Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias da Informação

SecJSON - Secure Javascript Object Notation

Tiago Filipe Paulino Santos

Dissertação submetida como requisito parcial para obtenção do grau de

Mestre em Engenharia Informática

Orientador

Prof. Dr. Carlos Serrão, Professor Auxiliar

ISCTE-IUL

Setembro 2016

"O único lugar onde o sucesso vem antes do trabalho é no dicionário."

Albert Einstein

Resumo

A presente dissertação tem como objetivo estudar, analisar, desenvolver e disponibilizar um mecanismo de segurança adicional para o formato JSON. Este trabalho surge da necessidade de aumentar a granularidade da segurança deste popular mecanismo de comunicação de serviços *Web*. O *Secure JavaScript Object Notation* (SecJSON) possibilita a cifra total ou parcial dos dados contidos em estruturas JSON (*payload*), garantindo confidencialidade, integridade e disponibilidade das mensagens neste formato. No decorrer deste trabalho é apresentada uma definição da sintaxe, regras de cifra/decifra bem como uma implementação na framework *Node.js*. Esta implementação foi submetida a diferentes validações de performance para permitir uma rigorosa comparação com outra solução de segurança, nomeadamente, o SSL/TLS, visto ser atualmente o principal mecanismo de cifra dos serviços *Web*. Em suma, o SecJSON mostrou que é uma alternativa ou complemento válida ao SSL/TLS, onde apresentou uma maior flexibilidade e até performance superior em diversos cenários de segurança.

Palavras-chave: SecJSON, JSON, segurança, SSL.

Abstract

This dissertation aims to study, analyze, develop and provide an additional security mechanism for the JSON format. This work arises from the need to increase the granularity of the safety of this popular web services communication mechanism. Secure JavaScript Object Notation (SecJSON) enables the partial or total encryption of a message payload in JSON structures, ensuring the confidentiality, integrity and availability of messages in this format. In this paper is presented a definition of the syntax rules of encryption / decryption and an implementation in Node framework. This implementation was subjected to different validations of performance to allow a precise comparison with other security solution, namely SSL/TLS, currently seen to be the main figure of mechanism of web services. In short, the SecJSON presents itself as a valid alternative or complement to SSL/TLS, which showed greater flexibility and even better performance in various security scenarios.

Keywords: SecJSON, JSON, security, SSL.

Agradecimentos

A dissertação que aqui se apresenta resulta de um trajeto complicado, ao longo do qual recebi o maior apoio e estímulo de muitas pessoas às quais quero prestar o meu maior agradecimento. Assim, os méritos que esta possa alcançar, devem-se aos contributos das pessoas que durante a sua elaboração, me proporcionaram testemunhos essenciais para o seu desenvolvimento.

Gostaria de destacar o papel desempenhado pelo meu orientador, o Sr. Dr. Carlos Serrão, por todo o trabalho de orientação, pela experiência e conhecimentos transmitidos e por toda a disponibilidade oferecida durante o desenvolvimento deste trabalho.

Agradeço aos meus pais que, apesar da distância sob a qual assistiram a esta minha nova etapa, me garantiram ao longo da minha vida um apoio e estabilidade incondicional, permitindo que me tornasse uma pessoa fiel aos ideais e valores por eles transmitidos.

Quero agradecer a toda a minha família e amigos chegados que me motivaram nas alturas mais complicadas e sem as quais esta dissertação não teria sido concluída.

Desejo ainda expressar o meu agradecimento a todos os restantes colegas que me acompanharam ao longo do meu percurso académico que, direta ou indiretamente, contribuíram para a conclusão desta dissertação.

Conteúdo

Resumo **v**

Abstract **vii**

Agradecimentos **ix**

Lista de Figuras **xv**

Listagem de Código-Fonte **xix**

Abreviaturas **xix**

1 Introdução **1**
- 1.1 Motivação . . . 2
- 1.2 Enquadramento . . . 3
- 1.3 Questões de Investigação . . . 6
- 1.4 Objetivos . . . 6
- 1.5 Metodologia de Investigação . . . 7
  - 1.5.1 *Relevance Cycle* . . . 8
  - 1.5.2 *Design Cycle* . . . 9
  - 1.5.3 *Rigor Cycle* . . . 9

2 Revisão da Literatura **11**
- 2.1 Arquiteturas orientadas por serviços . . . 11
- 2.2 Serviços *Web* . . . 14
  - 2.2.1 *Remote Procedure Call* (RPC) . . . 14
  - 2.2.2 *Simple Object Access Protocol* (SOAP) . . . 15
- 2.3 *Representational State Transfer* (REST) . . . 15
  - 2.3.1 *JavaScript Object Notation* (JSON) . . . 17
  - 2.3.2 *eXtensible Markup Language* (XML) . . . 19
  - 2.3.3 YAML . . . 20
- 2.4 Segurança para serviços *Web* . . . 21
  - 2.4.1 *Web Services Security* (WS-Security) . . . 22
    - 2.4.1.1 *XML Signature* (XMLDSign) . . . 23
    - 2.4.1.2 *XML Encryption* . . . 24
  - 2.4.2 SSL/TLS . . . 26

2.4.3 Segurança para JSON . . . 27
2.4.3.1 *JSON Web Encryption* . . . 28
2.4.3.2 *JSON Web Signature* . . . 29
2.4.3.3 *JSON Web Key* . . . 30
2.4.3.4 *Javascript Object Signing and Encryption* . . . 30
2.5 Plataformas Desenvolvimento . . . 30
2.5.1 Python . . . 31
2.5.2 Ruby . . . 32
2.5.3 Node.js . . . 33
2.6 Análise à literatura . . . 34

3 Desenho **35**
3.1 *Secure JSON* (SecJSON) . . . 36
3.1.1 Requisitos . . . 36
3.2 Sintaxe . . . 37
3.2.1 Elemento abstrato *EncryptedType* . . . 38
3.2.2 Elemento *EncryptionMethod* . . . 39
3.2.3 Elemento *CipherData* . . . 40
3.2.3.1 Elemento *CipherReference* . . . 41
3.2.4 Elemento *EncryptedData* . . . 42
3.2.5 Elemento *KeyInfo* . . . 43
3.2.5.1 Elemento *EncryptedKey* . . . 44
3.2.5.2 Elemento *RetrievalMethod* . . . 45
3.2.6 Regras de Processamento . . . 45
3.2.6.1 Cifra . . . 45
3.2.6.2 Decifra . . . 47
3.2.7 Algoritmos . . . 49

4 Implementação do Sistema **53**
4.1 Ambiente de Desenvolvimento . . . 54
4.2 Arquitectura . . . 54
4.3 *secjson.js* . . . 55
4.3.1 Processo de cifra . . . 55
4.3.2 Processo de decifra . . . 58
4.3.3 Criptografia JSON . . . 60
4.3.4 Processo de cifra JSON . . . 60
4.3.5 Processo de decifra JSON . . . 61
4.4 Utilização do sistema . . . 62
4.4.1 Criptografia Simples . . . 63
4.4.2 Criptografia JSON . . . 65

5 Validação do Sistema **69**
5.1 Transmissão entre dois nós (Cliente - Servidor) . . . 71
5.2 Transmissão para multiplos nós . . . 75
5.3 Número de Nós . . . 80

6 Conclusões **83**
6.1 Conclusões Gerais 83
6.2 Contribuições 85
6.3 Limitações e Trabalho Futuro 86

Bibliografia **89**

Lista de Figuras

1.1 Interesse ao longo do tempo - XML vs JSON [38] 3
1.2 Design Science Research Cycles [15] 7
1.3 Transmissão de objetos JSON com várias cifras 8

2.1 Client-Server [10] 16
2.2 Client-Cache-Stateless-Server [10] 16
2.3 Uniform-Client-Cache-Stateless-Server [10] 17
2.4 Encaminhamento de um pedido *Web* ao nível aplicacional [8] 22
2.5 Formas de assinaturas XMLDSign [8] 24
2.6 Problema SSL/TLS 27
2.7 Ranking TIOBE Julho 2015 [37] 31

3.1 Processo SecJSON para tipos primários/estruturados 36
3.2 Processo SecJSON para um documento JSON 36
3.3 Processo de cifra SecJSON 47
3.4 Processo de decifra SecJSON 49

4.1 Arquitectura SecJSON 55

5.1 Transmissão SecJSON entre dois nós 72
5.2 Transmissão SSL entre dois nós 72
5.3 Transmissão SecJSON do Ficheiro 1 73
5.4 Transmissão SSL do Ficheiro 1 73
5.5 Transmissão SecJSON do Ficheiro 2 73
5.6 Transmissão SSL do Ficheiro 2 74
5.7 Transmissão SecJSON do Ficheiro 3 74
5.8 Transmissão SSL do Ficheiro 3 74
5.9 Transmissão em função do tamanho do *Payload* 75
5.10 Transmissão SecJSON entre multiplos nós 77
5.11 Transmissão SSL entre multiplos nós 77
5.12 Transmissão SecJSON do Ficheiro 1 78
5.13 Transmissão SSL do Ficheiro 1 78
5.14 Transmissão SecJSON do Ficheiro 2 78
5.15 Transmissão SSL do Ficheiro 2 79
5.16 Transmissão SecJSON do Ficheiro 3 79
5.17 Transmissão SSL do Ficheiro 3 79
5.18 Transmissão em função do tamanho do *Payload* 80

5.19 Evolução relativa ao numero nós - Ficheiro 1 81
5.20 Evolução relativa ao numero nós - Ficheiro 2 81
5.21 Evolução relativa ao numero nós - Ficheiro 3 82

Listagem de Código-Fonte

2.1 Exemplo JSON 18
2.2 Exemplo XML 19
2.3 Exemplo YAML 21
2.4 Exemplo XML Encryption 24
3.1 Elemento abstrato *EncryptedType* 39
3.2 Elemento *EncryptionMethod* 39
3.3 Elemento *CipherData* 40
3.4 Elemento *CipherReference* 41
3.5 Elemento *EncryptedData* 42
3.6 Elemento *KeyInfo* 43
3.7 Elemento *EncryptedKey* 44
3.8 Elemento *RetrievalMethod* 45
4.1 Função encrypt 56
4.2 Função decrypt 58
4.3 Função de cifra JSON 61
4.4 Função de pesquisa e cifra JSON 61
4.5 Função de decifra JSON 62
4.6 Função de pesquisa e decifra JSON 62
4.7 Cifra SecJSON (Simples) 63
4.8 Resultado Cifra (Simples) 64
4.9 Decifra SecJSON (Simples) 65
4.10 Cifra elemento JSON 65
4.11 Resultado Cifra JSON 66
4.12 Decifra elemento JSON 67
5.1 Ficheiro de teste 1 69
5.2 Ficheiro de teste 2 70
5.3 Ficheiro de teste 2 70

Abreviaturas

JSON	**J**ava**S**cript **O**bject **N**otation
NSA	**N**ational **S**ecurity **A**gency
XML	e**X**tensible **M**arkup **L**anguage
YAML	**Y**AML **A**in't **M**arkup **L**anguage[1]
SSL	**S**ecure **S**ockets **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
TI	**T**ecnologias da **I**nformação
URI	**U**niform **R**esource **I**dentifier
API	**A**pplication **P**rogramming **I**nterface
NPM	**N**ode **P**ackage **M**anager

[1] Pode ainda significar *Yet Another Markup Language*

Capítulo 1

Introdução

Atualmente, uma das principais formas que permite a troca de informação entre serviços disponíveis na Web passa pela utilização do *JavaScript Object Notation* (JSON), um formato aberto e padrão que utiliza texto para facilitar o transporte, processamento e interoperabilidade durante a transmissão da informação [31].

Segundo Douglas Crockford, criador do formato JSON, a primeira vez que se identificou a utilização da linguagem JavaScript para transmissão da informação foi no Netscape (1996). Na NeXT (mais tarde comprada pela Apple) também é possível identificar padrões semelhantes na representação de dados, razões que levam Crockford a afirmar que o JSON é uma forma natural de representação para dados que serão consumidos por linguagens de programação [31].

A primeira implementação de Crockford tinha como objectivo facilitar a comunicação entre programas JavaScript e servidores baseados em Java. A simplificação desta solução resultou no que hoje conhecemos por JSON. Apesar de nascer do JavaScript, o JSON tornou-se independente da linguagem [31].

Nos últimos anos verificou-se um crescimento na utilização deste formato durante a transmissão da informação entre serviços *Web*, desta forma a segurança inerente ao JSON e à informação sensível que este transporta (*payload*) ganha uma maior importância.

Sabendo que hoje pode existir informação crítica a ser redirecionada por múltiplas partes, sem que as mesmas sejam o destinatário final dessa informação, a possibilidade de cifrar parcialmente/totalmente o *payload* de uma mensagem terá um impacto positivo na utilização do JSON. Desta forma poderemos usar, por exemplo, uma mensagem com múltiplas cifras destinadas a diferentes entidades.

1.1 Motivação

A criptografia de mensagens é utilizada há milhares de anos (os primeiros textos cifrados datam a 1900 a.C.), maioritariamente em cenários de guerra onde a decifra de mensagens secretas poderia ter resultados desastrosos [9]. Atualmente a maioria das trocas de informação são estabelecidas através da *World Wide Web* (WWW ou *Web*) [39], sabendo que estas trocas podem ser críticas para os seus intervenientes é necessário assegurar a máxima segurança das mesmas.

As organizações dependem cada vez mais da informação que obtêm do contexto em que estão envolvidas influenciando decisões organizacionais, competitividade e o acesso a vantagens estratégicas [32]. A segurança inerente ao processo de troca de mensagens entre sistemas é indispensável, pelo que o acesso a informação sensível por terceiros deve ser restrito ou bloqueado.

