, L. et al**. Factoring as a Service.** University of Pensylvania. 2015. Disponível em: https://eprint.iacr.org/2015/1000.pdf. Acesso em : Acesso em 06 out. 2016.

VOGT, F. C. **Criptografia: Desvendando métodos AES Advanced Encryption Stadard**. Departamento de Tecnologia da Universidade Regional do Noroeste do Estado do Rio Grande do Sul. Disponível em: http://www-usr.inf.ufsm.br/~fcvogt/artigos/Aes.PDF. Acesso em: 27 Set. 2016.