Por outro lado, o crescimento do número de API's baseadas em JSON verificado nos últimos anos (Figura 1.1) obriga a uma maior preocupação na proteção de mensagens JSON. Desta forma um mecanismo que possibilite cifrar parcialmente ou completamente a informação de uma mensagem JSON permitirá aumentar a granularidade da segurança neste popular mecanismo de comunicação em serviços Web, não descurando a confidencialidade e integridade da informação.

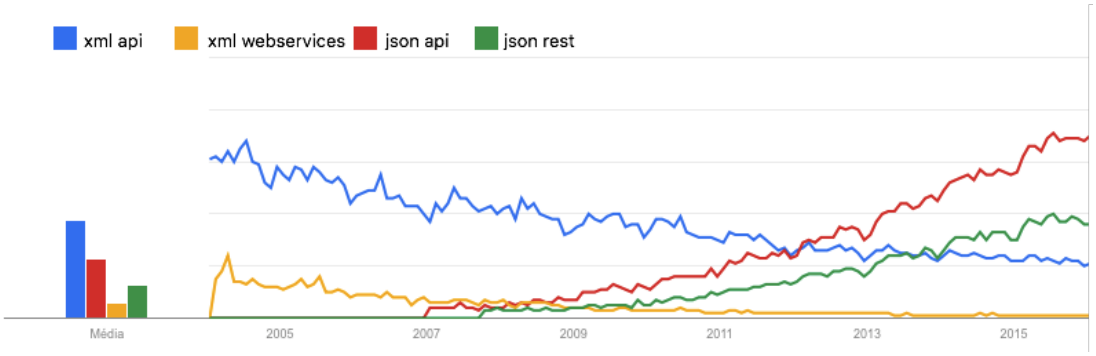

FIGURA 1.1: Interesse ao longo do tempo - XML vs JSON [38]

A principal motivação para esta dissertação é o de investigar e definir um mecanismo que permita proteger mensagens JSON e desenvolver um protótipo capaz de concorrer e/ou complementar o protocolo SSL/TLS, sendo este o principal mecanismo de segurança utilizado atualmente. Assim, o objetivo final será possibilitar uma segurança adicional a sistemas que neste momento são protegidos apenas pelo protocolo SSL/TLS.

A organização deste trabalho será dividido em duas fases, uma primeira de análise ao estado da arte e uma análise e desenvolvimento de uma solução. Na primeira fase, pretende-se analisar os mecanismos de segurança existentes para o formato JSON e compreender como as plataformas atuais estão preparadas para utilizar o sistema desenvolvido. Numa segunda fase pretende-se desenvolver e disponibilizar um protótipo do sistema definido, permitindo que outros programadores possam incluir funcionalidades de segurança JSON nos seus próprios programas/serviços *Web*. O protótipo desenvolvido será ainda submetido a uma série de testes cujo objetivo é compreender se este pode ser considerado como uma alternativa válida ao protocolo SSL/TLS.

1.2 Enquadramento

A despreocupação relativa à segurança que acompanhou o progresso tecnológico, verificado nos últimos anos, resultou em enormes prejuízos para muitas organizações [32]. A exposição a ameaças tecnológicas provoca graves consequências

(como perda de credibilidade, imagem ou danos financeiros), consequentemente as organizações definiram a segurança da informação como uma das suas maiores prioridades [32].

A incerteza sobre a segurança e privacidade patente no mundo digital é hoje inquestionável. Casos como o escândalo de Edward Snowden na NSA [22] serviram como um alerta para os perigos a que os utilizadores e as aplicações *web* estão expostos através das tecnologias da informação (TI).

A comunicação entre aplicações informáticas e/ou sistemas externos é uma necessidade constante e crescente [27]. Tecnologias como CORBA, RMI ou RPC foram e são utilizadas para transmissão de dados, no entanto estas tecnologias apresentam algumas limitações relevantes como complexidade, falta de interoperabilidade com diferentes tecnologias ou falta de padronização [27]. Estas limitações estão na origem dos serviços *Web*, que se apresentam como um conjunto de normas e métodos que permitem aplicações diferentes interagir e partilhar informação [27]. A interoperabilidade dos serviços *Web* é apoiada no uso do formato XML, permitindo que diferentes sistemas partilhem dados sem problemas associados a conversões [27].

Os avanços tecnológicos e, concretamente, a evolução dos *smartphones* provocaram uma alteração de paradigma nos serviços *Web* disponíveis, pois além de crescer o número de invocações destes serviços, os dispositivos móveis apresentam largura de banda reduzida, *plafons* de tráfego de dados reduzidos e menor poder de processamento [27]. A forma como estes serviços são invocados e como a sua resposta é processada é importante neste novo paradigma, que deverá exigir o menor processamento possível permitindo economizar largura de banda e melhorar os consumos de bateria [27].

Nos últimos anos, o formato eXtensible Markup Language (XML) tem apresentado argumentos para transmissão de estruturas computacionais e transporte de dados, sendo inclusivé o padrão definido e recomendado pela World Wide Web

Consortium (W3C) para anotação de documentos [12]. Os padrões *XML Signature [1]* e *XML Encryption [16]* definidos pela W3C oferecem mecanismos de proteção adicional sobre o XML fomentando a adoção deste formato [23].

Apesar da grande utilização do XML, como já foi referido, o formato JSON é atualmente um dos formatos mais utilizados para suportar a comunicação entre muitas das API disponibilizadas em serviços REST da *Web* (Figura 1.1). Para Crockford este crescimento deve-se, principalmente, à grande facilidade que existe em converter estruturas de dados de um programa para outro [31]. O JSON apresenta ainda um trunfo que garante uma maior aceitação por parte dos programadores, a existência de bons *parsers* funcionais em todos os principais *browsers* [23] e, consequentemente, nos *smartphones* que consomem os serviços *Web*.

Tal como acontece com o formato XML, existe a necessidade do desenvolvimento de mecanismos de segurança adicional para o formato JSON, para aumentar a granularidade da segurança e fomentar a adoção deste formato. Existem alguns projetos em desenvolvimento para utilização do JSON para transmitir chaves privadas (*JSON Web Key* [18]), assinaturas digitais (*JSON Web Signature* [19]), conteúdo (*JSON Web Encryption* [17]) e uma *framework* que utiliza estes projetos de forma a possibilitar a transmissão de dados (JOSE [20]). Estes projetos em desenvolvimento serão analisados no estado da arte desta dissertação.

Atualmente existem mecanismos que permitem assegurar a proteção e a autenticação do canal de comunicação e, consequentemente, a segurança entre serviços de Web, como por exemplo o SSL/TLS [36]. No entanto estes mecanismos cifram toda a informação que passa no canal de comunicação e da mesma forma. Assim, não é possível cifrar de forma condicional, com diferentes chaves, as mensagens de JSON, sendo que esta restrição pode ser um problema para casos de utilização específicos [36].

1.3 Questões de Investigação

Este trabalho irá dar resposta às seguintes questões de investigação:

- Quais o principais riscos de segurança inerente à utilização de mensagens JSON entre serviços web?
- Como garantir convenientemente a proteção de mensagens JSON?
- Como proteger o *payload* de uma mensagem JSON quando esta possui destinatários diferentes e esta deve ser protegida com múltiplos mecanismos de segurança?

1.4 Objetivos

Este trabalho foca-se principalmente no desenho e desenvolvimento de um mecanismo de segurança para as mensagens trocadas entre serviços *Web* baseados no formato JSON. Para tal será desenvolvido um protótipo de cifra parcial ou total do *payload* das mensagens em JSON, garantindo a confidencialidade, integridade e disponibilidade das mesmas. Com este trabalho, será possível aumentar a granularidade da segurança deste popular mecanismo de comunicação entre serviços *Web*, permitindo complementar outros já existentes.

Este protótipo será ainda incorporado numa *framework* de desenvolvimento de aplicações *Web*, ficando a sua utilização e/ou extensão disponível para qualquer programador.

Este trabalho tem ainda como objetivo perceber qual o impacto, em termos de desempenho, dos processos de cifra e decifra na troca de mensagens em serviços *Web*, quando comparados com outros mecanismos de segurança tradicionais. Por outro lado, será ainda relevante compreender como as atuais plataformas estão preparadas para utilizar este sistema.

Por fim pretende-se efetuar testes que permitam comparar a abordagem adoptada com outras alternativas de segurança já utilizadas atualmente, percebendo assim a viabilidade da mesma.

1.5 Metodologia de Investigação

Uma investigação deve ser reconhecida como sólida e relevante, tanto a nível académico como pela sociedade em geral [21]. Deve ser evidente o rigor sob o qual foi desenvolvida e que está sujeita a debate e/ou verificação [21]. Desta forma o método de investigação escolhido para este trabalho foi o *Design Science Research*, visto ser um método de pesquisa robusto que permite atingir o reconhecimento desta investigação [33].

O *Design Science Research* é motivado pelo desejo de melhorar através do desenvolvimento de novos e inovadores artefactos e processos para a construção dos mesmos [33]. O *Management Information Systems Quarterly (MISQ)* descreve a ciência do *design* como um paradigma de investigação que pode ser aplicado na área das TI [14]. Este paradigma não pode ser detalhado como um processo *standart*, mas podem ser identificados vários ciclos de investigação, nomeadamente *Relevance Cycle*, *Design Cycle* e *Rigor Cycle* (Figura 1.2).

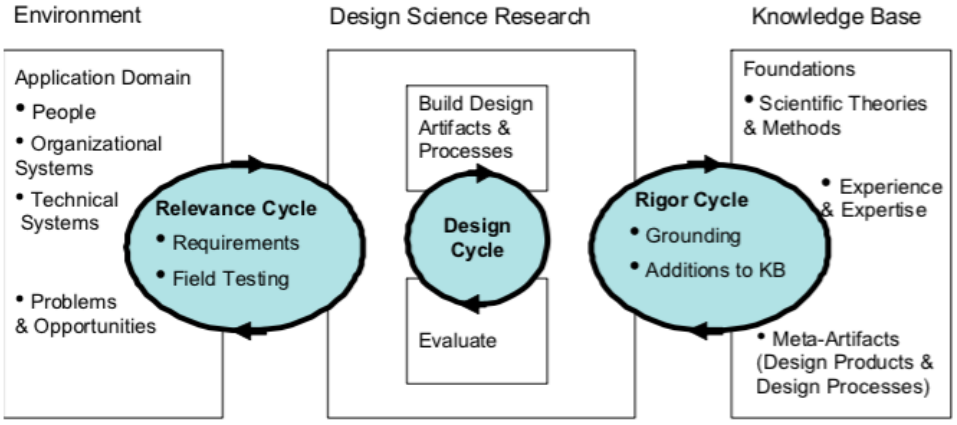

FIGURA 1.2: Design Science Research Cycles [15]

1.5.1 *Relevance Cycle*

Responsável por analisar o ambiente contextual do problema, identificação dos requisitos e oportunidades que poderemos utilizar para encontrar uma solução. Podem assim destacar-se os passos mais importantes deste ciclo:

1. Desenvolver um desenho conceptual

   O formato JSON tem crescido face às suas alternativas (principalmente face ao XML) mas não apresenta ferramentas de segurança adicional, tais como o *XML-Encryption*.

   O protótipo permite disponibilizar diversos objetos JSON, cifrados com diferentes chaves, todas para diferentes entidades recetoras, onde apenas está disponível a informação que é a si direcionada (Figura 1.3).

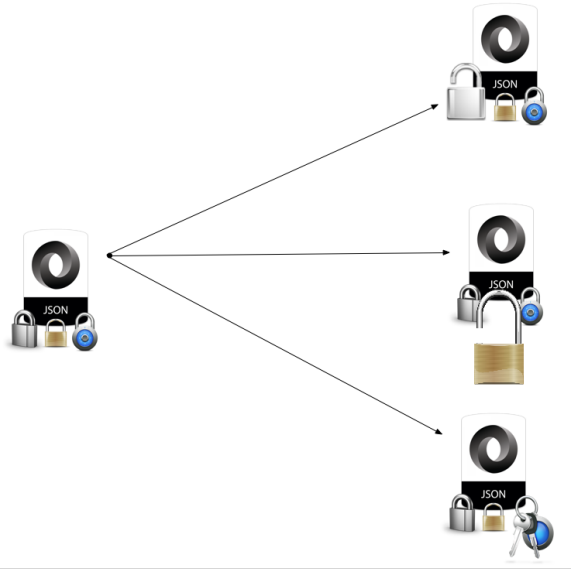

FIGURA 1.3: Transmissão de objetos JSON com várias cifras

2. Desenvolver a arquitetura do sistema

   O sistema inclui o desenvolvimento de bibliotecas em *Node.js* que permitirão efetuar múltiplas cifras sobre um ficheiro JSON e a sua disponibilização numa plataforma de distribuição de *software*, de forma a que outros programadores possam incluir estas funcionalidades nos seus próprios projetos de *software*.

1.5.2 *Design Cycle*

O ciclo *Design Cycle* pode ser visto como o coração de um projeto de *Design Science Research*. Este ciclo de atividades de pesquisa itera entre a construção de um artefacto, a sua avaliação e posterior *feedback* [15]. Este ciclo pode ser descrito como a geração de alternativas de projeto e sua avaliação relativamente aos requisitos, até alcançar um projeto satisfatório/compensatório [33].

Este ciclo está relacionado com as várias etapas de desenho, desenvolvimento e consequente validação do sistema de segurança adicional para o mecanismo JSON que irão ser documentadas nos capítulos subsequentes à análise do estado da arte.

1.5.3 *Rigor Cycle*

O *Rigor Cycle* resulta de um grande conhecimento baseado em teorias científicas e métodos de engenharia que oferecem os fundamentos para uma investigação com rigor. A base de conhecimento é composta por [15]:

- Experiências e conhecimentos que definem o *Estado da Arte* no domínio da investigação.
- Artefactos existentes e processos encontrados no domínio da investigação.

O passo que está inerente a este ciclo passa por analisar e desenhar o sistema, desenvolvendo o *Estado da Arte* para o tema atual, considerando as diferentes possibilidades para a resolução do problema.

Capítulo 2

Revisão da Literatura

A Internet é atualmente um dos principais meios para a comunicação entre as organizações [8]. A principal caracteríticas da Internet é a sua capacidade de agregar sistemas completamente distintos, que variam na sua arquitetura de máquina e/ou nos sistemas operacionais [8].

Algumas organizações estão a envolver os seus sistemas computacionais, que tipicamente não foram desenhados com o objetivo de serem interoperáveis, numa camada especial (*wrapper*) de forma a disponibilizar serviços através da Internet [2]. Estes processos originam o aparecimento dos *Aplication Service Providers*, organizações que disponibilizam serviços ou aplicações informáticas com base na *Web* [2].

2.1 Arquiteturas orientadas por serviços

O funcionamento flexível e dinâmico dos serviços *Web* apenas é possível através da utilização de um conjunto de protocolos que constituem uma Arquitetura Orientada a Serviços - *Service-oriented Architecture* (SOA) [2]. A arquitetura SOA expressa a interoperabilidade e fraca interdependência entre elementos básicos de *software* (aos quais chamamos serviços) através da definição de interfaces neutras e da utilização de protocolos padrão de transmissão de dados [34].

Principais componentes

A adoção da arquitetura SOA está diretamente relacionada com os seus principais componentes, entre eles podemos destacar:

Fraca interdependência

As aplicações desenvolvidas com base na arquitetura SOA têm, inerentemente, fraca interdependência, pois cada elemento é entidade responsável pelo desenvolvimento de uma unidade lógica de *software* que especifica a sua interface de acesso [34]. A implementação é isolada da interface, desta forma a aplicação cliente não tem acesso à implementação lógica do serviço desenvolvido [34].

Uma das grandes vantagens do desenvolvimento baseado na arquitetura SOA é a utilização do mecanismo de localização que evita a interdependência permitindo a transparência relativamente à localização do serviço [34].

Não existem restrições relativas à utilização da linguagem de programação e/ou plataforma de *software*, sendo utilizado um protocolo de aplicação independente [34].

Não existe interdependência relativa ao tipo de interações entre o cliente e o fornecedor de serviços, desta forma a arquitetura pode fornecer interações síncronas e assíncronas de acordo com as necessidades do sistema e/ou cliente [34].

Interoperabilidade

O desenvolvimento dos serviços deve utilizar tecnologias padronizadas e disponíveis entre várias plataformas de *software* [34]. Esta metodologia implica que os mecanismos de invocação (protocolos, definição de interfaces e mecanismos de localização) devem ser neutros e compatíveis com padrões largamente aceites [34].

A utilização de protocolos de aplicação independente da plataforma resulta em interoperabilidade entre serviços [34].

Composição

Os serviços *Web* podem ser desenvolvidos a partir de outros serviços já existentes. Desta forma é possível desenvolver uma arquitetura que oferece funcionalidades agregadas e com um nível superior de abstração. Esta flexibilidade na composição de novos serviços a partir de outros anteriormente disponibilizados na *Web* apresenta-se com uma das grandes vantagens deste tipo de arquiteturas. [34]

Reutilização

A reutilização dos serviços é facilitada pela forma como estes são desenvolvidos em módulos independentes e fracamente interdependentes [34]. A reutilização resulta numa alta produtividade, redução no tempo de desenvolvimento e redução de custos [34].

Granularidade

O encapsulamento[1] de funcionalidades nos serviços evoca um alto grau de granularidade na arquitetura [34].

No desenvolvimento de aplicações complexas a granularidade traz a vantagem de abstração relativa a detalhes de implementação, estes são da responsabilidade da equipa de desenvolvimento [34].

Ubiquidade

A disponibilidade dos serviços deve ser completamente assegurada, devendo estes estar acessíveis em qualquer lugar e momento de forma a facilitar a sua composição entre várias organizações. [34]

[1]Separar o programa em partes, o mais isoladas possível, tornando o software mais flexível, fácil de modificar e estender

2.2 Serviços *Web*

Os serviços *Web* surgiram como uma solução para a integração de sistemas e comunicação entre aplicações diferentes. Assim, aplicações desenvolvidas sobre linguagens de programação diferentes são traduzidas para uma linguagem de *transição* como o JSON, XML, YAML, entre outros.

Como grande parte destes *Web Services* utilizam o protocolo HTTP para comunicação. Um dos seus criadores, Roy Fielding, criou um novo padrão cuja finalidade é simplificar a criação destes serviços. Este novo padrão, guiado pelas boas práticas do uso do HTTP, é conhecido como REST [11].

2.2.1 *Remote Procedure Call* (RPC)

A tecnologia *Remote Procedure Call* é composta por um conjunto de regras para a construção de sistemas distribuídos [3]. Concretamente, a tecnologia RPC oferece um mecanismo que permite definir interfaces que poderão ser invocadas através da *Web* [3].

Cada operação RPC tem associada uma assinatura encapsulada numa estrutura chamada *Interface Definition Language* (IDL) [3]. Esta assinatura contem o nome da operação, os parâmetros de entrada, o resultado e as possíveis excepções [3].

A invocação de um sistema remoto implica a especificação do seu endereço, como cifrar os parâmetros e como decifrar o resultado para utilização de um sistema específico [3].

A primeira implementação desta tecnologia foi o *Distributed Computing Environment*, desenvolvido pela *Open Software Foundation* [3]. Posteriormente surgiram outras implementações, como o CORBA (OMG) e DCOM (Microsoft), que forçam a utilização de um protocolo específico para a comunicação entre sistemas [3].

2.2.2 *Simple Object Access Protocol* (SOAP)

O protocolo *Simple Object Access Protocol* (SOAP) é a idealização de um mecanismo RPC baseado no formato XML (descrito em 2.3.2), proposto por Dave Winer [3]. Atualmente este protocolo é uma especificação recomendada pela W3C [3].

A utilização de sistemas RPC na *Web* apresenta problemas de segurança, pois este fluxo de dados é tipicamente bloqueado por *firewalls* e/ou *proxys* [3]. Desta forma, a utilização do protocolo HTTP surge como uma solução eficaz, uma vez que este é suportado por todos os servidores *Web*, *browsers* e *firewalls* [3].

A sintaxe de uma mensagem SOAP deve ser codificada no formato XML e composta por [3]:

- SOAP Envelope

  Payload da mensagem e *namespaces* necessários à transmissão da mensagem;

- SOAP Header (opcional)

  Define como o receptor deve processar a mensagem;

- SOAP Body

  Métodos e parâmetros necessários ou respostas enviadas.

2.3 *Representational State Transfer* (REST)

Representational State Transfer (REST) é uma abstração da arquitetura da WWW, concretamente definindo-se como um estilo de arquitetura para sistemas distribuídos.

Apesar do padrão REST ser baseado nos conceitos do protocolo HTTP, existem regras que devem ser seguidas para realizar um uso efetivo do protocolo.

Podemos destacar algumas das principais características do REST:

- Arquitetura Cliente/Servidor

  Esta arquitetura procura separar as responsabilidades, onde o servidor fica responsável por oferecer um conjunto de serviços que serão invocados pelo(s) cliente(s), independente da sua linguagem [10].

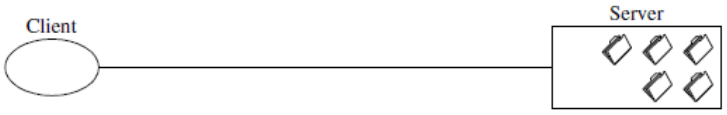

FIGURA 2.1: Client-Server [10]

- *Stateless*

  As interações entre Cliente/Servidor são *client-stateless-server*, ou seja, em cada pedido será enviada toda a informação necessária para a sua resposta. Desta forma obtemos benefícios como visibilidade, confiabilidade e escalabilidade [10].

- *Cache*

  Uma *cache* no lado do cliente melhora o desempenho de rede e elimina parcialmente as interações Cliente/Servidor, melhorando a eficiência, escalabilidade e desempenho [10].

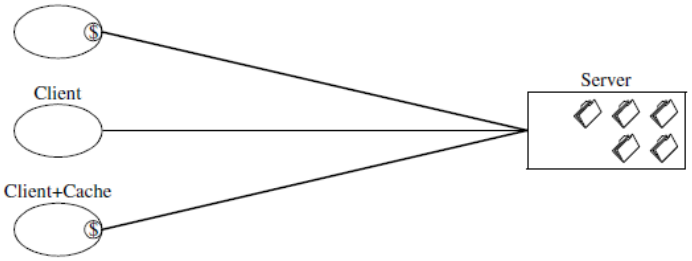

FIGURA 2.2: Client-Cache-Stateless-Server [10]

- Interface Uniforme

  A Interface Uniforme presente no REST é um estilo que define quatro princípios: identificação de recursos, manipulação de recursos através de representações, mensagens descritivas e recursos de hipermédia[2] [10].

[2]Suporte de difusão de informação apresentada sob a forma de texto, gráficos, áudio ou vídeo num sistema de hipertexto

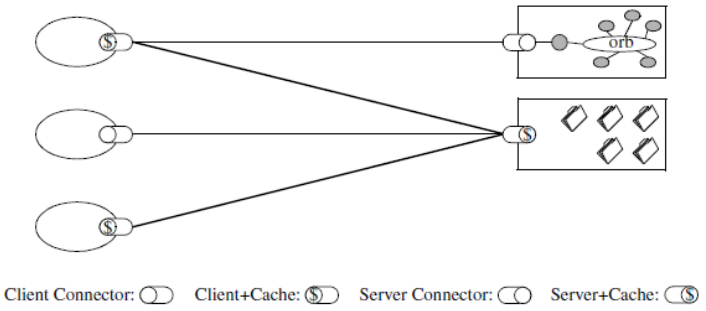

FIGURA 2.3: Uniform-Client-Cache-Stateless-Server [10]

- *Code-on-demand*

  Paradigma de sistemas distribuídos cuja finalidade é oferecer a possibilidade de executar código existente no servidor pelo cliente (i.e. *JavaScript*) [10].

No entanto, a especificação do REST não define as tecnologias utilizadas ou forma como a implementação do *Web Service* deve seguir, apenas define as características que este deve apresentar.

Existem variações apresentadas (i.e. RESTful) que adiciona definições aos protocolo REST, como a utilização do protocolo HTTP para transporte de dados, de recursos (utilizando URIs), de métodos HTTP (*GET, POST, PUT, DELETE*) para determinar ações sobre os recursos e a independência na representação de dados.

2.3.1 *JavaScript Object Notation* (JSON)

O JSON é um formato de texto para a transmissão de dados estruturados que deriva dos *object literals* do JavaScript [12]. Permite representar um conjunto de tipos primários (cadeias de caracteres, números, booleanos e nulos) e dois tipos estruturados (vetores e objetos). Um vetor é uma sequência ordenada de zero ou mais valores, onde um valor é qualquer tipo primário ou estruturado. Um objeto é uma sequência não ordenada de zero ou mais pares nome/valor, onde o nome é uma cadeia de caracteres [6].

Este formato é intensivamente utilizado para codificar a informação durante transferências entre servidores e aplicações *AJAX*[3], serviços *Web* ou em muitos cenários similares [31].

As principais vantagens são a sua simplicidade, portabilidade e utilização de texto simples. Desta forma, facilita a leitura/escrita por humanos e interpretação/geração através de máquinas, como podemos observar no exemplo 2.1.

```
{
    "Alunos" : [
        { "Nome": "Ana", "Notas": [15, 14, 12 ]  },
        { "Nome": "Maria", "Notas": [ 8, 10, 14 ] },
        { "Nome": "Pedro", "Notas": [ 20, 20, 18 ] }
    ]
}
```

Código Fonte 2.1: Exemplo JSON

Possui ainda uma grande vantagem na transmissão da informação, sendo um subconjunto do *JavaScript* permite transportar um objeto complexo e realizar a sua conversão para *JavaScript* de forma simples (por exemplo, através da função *Javascript "eval"*) [12].

Desta forma o JSON é cada vez mais utilizado para a transmissão de dados no contexto da Internet, em detrimento do XML. Além destas características o JSON conta ainda com um trunfo importante - a existência de bons *parsers* funcionais em todos os principais *browsers Web*, facto que permite o uso eficiente de transmissão no desenvolvimento de aplicações *Web* [12].

O JSON é uma escolha típica em ambientes onde o fluxo de dados entre o cliente e o servidor é de grande importância, onde a fonte dos dados tem de ser confiável e não pode existir perda dos recursos de processamento do lado do cliente (para manipulação de dados ou geração de interface). Este é o tipo de ambientes onde os grandes *players* tecnológicos (Google, Yahoo, entre outros) estão inseridos sendo esta a razão pela qual utilizam este formato.

[3]Asynchronous JavaScript XML (AJAX) é um conjunto de tecnologias utilizadas com o objetivo de tornar as páginas *Web* mais iterativas para o utilizador através de solicitações assíncronas [13]

Alternativas ao JSON

Existem dezenas de alternativas ao JSON (por exemplo XML, YAML, Protocol Buffers, etc.), mas a maioria delas não tem a adesão necessária para se tornarem largamente difundidas e utilizadas [12].

A grande alternativa ao JSON é o XML, sendo o formato mais utilizado nos sistemas de informação. Nos últimos anos, além do JSON, também o YAML se tem destacado [12].

2.3.2 *eXtensible Markup Language* (XML)

O formato *eXtensible Markup Language* (XML) tem sido adotado nas mais diversas áreas (por exemplo, na Justiça, nas Finanças, Comunicações, entre outros). O XML é a escolha padrão para novos formatos de documentos na maioria das aplicações informáticas, sendo utilizado na maioria dos contextos [12].

O XML é o formato *standard* definido e recomendado pela W3C na anotação de documentos, que utiliza texto simples para codificar um conjunto hierárquico de informação [25]. Entre as principais vantagens que o XML apresenta, podemos destacar a sua flexibilidade, robustez e relativa simplicidade, razões que levaram à sua rápida aceitação por todo o tipo de entidades [12].

Apesar do JSON ser normalmente posicionado como um *"concorrente"* do XML, não é invulgar verificar a sua utilização na mesma aplicação.

O exemplo 2.2 representa a mesma informação apresentada no exemplo 2.1.

```
<?xml version="1.0" encoding="UTF-8" ?>
    <Alunos>
        <Nome>Ana</Nome>
        <Notas>15</Notas>
        <Notas>14</Notas>
        <Notas>12</Notas>
    </Alunos>
    <Alunos>
        <Nome>Maria</Nome>
        <Notas>8</Notas>
```

```
        <Notas>10</Notas>
        <Notas>14</Notas>
    </Alunos>
    <Alunos>
        <Nome>Pedro</Nome>
        <Notas>20</Notas>
        <Notas>20</Notas>
        <Notas>18</Notas>
    </Alunos>
```

Código Fonte 2.2: Exemplo XML

Apesar das inegáveis vantagens que o XML trouxe ao mundo da representação e transferência de dados, também apresenta pontos fracos, dos quais são exemplo [12]:

1. Sintaxe redundante
2. A construção de *parsers* não é tão trivial como aparenta
3. Requisitos básicos de processamento não suportam alguns tipos de dados[4]
4. Apesar de ser relativamente fácil para os humanos ler um ficheiro XML, a sua sintaxe não é de fácil aplicação

2.3.3 YAML

O significado do acrónimo deste formato (YAML) não é unânime, existem dois significados, aparentemente contraditórios, *Yet Another Markup Language* e o mais recursivo *YAML Ain't Markup Language* [35]. O YAML oferece o que a maioria das linguagens de notação oferecem, com uma notação mais simples.

O YAML é, tal como o JSON e XML, um formato de transmissão de dados legível por humanos [12]. Este formato foi inspirado em linguagens como XML, C, Python, Perl e no formato de correio eletrónico especificado no RFC 2822 [12]
.

[4] i.e. Não existe nenhum mecanismo XML para que 2,71828 seja um número de vírgula flutuante em vez de um conjunto de caracteres

O YAML pode ser visto como uma simplificação do XML, pois quase tudo o que é possível representar em XML pode ser representado com YAML, com a vantagem de possuir uma forma mais compacta [12]. Uma das maiores vantagens do YAML está na transmissão da informação, pois as suas características permitem, tal como o JSON, representar facilmente os tipos de dados mais comuns das linguagens de programação [12].

Utilizando, uma vez mais, a mesma informação utilizada nos exemplos de JSON (exemplo 2.1) e XML (exemplo 2.2), o exemplo 2.3 mostra o aspeto deste formato.

```
---
  Alunos:
    -
      Nome: "Ana"
      Notas:
        - 15
        - 14
        - 12
    -
      Nome: "Maria"
      Notas:
        - 8
        - 10
        - 14
    -
      Nome: "Pedro"
      Notas:
        - 20
        - 20
        - 18
```

Código Fonte 2.3: Exemplo YAML

2.4 Segurança para serviços *Web*

Atualmente, a maioria das transações comerciais dependem de *Web Services*, razão pela qual se tornaram numa das áreas mais importantes da indústria das TI, sendo a segurança presente neste tipo de transações fundamental para garantir o sucesso de uma organização [32].

A possibilidade de um utilizador interagir diretamente com os sistemas das organizações levanta problemas de segurança. Como podemos garantir que as informações do utilizador chegam ao sistema final de forma segura, sendo que estas informações sensíveis ao utilizador são encaminhadas pela *Web* [8].

Existem diferentes protocolos e tecnologias para garantir a segurança e confidencialidade na Internet, sendo que cada uma utiliza diversas maneiras de proteger a informação. Um dos protocolos mais utilizados é o SSL/TLS (descrito na secção 2.4.2) que procura proteger as comunicações efetuadas através da Internet [36]. No entanto, este protocolo garante a segurança apenas durante a fase de transporte, sendo que uma mensagem, para chegar ao seu destinatário, pode passar por diversos nós intermédios ao nível aplicacional (Figura 2.4) [8].

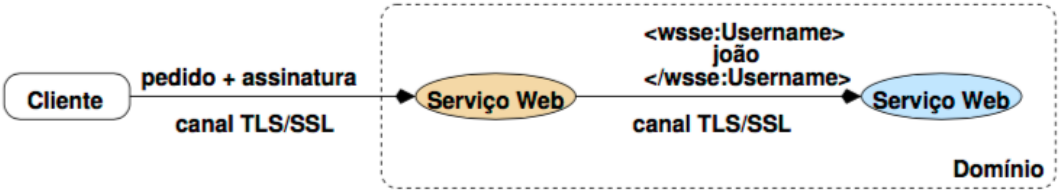

FIGURA 2.4: Encaminhamento de um pedido *Web* ao nível aplicacional [8]

2.4.1 *Web Services Security* (WS-Security)

Web Services Security (ou WS-Security) é um extensão de segurança do protocolo SOAP para serviços *Web*. O WS-Security surgiu como uma iniciativa conjunta das empresas IBM e Microsoft destinada à utilização das tecnologias *XML Signature* e *XML Encryption* oferecendo assim segurança *end to end* às mensagens SOAP [8].

O WS-Security define três mecanismos fundamentais [8]:

- Credenciais de segurança:

  As credenciais de segurança com informações de autenticação devem estar incluídas na mensagem SOAP;

- Integridade da mensagem:

Informações relacionadas a assinaturas digitais de toda ou de partes da mensagem devem ser incluídas na mensagem SOAP;

- Confidencialidade da mensagem:

  As mensagens SOAP podem ser parcial/totalmente cifradas.

2.4.1.1 *XML Signature* (XMLDSign)

A utilização de assinaturas digitais é uma forma de garantir a integridade e autenticidade de informações digitais. A especificação *XML Signature* define regras para gerar e validar assinaturas digitais sob o formato XML [8].

A XMLDSign permite assinar qualquer tipo de documento digital (binários ou texto), no entanto a assinatura será representada através de uma estrutura XML [8]. Também é possível assinar apenas uma parte de um documento XML, desta forma será possível alterar o mesmo sem que isso invalide a parte assinada [8].

As assinaturas podem ser representadas de três formas diferentes (figura 2.5) [8]:

- *enveloped*

  Assinatura contida no próprio documento XML, sendo a mais indicada para utilização em serviços *Web*, inserida nas mensagens SOAP.

- *enveloping*

  Os dados assinados ficam contidos na estrutura da XMLDSign.

- *detached signature*

  Assinatura separada dos dados assinados.

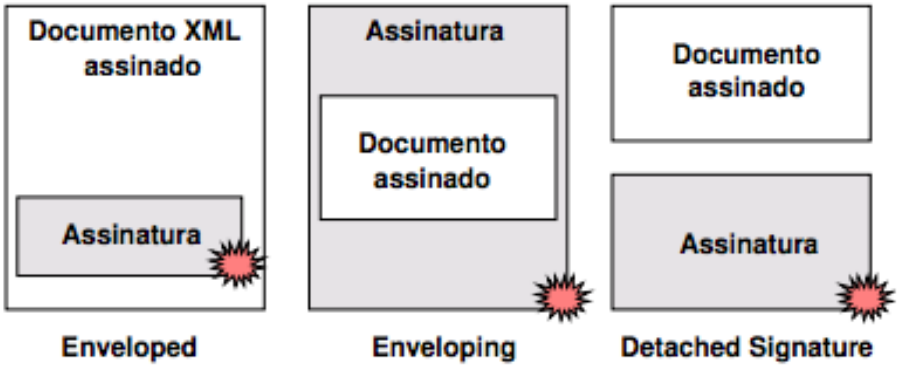

FIGURA 2.5: Formas de assinaturas XMLDSign [8]

2.4.1.2 *XML Encryption*

O *XML Encryption* é um mecanismo amplo de segurança adicional ao conteúdo da mensagem XML, que especifica o processo para a cifra dos dados e a sua representação em formato XML ou conteúdos de elementos de XML [25]. Contudo este mecanismo exige um processamento extra que afeta diretamente a performance do processo de transmissão [25].

Um documento XML cifrado com XML Encryption pode ser visto por qualquer utilizador, mas apenas os proprietários das chaves poderão compreender o(s) conteúdo(s) cifrado(s). Desta forma é possível utilizar múltiplas cifras que se destinam a diferentes entidades.

O *XML-Encryption* pode encriptar um documento XML, um elemento XML ou o próprio conteúdo do elemento [23]. O resultado deste processo é um elemento XML *EncryptedData*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Alunos>
    <Nome>Ana</Nome>
    <EncryptedData
        xmlns='http://www.w3.org/2001/04/xmlenc#'
Type='http://www.w3.org/2001/04/xmlenc#Element'/>
        <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
        <ds:KeyInfo
            xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
            <ds:RetrievalMethod URI='#EK'
```

```
Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
            <ds:KeyName>Ana</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
            <CipherValue>DEADBEEF</CipherValue>
        </CipherData>
    </EncryptedData>
</Alunos>
<Alunos>
    <Nome>Maria</Nome>
    <EncryptedKey Id='EK'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <ds:KeyInfo
            xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
            <ds:KeyName>Maria</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
            <CipherValue>xyzabc</CipherValue>
        </CipherData>
        <ReferenceList>
            <DataReference URI='#ED'/>
        </ReferenceList>
        <CarriedKeyName>Sofia</CarriedKeyName>
    </EncryptedKey>
</Alunos>
<Alunos>
    <Nome>Pedro</Nome>
    <Notas>20</Notas>
    <Notas>20</Notas>
    <Notas>18</Notas>
</Alunos>
```

Código Fonte 2.4: Exemplo XML Encryption

No exemplo 2.4 podemos observar um resultado do *XML Encryption* sobre o exemplo 2.2, neste caso foram aplicadas duas cifras com diferentes algoritmos, um simétrico (AES 128) e um assimétrico (RSA 1.5), sobre as tags *"Notas"* de dois alunos.

As tags resultantes do processo contêm informação fundamental que serão utilizadas do processo de decifra, como algoritmo utilizado, localização do URI da chave ou identificação de propriedade da chave.

2.4.2 SSL/TLS

O protocolo *Secure Sockets Layer/Transport Layer Security* (SSL/TLS) consiste num conjunto de mensagens e regras sobre quando estas devem ou não ser enviadas [36]. Este protocolo define dois papéis diferentes para as partes que comunicam. Um sistema é sempre um cliente, enquanto que o outro é sempre o servidor. A distinção é muito importante porque o protocolo requer que estes dois se comportem de formas diferentes. No uso mais comum deste protocolo, *Secure Web Browsing*, o *Web Browser* é o cliente e o *Web site* o servidor [36].

A principal funcionalidade do protocolo SSL/TLS é estabelecer um canal cifrado e autenticado de comunicação entre o cliente e servidor [36].

O protocolo SSL/TLS concentra-se na proteção da informação durante o transporte entre duas entidades, sendo esta imediatamente decifrada na chegada ao *endpoint*, independentemente do seu destinatário final.

Caso o canal seja comprometido toda a informação transmitida poderá ficar acessível para um atacante.

Num cenário onde existe informação sensível redirecionada por múltiplas partes, sem que as mesmas sejam o destinatário final da informação, se uma das partes for comprometida toda a informação poderá ficar exposta. Neste cenário a proteção dos canais com o protocolo SSL/TLS é insuficiente (figura 2.6).

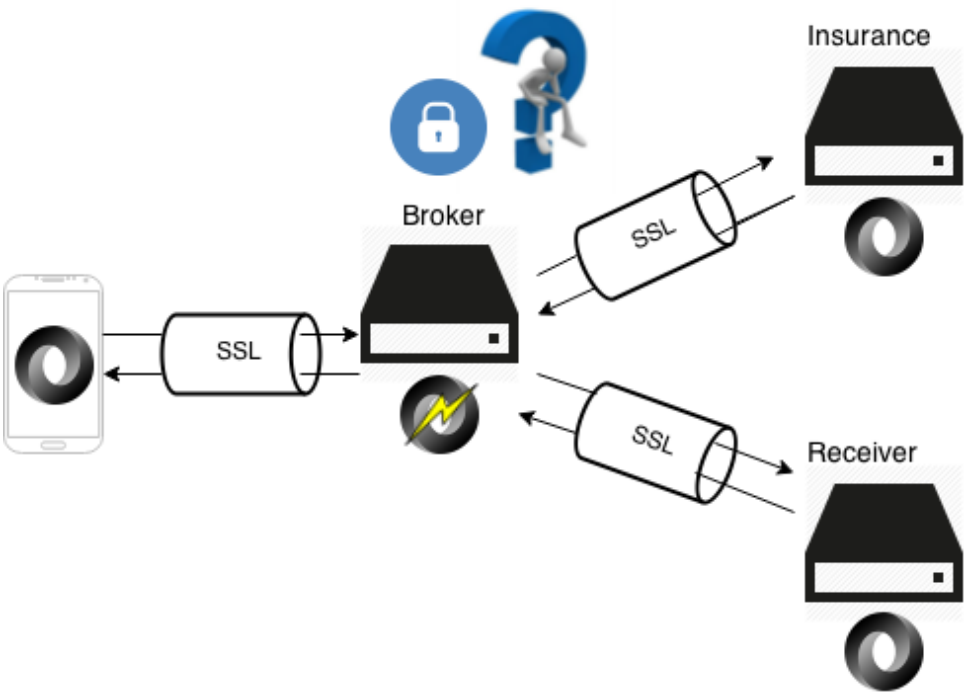

FIGURA 2.6: Problema SSL/TLS

Desta forma, não é suficiente proteger apenas o canal de transmissão, é necessário proteger a própria informação crítica. Sabendo que o XML é o formato definido e recomendado pela W3C e um dos mais utilizados nos *Web Services* existentes, possui vários mecanismos de segurança para proteger o seu *payload*.

2.4.3 Segurança para JSON

Com o recente crescimento da utilização do JSON (Figura 1.1), principalmente nos *Web Services* REST, aumenta também a necessidade de mecanismos de segurança adicional [8].

Apesar de ser um crescimento relativamente recente, já existem alguns mecanismos de segurança em desenvolvimento, concretamente *JSON Web Encryption*, *JSON Web Signature*, *JSON Web Key* e *Javascript Object Signing and Encryption*.

2.4.3.1 *JSON Web Encryption*

O *JSON Web Encryption* (JWE) é um padrão de segurança que representa o conteúdo cifrado utilizando estruturas de dados JSON e encoding *Base64* [17].

Uma estrutura JWE é composta através dos valores lógicos seguintes [17]:

- JOSE Header

  Para um elemento JWE, o *JOSE Header* é composto pelos seguintes elementos:

  - JWE Protected Header

    Esta estrutura JSON contém os parâmetros que garantem a integridade e proteção através da operação de cifra autenticada [17]. Estes parâmetros aplicam-se a todos os destinatários da JWE [17].

  - JWE Shared Unprotected Header

    Estrutura JSON que contém os parametros que se aplicam a todos os destinatários cuja integridade está desprotegida [17].

  - JWE Per-Recipient Unprotected Header

    Estrutura JSON que contém os parametros que se aplicam unicamente a um destinatário da JWE. Estes não estão integralmente protegidos [17].

- JWE Encrypted Key

  Conteúdo da chave criptográfica cifrada. Em alguns algoritmos este valor poderá ser uma sequência vazia de octetos [17].

- JWE Initialization Vector

  Valor de inicialização do vector usado para cifrar o *payload*. Alguns algoritmos podem não utilizar um vector de inicialização, nestes casos este valor será uma sequência vazia de octetos [17].

- JWE AAD

  Valor adicional que será integralmente protegido pela operação de criptografia autenticada [17].

- JWE Ciphertext

  Valor cifrado resultante da criptografia autenticada do *payload* com os dados adicionais autenticados [17].

- JWE Authentication Tag

  Valor da *tag* de autenticação resultante da criptografia autenticada do *payload* com os dados adicionais autenticados [17].

Este padrão utiliza autenticação cifrada assegurando a confidencialidade e integridade dos dados e a integridade das estruturas *JWE Protected Header* e *JWE AAD* [17].

No reverso da medalha, esta proteção exige uma expansão drástica do seu tamanho, uma vez que será necessário adicionar toda a informação indispensável à sua decifra [29].

2.4.3.2 *JSON Web Signature*

JSON Web Signature (JWS) é um formato que utiliza estruturas de dados JSON para representar um conteúdo protegido com um código de *Hash*, concretamente *Hash based Message Authentication codes* (HMACs) ou esquemas de assinaturas digitais, tais como ECDSA [30].

Uma estrutura JWS é composta em três partes:

1. Cabeçalho

   Descreve o algoritmo HMAC ou assinatura utilizada.

2. *Payload*

   Conteúdo protegido ou seja a assinatura permitirá saber se este conteúdo foi ou não alterado após a sua assinatura.

3. Assinatura JWS

   Resultado da aplicação do algoritmo criptográfico ou HMAC sobre o cabeçalho e *payload*.

2.4.3.3 *JSON Web Key*

O formato *JSON Web Key* (JWK), de forma semelhante ao JWS, utiliza estruturas de dados JSON para representar chaves públicas [30]. A representação JWK de uma chave pública consiste num objeto JSON cujos objetos membros representam uma chave pública, tal como o algoritmo criptográfico utilizado [30].

As estruturas JWK são utilizadas na especificação dos formatos JWS e JWE [18].

2.4.3.4 *Javascript Object Signing and Encryption*

Javascript Object Signing and Encryption (JOSE) é uma *framework* que possibilita a transmissão de dados, de forma segura, entre as partes envolvidas, utilizando, entre outros, os formatos referenciados anteriormente (JWE, JWS e JWK) e um conjunto de especificações que permitem atingir esta transmissão [20].

Um dos principais objetivos, para o qual o JOSE foi desenhado, é a possibilidade de funcionar com um meio de autenticação e autorização em sistemas que estejam expostos na *Web* [20].

2.5 Plataformas Desenvolvimento

De forma a alcançar os objetivos deste trabalho, serão analisadas algumas plataformas de desenvolvimento mais utilizadas segundo o ranking divulgado pela TIOBE. A empresa TIOBE Index publica mensalmente a popularidade de todas as linguagens de programação através de várias métricas definidas [37].

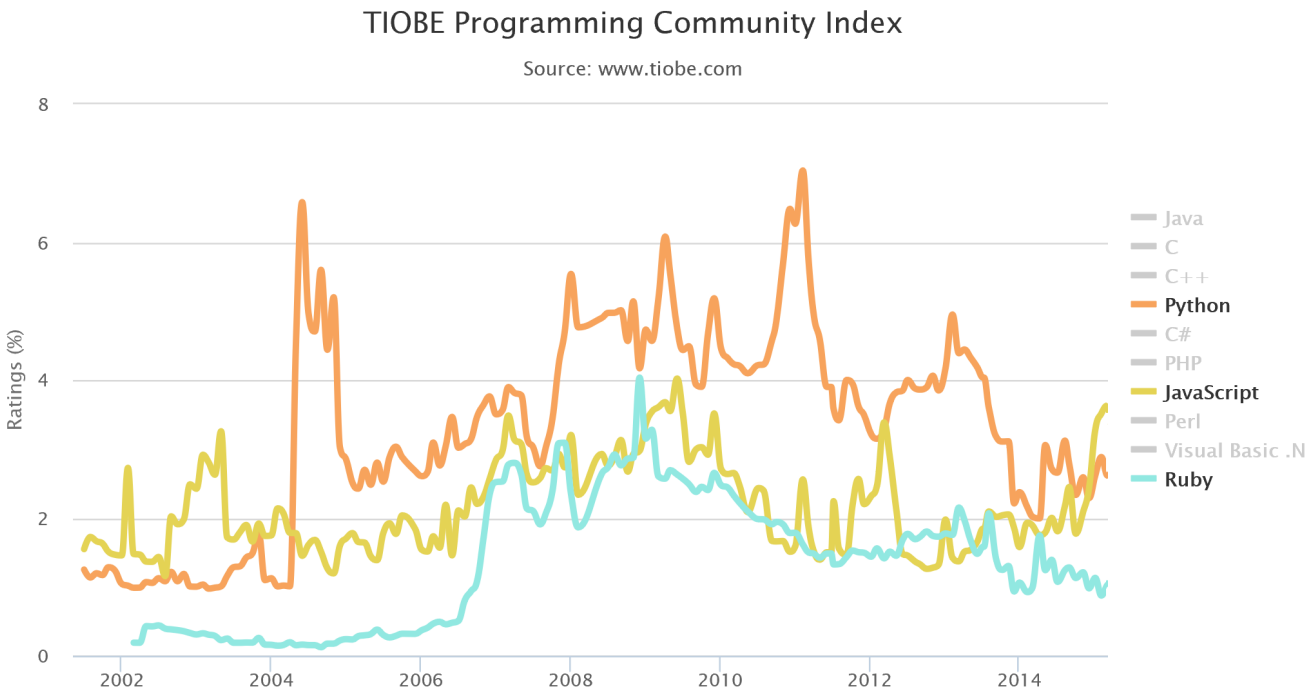

FIGURA 2.7: Ranking TIOBE Julho 2015 [37]

Através do ranking de Julho de 2015 (Figura 2.7) foram escolhidas três entre as dez mais populares, nomeadamente Python, Ruby e Node.js (tecnologia baseada em *JavaScript*). Além de um conjunto de características específicas que serão abordadas nas próximas secções, estas linguagens têm em comum a possibilidade de uma rápida aprendizagem, requisito fundamental para o desenvolvimento deste trabalho.

2.5.1 Python

Python é uma linguagem de *scripting opensource* de alto nível, orientada a objetos, dinâmica, interpretada e interativa [4]. Possui uma sintaxe clara e concisa, fomentando a legibilidade e manutenção do código fonte [4].

As principais vantagens do *Python* são [4]:

- Programação funcional;
- Interoperabilidade com outras linguagens;

- Multiplataforma.

Python Package Index

Python Package Index (também conhecido como PYPI) é, tal como o nome sugere, o gestor de pacotes da plataforma Python, responsável pelas dependências e respectivas atualizações [4].

Esta gestor permite agilizar os processos de instalação e atualização de outras bibliotecas desenvolvidas em Python, libertando os programadores deste tipo de tarefas ao mesmo tempo que fomenta a sua utilização.

2.5.2 Ruby

Ruby é uma linguagem de programação de *scripting* interpretada, *open-source* e orientada a objetos [7].

A linguagem *Ruby* é descrita como transparente, possibilitando expressar as ideias em blocos de código simples, facilmente legíveis e sem muitos casos especiais ou excepções [5].

As principais vantagens do *Ruby* são [24]:

- Procura de métodos dinâmica, isto significa que quando um método não é encontrado no contexto em que está inserido, este será procurado no contexto da sua superclasse;
- Variáveis sem tipo definido, isto significa que, à semelhança do *Python*, uma variável pode assumir vários tipos ao longo do código;
- Gestão de memória;
- Utilização de *threads*.

Ruby Gems

RubyGems é, à semelhança do *Python Package Index*, um gestor de pacotes para a linguagem *Ruby* que oferece um formato padrão para a distribuição de programas escritos em *Ruby* (formato *gem*), uma forma simples de instalar *gems* e um servidor para os disponibilizar [28].

2.5.3 Node.js

O *Node.js* (ou simplesmente *Node*) é uma plataforma *open-source* para *server-side* e aplicações web. As aplicações *Node* são escritas em *JavaScript*. Esta plataforma possui uma arquitetura orientada a eventos e um I/O API não bloqueante, que proporciona um aumento de performance e escalabilidade de uma aplicação.

As principais vantagens do *Node* são [26]:

- Desenvolvimento *web* numa linguagem dinâmica, *JavaScript* apresenta uma performance superior a outras linguagens, como *Ruby*, *Python* ou *Perl*;
- *JavaScript* é uma linguagem muito popular e relativamente fácil de se aprender;
- Possibilidade de lidar com milhares de ligações simultâneas com recurso a pouco memória e a um único processo.

Node Package Manager

Além das vantagens já mencionadas, o *Node* conta ainda com o *Node Package Manager* (NPM).

O NPM é um gestor de bibliotecas para *JavaScript* e o gestor padrão para *Node* que vem incluído na sua instalação desde a versão 0.6.3. Este gestor permite que novas bibliotecas sejam disponibilizadas aos programadores, facilitando a reutilização de código para um desenvolvimento eficiente.

2.6 Análise à literatura

Após a análise desenvolvida nas seções anteriores podemos concluir que as preocupações de segurança nos sistemas de informação tem uma grande importância dentro da maioria das organizações atuais.

No caso concreto do formato JSON, que registou um grande crescimento relativamente às suas principais alternativas, apresenta poucos mecanismos de segurança, sendo que estes ainda estão em fase de desenvolvimento e/ou maturação. A principal ferramenta utilizada é, ainda, o protocolo SSL/TLS, responsável por proteger o transporte das mensagens JSON.

Desta forma, nos próximos capítulos será estudado, desenhado, desenvolvido e validado uma proposta de um novo mecanismo de segurança para o formato JSON, baseado no mecanismo de segurança mais utilizado pela sua principal alternativa, concretamente o *XML Encryption*. A semelhança com um mecanismo recomendado pela W3C e amplamente utilizado pode fomentar a sua utilização e, consequentemente, contribuir para uma maior segurança em sistemas que utilizam o formato JSON. Este mecanismo poderá funcionar como uma alternativa e/ou complemento ao protocolo SSL/TLS.

Capítulo 3

Desenho

Ao longo deste capítulo será descrita a especificação, desenvolvida nesta dissertação, para cifrar uma mensagem JSON e a representação deste processo no formato JSON. Este capítulo está dividido em duas secções que apresentam e justificam a fase de concepção de um mecanismo de segurança para o formato JSON, baseado no mecanismo *XML Encryption*. Desta forma, para alcançar os objetivos definidos na secção 1.4 esta investigação foi dividida em duas fases, cada uma das quais apresentada na sua secção específica.

A definição de um mecanismo de segurança adicional para o formato JSON e os respectivos requisitos que suprimam a necessidade de um mecanismo similar ao XML Encryption, identificada nas secções anteriores, será apresentado na secção 3.1.

A definição de uma sintaxe composta por um conjunto de regras necessárias para os processos criptográficos e algoritmos utilizados no mecanismo definido anteriormente será apresentada na secção 3.2.

No final deste capítulo será possível realizar uma implementação deste mecanismo, com base nas regras e especificações definidas.

3.1 *Secure JSON* (SecJSON)

O *Secure JSON* será definido por um conjunto de regras e especificações para cifrar informação e representar o seu resultado no formato JSON. A informação pode ser um documento JSON, um tipo primário ou um tipo estruturado. O resultado do processo de cifra é um elemento *EncryptedData* SecJSON que contém a informação cifrada.

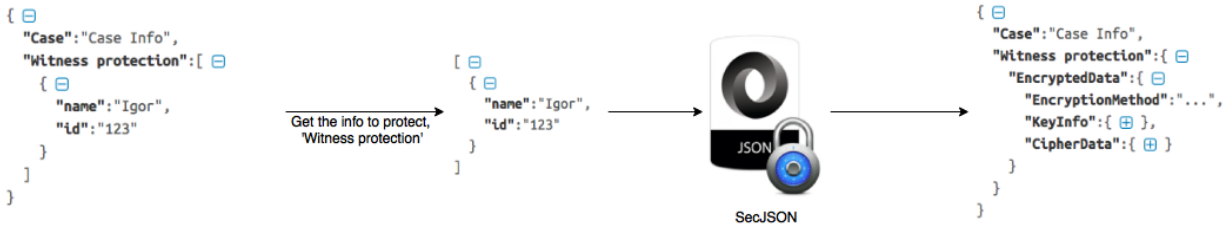

FIGURA 3.1: Processo SecJSON para tipos primários/estruturados

Após o processo de cifra do SecJSON sobre um elemento primário ou estruturado do JSON, este é substituído pelo elemento *EncryptedData* respetivo (Figura 3.1).

FIGURA 3.2: Processo SecJSON para um documento JSON

Quando o processo é aplicado ao documento JSON o resultado é um documento com um único elemento *EncryptedData* (Figura 3.2).

3.1.1 Requisitos

Com base nos objetivos definidos anteriormente, é importante definir concretamente quais os requisitos que irão guiar o desenvolvimento do trabalho apresentado nesta dissertação.

1. Estudo e definição de uma sintaxe composta por um conjunto de regras que permitem a cifra e decifra parcial ou completa de um objeto JSON com uma ou múltiplas chaves criptográficas.

2. Implementação de um protótipo do mecanismo definido anteriormente.

   (a) Desenvolvimento de módulos que permitem cifrar e decifrar informação através de algoritmos criptográficos assimétricos.

   (b) Desenvolvimento de um módulo que percorrem um objecto JSON através de expressões regulares.

   (c) Permitir cifrar um objeto JSON com múltiplas chaves, destinadas a múltiplos receptores, onde cada um apenas tem acesso à informação a si dirigida;

   (d) Possibilitar uma fácil extensão do protótipo através da implementação de novos algoritmos criptográficos.

   (e) Garantir tempos de resposta nos processos criptográficos similares aos tempos apresentados pelo protocolo SSL/TLS.

   (f) Inserção do protótipo numa plataforma de distribuição de software *open source*.

   (g) Desenvolvimento de um manual de utilização do protótipo e disponibilização do mesmo.

3.2 Sintaxe

Nesta secção é apresentada e definida a sintaxe do mecanismo SecJSON, composta por um conjunto de elementos que são essenciais para os processos de cifra e decifra definidos nas regras de processamento. Os elementos da sintaxe SecJSON, definidos posteriormente de forma individual, são os seguintes:

- Elemento abstrato *EncryptedType*

  - Elemento *EncryptedData*
  - Elemento *EncryptedKey*

- Elemento *EncryptionMethod*
- Elemento *CipherData*
- Elemento *CipherReference*
- Elemento *KeyInfo*
- Elemento *RetrievalMethod*

Os tipos de dados que serão apresentados para especificar os elementos referidos anteriormente são os seguintes:

- **string**

  O conteúdo deste elemento é uma sequencia de caracteres;

- **integer**

  O conteúdo deste elemento é um numero inteiro.

- **object**

  O conteúdo deste elemento é um objeto JSON que será detalhado através dos pares *properties* e *required*.

  Os elementos cujo conteúdo é apresentado na forma *"$ref": "/definitions/nomeDoElemento/"* serão especificados mais à frente, numa seção específica.

3.2.1 Elemento abstrato *EncryptedType*

EncryptedType é um tipo abstrato do qual derivam os elementos *EncryptedData* e *EncryptedKey*. Apesar do conteúdo destes dois elementos ser muito similar, uma diferença sintática é útil para o processamento. A implementação do SecJSON deverá criar elementos *EncryptedType* e descendentes.

Apesar do JSON Schema não suportar elementos abstratos, uma representação deste elemento é útil para facilitar a interpretação da sintaxe.

A implementação deve gerar uma estrutura aceite pelo JSON *Schema* definido. O *schema* do SecJSON está disponível em:

- http://tiagomistral.github.io/SecJSON/secjson-schema.json.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"EncryptedType",
    "type":"object",
    "abstract": true,
    "properties":{
        "EncryptionMethod":{
            "$ref":"#/definitions/EncryptionMethod"
        },
        "KeyInfo":{
            "$ref":"#/definitions/KeyInfo"
        },
        "CipherData":{
            "$ref":"#/definitions/CipherData"
        }
    },
    "required":[
        "CipherData"
    ]
}
```

Código Fonte 3.1: Elemento abstrato *EncryptedType*

3.2.2 Elemento *EncryptionMethod*

O elemento *EncryptionMethod* é um elemento opcional que define o algoritmo aplicado aos dados cifrados. Se este elemento estiver ausente, o algoritmo deverá ser conhecido pelo destinatário da mensagem.

```
JSON Schema Definition:

{
```

```
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"EncryptionMethod",
    "type":"object",
    "properties":{
        "Algorithm":{
            "type":"string"
        },
        "KeySize":{
            "type":"integer"
        },
        "OAEPparams":{
            "type":"string"
        }
    },
    "required":[
        "Algorithm"
    ]
}
```

Código Fonte 3.2: Elemento *EncryptionMethod*

3.2.3 Elemento *CipherData*

O elemento *CipherData* contém os dados cifrados. Este elemento deve ainda conter o conteúdo do elemento *CipherValue* cifrado e codificado numa sequência em formato *Base64*.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"CipherData",
    "type":"object",
    "properties":{
        "CipherValue":{
            "properties":{
                "CipherValue":{
                    "type":"string"
                }
            },
            "required":[
                "CipherValue"
            ]
        }
```

```
    }
}
```

Código Fonte 3.3: Elemento *CipherData*

3.2.3.1 Elemento *CipherReference*

Caso o conteúdo do elemento *CipherValue* não seja fornecido diretamente, o elemento *CipherReference* identifica uma fonte que quando processada, produz a sequência de octetos cifrada.

A obtenção do valor real ocorre através do identificador presente no *CipherReference URI*, cujo valor funciona como referência para a fonte. Caso o elemento possua uma sequência de transformações (elemento *Transforms*), os dados obtidos na fonte são transformados de acordo com a especificação, obtendo-se assim a cifra pretendida.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"CipherReference",
    "type":"object",
    "properties":{
        "URI":{
            "type":"string"
        },
        "Transforms":{
            "type":"object",
            "properties":{
                "Transform":{
                    "type":"object",
                    "properties":{
                        "Algorithm":{
                            "type":"string"
                        }
                    },
                    "required":[
                        "Algorithm"
                    ]
                },
                "Algorithm":{
```

```
                    "type":"string"
                }
            },
            "required":[
                "Algorithm"
            ]
        }
    },
    "required":[
        "URI"
    ]
}
```

Código Fonte 3.4: Elemento *CipherReference*

3.2.4 Elemento *EncryptedData*

O *EncryptedData* é o elemento principal da sintaxe, que deriva do elemento *EncryptedType*. Este elemento substitui o elemento cifrado.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"EncryptedData",
    "type":"object",
    "properties":{
        "EncryptionMethod":{
            "$ref":"#/definitions/EncryptionMethod"
        },
        "KeyInfo":{
            "$ref":"#/definitions/KeyInfo"
        },
        "CipherData":{
            "$ref":"#/definitions/CipherData"
        }
    },
    "required":[
        "EncryptionMethod",
        "KeyInfo",
        "CipherData"
    ],
    "additionalProperties":false
}
```

Código Fonte 3.5: Elemento *EncryptedData*

3.2.5 Elemento *KeyInfo*

Existem duas maneiras de fornecer a informação relativa a chaves criptográficas necessárias para decifrar o *CipherData*:

- Os elementos *EncryptedData* ou *EncryptedKey* especificam o material relativo à chave de criptografia através de um elemento filho de *KeyInfo*.
- O material relativo à chave criptográfico pode ser determinada, implicitamente pelo destinatário, desta forma não precisa ser explicitamente mencionado no JSON transmitido.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"KeyInfo",
    "type":"object",
    "oneOf":[
        {
            "type":"object",
            "properties":{
                "EncryptedKey":{
                    "$ref":"#/definitions/EncryptedKey"
                }
            },
            "required":[
                "EncryptedKey"
            ]
        },
        {
            "type":"object",
            "properties":{
                "RetrievalMethod":{
                    "$ref":"#/definitions/RetrievalMethod"
                }
            },
```

```
            "required":[
                "RetrievalMethod"
            ]
        }
    ]
}
```

Código Fonte 3.6: Elemento *KeyInfo*

3.2.5.1 Elemento *EncryptedKey*

O elemento *EncryptedKey* é utilizado para transportar a chave de encriptação desde o emissor até aos receptores conhecidos. A chave é sempre cifrada para os destinatários. Quando este elemento é decifrado, os octetos resultantes são colocados à disposição do algoritmo *EncryptionMethod* sem processamento adicional.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"EncryptedKey",
    "type":"object",
    "properties ":{
        "EncryptionMethod":{
            "$ref":"#/definitions/EncryptionMethod"
        },
        "KeyInfo":{
            "$ref":"#/definitions/KeyInfo"
        },
        "CipherData":{
            "$ref":"#/definitions/CipherData"
        }
    },
    "required":[
        "EncryptionMethod",
        "KeyInfo",
        "CipherData"
    ]
}
```

Código Fonte 3.7: Elemento *EncryptedKey*

3.2.5.2 Elemento *RetrievalMethod*

O elemento *RetrievalMethod* identifica um elemento *EncryptedKey* que contém a chave necessária para decifrar o elemento *CipherData* associado.

```
JSON Schema Definition:

{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"RetrievalMethod",
    "type":"string"
}
```

Código Fonte 3.8: Elemento *RetrievalMethod*

3.2.6 Regras de Processamento

Esta secção descreve as operações de processamento criptográfico, os momentos em que a informação é cifrada/decifrada e os elementos utilizados. Os perfis seguintes foram definidos como requisitos de conformidade:

- **Aplicação**

  Aplicação que efetua o pedido a uma implementação SecJSON através de dados e parâmetros necessários para o seu processamento.

- ***Encryptor***

  Implementação SecJSON com perfil de cifra de informação.

- ***Decryptor***

  Implementação SecJSON com perfil de decifra de informação.

3.2.6.1 Cifra

Para cifrar um conjunto de informação como um elemento *EncryptedData* ou *EncryptedKey* (elementos derivados de *EncryptedType*) o ***Encryptor*** deve:

1. Seleccionar o algoritmo e parâmetros necessários para a cifra dos dados.

2. Obter e (opcionalmente) representar a chave.

   (a) A chave em si poderá ser cifrada em conjunto com a informação. Para isso deverá ser construido um elemento *EncryptedKey* através da aplicação deste processo de forma recursiva. O elemento *EncryptedKey* resultante será um filho do elemento *KeyInfo*.

3. Cifrar os dados:

   (a) Obter os octetos através da serialização dos dados em UTF-8 (ou outro *enconding* escolhido pela **Aplicação**). A serialização pode ser feita pelo ***Encryptor***, caso contrário a mesma deverá ser feita pela **Aplicação**.

   (b) Cifrar octetos através da utilização do algoritmo e chave obtidos nos passos 1 e 2.

   (c) Com excepção do ***Decryptor*** ter conhecimento do tipo de dados sujeitos ao processo de cifra, o ***Encryptor*** deverá fornecer esta informação.

4. Construir a estrutra *EncryptedType* (*EncryptedData* ou *EncryptedKey*):

   A estrutura *EncryptedType* representa todas as informações préviamente discutidas, incluindo o algoritmo de criptografia, parâmetros, chave, dados cifrados, etc.

   (a) Caso seja necessário armazenar a sequência de octetos obtida no passo 3 (*CipherData*), a sequência é codificada em formato *Base64* e inserida como conteúdo de um elemento *CipherValue*.

   (b) Caso seja necessário armazenar externamente a sequência de octetos, a representação do URI e transformações (caso existam) correspondente à localização da sequência cifrada será armazenada dentro do elemento *CipherReference*.

5. Processar *EncryptedData*:

(a) Caso o tipo de dados cifrados seja um elemento JSON o ***Encryptor*** deve permitir o retorno do elemento *EncryptedData* para a **Aplicação**. Esta poderá usar o elemento como um novo documento JSON ou inserir o mesmo noutro documento JSON.

O ***Encryptor*** deve permitir a substituição do elemento não cifrado pelo elemento *EncryptedData*. Quando a **Aplicação** requer um elemento JSON para substituição, fornece o contexto do documento JSON e identifica o elemento a ser substituído. O ***Encryptor*** remove o elemento e insere o elemento *EncryptedData* respetivo.

(b) Caso o tipo de dados cifrados não seja um elemento JSON, o ***Encryptor*** deve retornar sempre o elemento *EncryptedData* para a **Aplicação**. Esta poderá usar o elemento como um novo documento JSON ou inserir o mesmo noutro documento JSON.

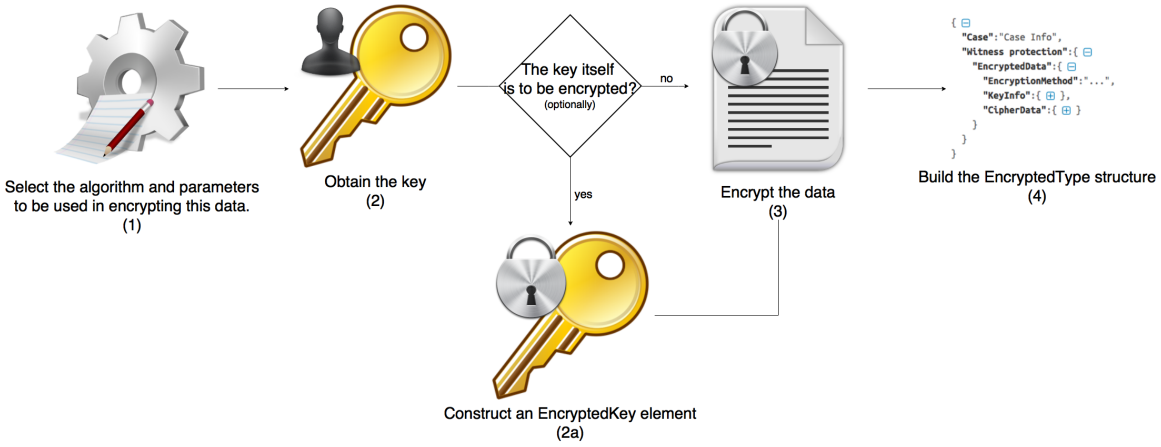

FIGURA 3.3: Processo de cifra SecJSON

3.2.6.2 Decifra

Para cada elemento, derivado de *EncryptedType*, a decifrar, o ***Decryptor*** deve:

1. Processar o elemento e determinar o algoritmo, parâmetro e elemento *KeyInfo* a utilizar. Caso alguma informação esteja omitida, a **Aplicação** é responsáveis pelo seu fornecimento.

2. Identificar a chave criptográfica de acordo com o elemento *KeyInfo*. No caso de a chave se encontrar cifrada, deverá ser identificada a chave correspondente, posteriormente deverá ser realizado o processo de decifrar de modo recursivo. Caso contrário, a chave criptográfica poderá ser obtida através de uma *local store* utilizando os parâmetros fornecidos ou vínculo implícito.

3. Decifrar os dados contidos no elemento *CipherData*:

   (a) Caso o elemento *CipherValue* esteja presente, o contúdo será sujeito a uma descodificação *Base64* obtendo a sequência de octetos cifrada.

   (b) Caso o elemento *CipherReference* esteja presente, o URI e transformações (caso existam) são usadas para obter a sequência de octetos cifrada.

   (c) A sequência de octetos cifrada é decifrada utilizando o algoritmo/parâmetros e chave determinados nos passos 1 e 2.

4. Processar dados decifrados

   (a) O sequência de octetos obtida no passo 3 é interpretado com o *encoding* UTF-8.

   (b) O ***Decryptor*** deve permitir o retorno dos dados numa estrutura JSON com o *encoding* definido. O ***Decryptor*** não é obrigado a validar o JSON serializado.

   (c) O ***Decryptor*** deve suportar a capacidade de substituir o elemento *EncryptedData* pelo elemento JSON decifrado ou conteúdo simples. O ***Decryptor*** não é obrigado a validar o resultado desta operação de substituição.

   A **Aplicação** fornece o contexto do documento JSON e identifica o elemento *EncryptedData* a substítuir. Se o documento alvo não possuir o *encoding* UTF-8, o ***Decryptor*** deve efetuar a codificação para o *encoding* correto.

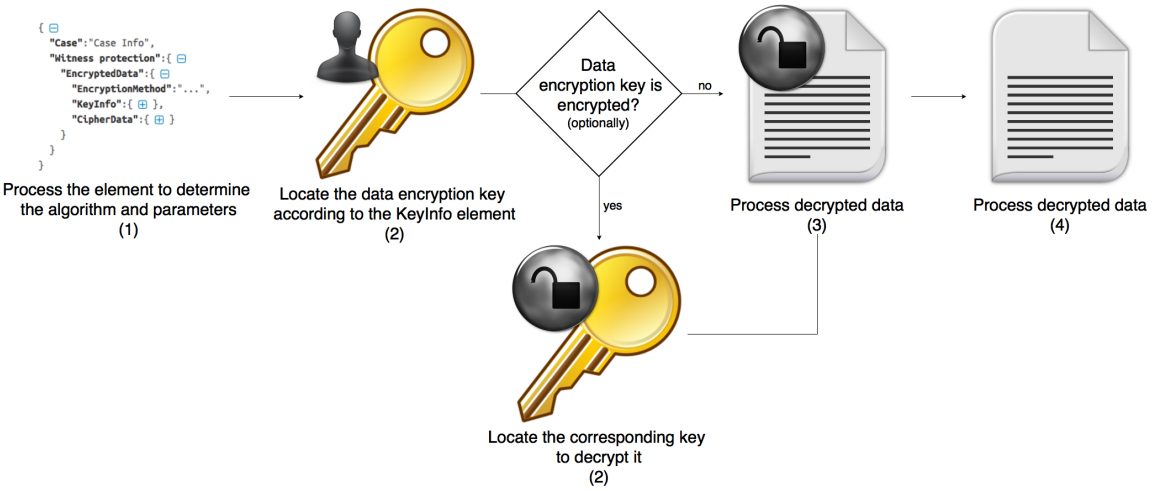

FIGURA 3.4: Processo de decifra SecJSON

3.2.7 Algoritmos

Esta secção discute os algoritmos utilizados com a especificação SecJSON. O conteúdo do *Algorithm* do elemento *EncryptionMethod* é o identificador do algoritmo (secção 3.2.7) ou outro elemento representativo do papel do algoritmo, uma referência formal para uma especificação, definição da chave criptográfica e resultados das operações de criptografia aplicadas.

O conjunto de algoritmos listados abaixo têm parâmetros implícitos que dependem do seu papel (dados para cifra/decifra, materiais relativos a chave ou a operação criptográfica a executar). Quaisquer parâmetros adicionais explícitos surgem como elementos filho do elemento *EncryptionMethod*.

Lista de Algoritmos por Categoria

Foram definidos um conjunto de algoritmos obrigatórios (e recomendados) para as categorias *Block Encryption*, *Key Transport* e *Message Digest*. Desta forma, qualquer implementação do mecanismo SecJSON permitirá, no minimo, a utilização dos algoritmos definidos como obrigatórios.

Em cada categoria existe um nome breve, nível de exigência da implementação e um URI de identificação de cada algoritmo.

Block Encryption

1. OBRIGATÓRIO - TRIPLEDES
   http://tiagomistral.github.io/SecJSON#tripledes-cbc
2. OBRIGATÓRIO - AES-128
   http://tiagomistral.github.io/SecJSON#aes128-cbc
3. RECOMENDADO - AES-192
   http://tiagomistral.github.io/SecJSON#aes192-cbc
4. OBRIGATÓRIO - AES-256
   http://tiagomistral.github.io/SecJSON#aes256-cbc

Key Transport

1. OBRIGATÓRIO - RSA-v1.5
   http://tiagomistral.github.io/SecJSON#rsa-1_5
2. OBRIGATÓRIO - RSA-OAEP
   http://tiagomistral.github.io/SecJSON#rsa-oaep-mgf1p

Message Digest

1. OBRIGATÓRIO - SHA1
   http://tiagomistral.github.io/SecJSON#sha1

Conclusões

Após a definição desta sintaxe é possível implementar, utilizando todas as regras e restrições anteriores. um sistema que permite cifrar e decifrar qualquer tipo de dados num objeto SecJSON. Caso os dados a cifrar já se encontrem no formato JSON, será ainda possível aplicar várias cifras diferentes sobre objetos diferentes dentro do mesmo documento JSON.

No próximo capitulo será desenvolvido um protótipo deste sistema numa das linguagens analisadas na seção 2.5.

Capítulo 4

Implementação do Sistema

Ao longo deste capítulo será definido e apresentado um ambiente para a implementação da especificação SecJSON definida e proposta anteriormente. Este capítulo está dividido em três secções que apresentam e justificam a fase de implementação e utilização do SecJSON. Desta forma, para alcançar os objetivos definidos a implementação de uma solução do mecanismo SecJSON foi divido nas três fases que se apresentam de seguida.

1. Definição de um ambiente de implementação onde será escolhida uma linguagem de programação que satisfaça todos os requisitos definidos anteriormente.
2. Implementação e disponibilização da solução na linguagem definida.
3. Exemplos de utilização da solução desenvolvida.

No final deste capítulo será possível para qualquer pessoa incorporar o mecanismo de segurança SecJSON nas suas próprias aplicações e serviços.

4.1 Ambiente de Desenvolvimento

Para a implementação e validação de uma solução deste mecanismo de segurança foi usada a *framework Node.js*. Além de todas as vantagens enumeradas anteriormente na secção 2.5.3 e do gestor de pacotes NPM, o Node.js é baseado em *JavaScript* tal como o formato JSON. Esta base comum apresenta-se como um trunfo importante ao nível da performance e aprendizagem inerente ao desenvolvimento deste trabalho.

4.2 Arquitectura

A arquitectura do sistema será divida em três módulos distintos (Figura 4.1):

- Módulo de obtenção dos dados a cifrar/decifrar

  Módulo é responsável por receber da aplicação/serviço cliente a informação necessária para as operações criptográficas e invocação do módulo cifra/decifra.

- Módulo de cifra

  Módulo que agrega todas as funções criptográficas relativas à cifra da informação e construção da estrutura SecJSON resultante.

- Módulo de decifra

  Contrariamente ao módulo de cifra, o módulo de decifra agrega todas as funções criptográficas relativas à decifra da estrutura SecJSON e construção do objeto JSON resultante.

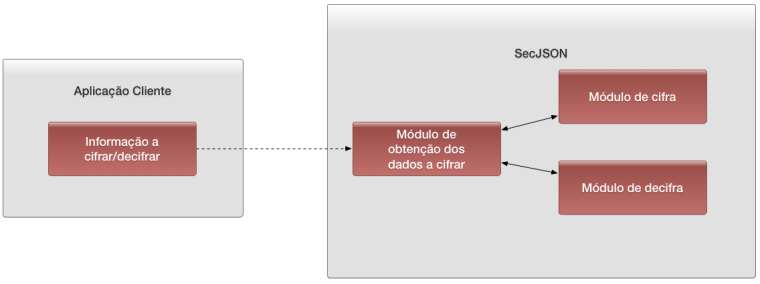

FIGURA 4.1: Arquitectura SecJSON

4.3 *secjson.js*

Durante esta secção serão apresentadas as principais funções Node.js desenvolvidas de acordo com a sintaxe definida nas secções anteriores.

4.3.1 Processo de cifra

O processo de cifra (função *encrypt* - código fonte 4.1) é responsável por receber um conteúdo a cifrar e devolver um objeto JSON de acordo com a sintaxe definida. Como parâmetros obrigatórios esta função exige:

- Conteúdo a cifrar;
- Uma chave pública RSA;
- Um certificado x509 em formato PEM.

Posteriormente serão invocados, de forma sequencial, os métodos necessários à cifra do conteúdo fornecido:

1. *generate_symmetric_key*

   Gerar uma chave simétrica para cifrar o conteúdo definido pelo utilizador.

2. *encrypt_content*

Cifrar o conteúdo definido pelo utilizador com a chave gerada no ponto anterior.

3. *encrypt_key*

   Cifrar a chave simétrica utilizada na criptografia com a chave publica fornecida pelo utilizador.

   (a) *utils.renderTemplate*

   Gerar o objeto JSON com a informação criptográfica nos elementos SecJSON correspondentes.

```
function encrypt(content, options, callback) {
  if (!options)
    return callback(new Error('must provide options'));
  if (!content)
    return callback(new Error('must provide content to encrypt'));
  if (!options.rsa_pub)
    return callback(new Error('rsa_pub option is mandatory and you should provide a
                                    valid RSA public key'));
  if (!options.pem)
    return callback(new Error('pem option is mandatory and you should provide a
                                    valid x509 certificate encoded as PEM'));

  options.input_encoding = options.input_encoding || 'utf8';

  async.waterfall([
    function generate_symmetric_key(cb) {
      switch (options.encryptionAlgorithm) {
        case 'http://tiagomistral.github.io/SecJSON#aes128-cbc':
          crypto.randomBytes(16, cb); // generate a symmetric random key 16 bytes
                                      // length
          break;
        case 'http://tiagomistral.github.io/SecJSON#aes256-cbc':
          crypto.randomBytes(32, cb); // generate a symmetric random key 32 bytes
                                      // length
          break;
        case 'http://tiagomistral.github.io/SecJSON#tripledes-cbc':
          crypto.randomBytes(24, cb); // generate a symmetric random key 24 bytes
                                      // (192 bits) length
          break;
        default:
          crypto.randomBytes(32, cb); // generate a symmetric random key 32 bytes
                                      // length
      }
```

```
    },
    function encrypt_content(symmetricKey, cb) {
      switch (options.encryptionAlgorithm) {
        case 'http://tiagomistral.github.io/SecJSON#aes128-cbc':
          encryptWithAlgorithm('aes-128-cbc', symmetricKey, 16, content,
              options.input_encoding, function (err, encryptedContent) {
            if (err) return cb(err);
            cb(null, symmetricKey, encryptedContent);
          });
          break;
        case 'http://tiagomistral.github.io/SecJSON#aes256-cbc':
          encryptWithAlgorithm('aes-256-cbc', symmetricKey, 16, content,
              options.input_encoding, function (err, encryptedContent) {
            if (err) return cb(err);
            cb(null, symmetricKey, encryptedContent);
          });
          break;
        case 'http://tiagomistral.github.io/SecJSON#tripledes-cbc':
          encryptWithAlgorithm('des-ede3-cbc', symmetricKey, 8, content,
              options.input_encoding, function (err, encryptedContent) {
            if (err) return cb(err);
            cb(null, symmetricKey, encryptedContent);
          });
          break;
        default:
          cb(new Error('encryption algorithm not supported'));
      }
    },
    function encrypt_key(symmetricKey, encryptedContent, cb) {
      encryptKeyInfo(symmetricKey, options, function(err, keyInfo) {
        if (err) return cb(err);

        var result = utils.renderTemplate('encrypted-key', {
          encryptedContent: encryptedContent.toString('base64'),
          keyInfo: keyInfo,
          contentEncryptionMethod: options.encryptionAlgorithm
        });

        cb(null, result);
      });
    }
  ], callback);
}
```

Código Fonte 4.1: Função encrypt

4.3.2 Processo de decifra

O processo de decifra (função *decrypt* - código fonte 4.2) é responsável por receber um objeto JSON de acordo com a sintaxe SecJSON e uma chave privada e devolver o conteúdo decifrado. Como parâmetros obrigatórios esta função exige:

- Objeto JSON de acordo com a sintaxe SecJSON;
- Uma chave privada RSA.

Posteriormente serão invocados, de forma sequencial, os métodos necessários à cifra do conteúdo fornecido:

1. *JSON.parse*

   Validar objeto JSON fornecido.

2. *decryptKeyInfo*

   Decifrar o conteúdo do elemento *EncryptedData.KeyInfo.CipherData* com a chave privada fornecida, obtendo a chave simétrica utilizada no processo de cifra do *payload*.

3. *switch (encryptionAlgorithm)*

   Decifrar o *payload* com a chave simétrica obtida no ponto anterior. Este processo está dependente do elemento *EncryptedData.EncryptionMethod*, cuja informação corresponde ao algoritmo criptográfico utilizado (AES 128, AES 256 ou TripleDES).

```
function decrypt(json, options, callback) {
  if (!options)
    return callback(new Error('must provide options'));
  if (!json)
    return callback(new Error('must provide JSON to decrypt'));
  if (!options.key)
    return callback(new Error('key option is mandatory and you should provide a
                                                          valid RSA private key'));

  var decrypted;
```

```
try {
  var doc = JSON.parse(json);

  var symmetricKey = decryptKeyInfo(doc, options);
  var encryptionAlgorithm = doc.EncryptedData.EncryptionMethod.Algorithm;

  var decipher;
  var padding;
  var encryptedContent = doc.EncryptedData.CipherData.CipherValue;

  var encrypted = new Buffer(encryptedContent, 'base64');

  switch (encryptionAlgorithm) {
    case 'http://tiagomistral.github.io/SecJSON#aes128-cbc':
      decipher = crypto.createDecipheriv('aes-128-cbc', symmetricKey,
                                         encrypted.slice(0, 16));

      decipher.setAutoPadding(false);
      decrypted = decipher.update(encrypted.slice(16), null, 'binary')
                                                  + decipher.final('binary');

      // Remove padding bytes equal to the value of the last byte of the returned data.
      padding = decrypted.charCodeAt(decrypted.length - 1);
      if (1 <= padding && padding <= 16) {
        decrypted = decrypted.substr(0, decrypted.length - padding);
      } else {
        callback(new Error('padding length invalid'));
        return;
      }

      decrypted = new Buffer(decrypted, 'binary').toString('utf8');
      break;
    case 'http://tiagomistral.github.io/SecJSON#aes256-cbc':
      decipher = crypto.createDecipheriv('aes-256-cbc', symmetricKey,
                                           encrypted.slice(0, 16));

      decipher.setAutoPadding(false);
      decrypted = decipher.update(encrypted.slice(16), null, 'binary')
                                                  + decipher.final('binary');

      // Remove padding bytes equal to the value of the last byte of the returned data.
      padding = decrypted.charCodeAt(decrypted.length - 1);
      if (1 <= padding && padding <= 16) {
        decrypted = decrypted.substr(0, decrypted.length - padding);
      } else {
        callback(new Error('padding length invalid'));
        return;
```

```
        }
        decrypted = new Buffer(decrypted, 'binary').toString('utf8');
        break;
      case 'http://tiagomistral.github.io/SecJSON#tripledes-cbc':
        decipher = crypto.createDecipheriv('des-ede3-cbc', symmetricKey,
                                                          encrypted.slice(0,8));
        decrypted = decipher.update(encrypted.slice(8), null, 'binary')
                                                    + decipher.final('binary');
        decrypted = new Buffer(decrypted, 'binary').toString('utf8');
        break;
      default:
        return callback(new Error('encryption algorithm ' + encryptionAlgorithm
                                                        + ' not supported'));
    }
  } catch (e) {
    return callback(e);
  }

  callback(null, decrypted);
}
```

Código Fonte 4.2: Função decrypt

4.3.3 Criptografia JSON

As funções anteriores foram desenvolvidas para o processamento de informação simples, ou seja, criptografia de uma cadeia de caracteres que pode representar, ou não, um objeto JSON. No entanto, um dos objetivos deste trabalho está na aplicação de múltiplas cifras a um objeto JSON.

Desta forma foi necessário realizar algum trabalho anterior ao processo de cifra e posterior ao processo de decifra, a este trabalho adicional designamos como **cifra JSON** e **decifra JSON**, respetivamente.

4.3.4 Processo de cifra JSON

O processo de *cifra JSON* é responsável por identificar a chave JSON (*JSON keypath*) fornecida pelo utilizador, realizar o processo de cifra sobre o seu *value* e substituir o mesmo pelo elemento SecJSON correspondente.

Este processo é iniciado na função *jsonEncrypt* (código fonte 4.3) onde a chave fornecida pelo utilizador é decomposta, posteriormente é invocada a função recursiva *findKeyEncryptValue* (código fonte 4.4) que irá iterar o objeto JSON até obter o valor correspondente à chave JSON.

Quando obtemos a sequência de caracteres a cifrar será efetuada uma chamada síncrona ao processo de cifra (descrito em 4.3.1). Finalmente o elemento a cifrar é substituído pelo elemento SecJSON resultante do processo de cifra e o objeto JSON é devolvido.

```
function jsonEncrypt(json, jsonKey, options, callback) {
  if (!json)
    return callback(new Error('must provide json object'));
  if (!jsonKey)
    return callback(new Error('must provide jsonKey path to encrypt'));

  var key = jsonKey.split(/[\[\].,]/).filter(String);

  callback(null, findKeyEncryptValue(json,key,options));
}
```

Código Fonte 4.3: Função de cifra JSON

```
function findKeyEncryptValue(json, keyPath, options){
  if(keyPath.length == 1) {
    json[keyPath[0]] = jsonSyncEncrypt(JSON.stringify(json[keyPath[0]]), options);
    return json;
  }

  json[keyPath[0]] = findKeyEncryptValue(json[keyPath[0]],
                                         keyPath.slice(1, keyPath.length), options);
  return json;
}
```

Código Fonte 4.4: Função de pesquisa e cifra JSON

4.3.5 Processo de decifra JSON

O processo de *decifra JSON* é responsável por identificar o elemento SecJSON através da chave JSON (*JSON keypath*) fornecida pelo utilizador, realizar o processo de decifra e devolver o objeto JSON resultante.

De forma análoga ao processo de *cifra JSON*, este processo é iniciado na função *jsonDecrypt* (código fonte 4.5) onde a chave fornecida pelo utilizador é decomposta e invocada a função recursiva *findKeyDecryptValue* (código fonte 4.6) que irá iterar o objeto JSON até obter o elemento SecJSON correspondente à chave fornecida pelo utilizador.

Quando estamos na presença do elemento SecJSON será efetuada uma chamada síncrona ao processo de cifra (descrito em 4.3.2). Para concluir o elemento a decifrar é substituído pela sequencia de caracteres inicialmente cifrado e o objeto JSON resultante é devolvido ao utilizador.

```
function jsonDecrypt(json, jsonKey, options, callback) {
  if (!json)
    return callback(new Error('must provide json object'));
  if (!jsonKey)
    return callback(new Error('must provide jsonKey to encrypt'));

  var key = jsonKey.split(/[\[\].,]/).filter(String);

  callback(null, findKeyDecryptValue(json,key,options));
}
```

Código Fonte 4.5: Função de decifra JSON

```
function findKeyDecryptValue(json, keyPath, options){
  if(keyPath.length == 1) {
    json[keyPath[0]] = jsonSyncDecrypt(JSON.stringify(json[keyPath[0]]), options);
    return json;
  }

  json[keyPath[0]] = findKeyDecryptValue(json[keyPath[0]],
                                         keyPath.slice(1, keyPath.length), options);
  return json;
}
```

Código Fonte 4.6: Função de pesquisa e decifra JSON

4.4 Utilização do sistema

Nesta secção será descrito como utilizar o protótipo desenvolvido, expondo alguns exemplos e respectivos resultados.

4.4.1 Criptografia Simples

O processo de cifra de uma simples frase (neste caso *"content to encrypt"*) está descrito no exemplo 4.7 e o respectivo resultado em 4.8. Posteriormente poderá ser aplicado o processo de decifra, descrito no exemplo 4.9, sobre o cifra obtendo a frase original.

O objeto *options* é responsável por agregar toda a informação necessária aos processos de cifra/decifra e é composto pelos seguintes elementos:

- ***encryptionAlgorithm***

  Algoritmo criptográfico utilizado para cifrar a informação recebida pelo utilizador.

- ***keyEncryptionAlgorithm***

  Algoritmo criptográfico que será utilizado para cifrar uma chave gerada para cifrar a informação recebida pelo utilizador. Desta forma é possível obter uma melhoria de performance utilizando algoritmos assimétricos.

- ***rsa_pub***

  Caso seja escolhido um algoritmo assimétrico será necessário passar a chave publica a utilizar no processo criptográfico.

- ***pem***

  Certificado da chave publica passada no ponto anterior.

- ***key***

  Chave privada que será utilizada no processo criptográfico invocado.

Cifra

```
var secjson = require('secjson');

var options = {
    rsa_pub: fs.readFileSync(__dirname + '/test-auth0_rsa.pub'),
```

```
    pem: fs.readFileSync(__dirname + '/test-auth0.pem'),
    encryptionAlgorithm: 'http://tiagomistral.github.io/SecJSON#aes128-cbc',
    keyEncryptionAlgorighm: 'http://tiagomistral.github.io/SecJSON#rsa-oaep-mgf1p'
};

secjson.encrypt('content to encrypt', options, function(err, result) {
    console.log(result);
});
```

Código Fonte 4.7: Cifra SecJSON (Simples)

Resultado da Cifra

```
{
  "EncryptedData":{
    "EncryptionMethod":{
      "Algorithm":"http://tiagomistral.github.io/SecJSON#aes128-cbc"
    },
    "KeyInfo":{
      "EncryptedKey":{
        "EncryptionMethod":{
          "Algorithm":"http://tiagomistral.github.io/SecJSON#rsa-oaep-mgf1p",
          "DigestMethod":{
            "Algorithm":"http://tiagomistral.github.io/SecJSON#sha1"
          }
        },
        "KeyInfo":{
          "RetrievalMethod":"MIIEDzCCAveg...[base64 cert]...q3uaLvlAUo="
        },
        "CipherData":{
          "CipherValue":"Ad/F7DvLVc...[encrypted symmetric key]...0bb2VrjXcTZxQ=="
        }
      }
    },
    "CipherData":{
      "CipherValue":"g5HaGPWeQZ...[encrypted content]...4qYHVr5/pUQgjwVd"
    }
  }
}
```

Código Fonte 4.8: Resultado Cifra (Simples)

Decifra

```
var secjson = require('secjson');

var decryptOptions = {
  key: fs.readFileSync(__dirname + '/test-auth0.key')
};


secjson.decrypt(encryptResult, decryptOptions, function(err, dec) {
  console.log(dec);
});
```

Código Fonte 4.9: Decifra SecJSON (Simples)

4.4.2 Criptografia JSON

A cifra de um elemento JSON específico obriga à sua identificação durante os processos de cifra e decifra.

Assim este processo recebe um argumento adicional, que define a localização da informação a cifrar/decifrar através de uma *JSON keypath*. No exemplo 4.10 temos o argumento *store.book[1]* que localiza o livro com o título *Sword of Honour*.

Cifra

```
var options = {
      rsa_pub: fs.readFileSync(__dirname + '/test-auth0_rsa.pub'),
      pem: fs.readFileSync(__dirname + '/test-auth0.pem'),
      encryptionAlgorithm: 'http://tiagomistral.github.io/SecJSON#aes128-cbc',
      keyEncryptionAlgorighm: 'http://tiagomistral.github.io/SecJSON#rsa-oaep-mgf1p'
    };

    var obj = {
    "store": {
      "book": [
        {
          "category": "reference",
          "author": "Nigel Rees",
          "title": "Sayings of the Century",
          "price": 8.95
        },
        {
```

```
          "category": "fiction",
          "author": "Evelyn Waugh",
          "title": "Sword of Honour",
          "price": 12.99
        }
      ],
      "bicycle": {
        "color": "red",
        "price": 19.95
      }
        }
      };

secjson.jsonEncrypt(obj, 'store.book[1]', options, function(err, encryptJsonObject) {
    console.log(JSON.stringify(encryptJsonObject));
});
```

Código Fonte 4.10: Cifra elemento JSON

Resultado da Cifra

```
{
  "store":{
    "book":[
      {
        "category":"reference",
        "author":"Nigel Rees",
        "title":"Sayings of the Century",
        "price":8.95
      },
      {
        "EncryptedData":{
          "EncryptionMethod":{
            "Algorithm":"http://tiagomistral.github.io/SecJSON#aes128-cbc"
          },
          "KeyInfo":{
            "EncryptedKey":{
              "EncryptionMethod":{
                "Algorithm":"http://tiagomistral.github.io/SecJSON#rsa-oaep-mgf1p",
                "DigestMethod":{
                  "Algorithm":"http://tiagomistral.github.io/SecJSON#sha1"
                }
              },
              "KeyInfo":{
                "RetrievalMethod":"MIIEDzCCAveg...[base64 cert]...q3uaLvlAUo="
              },
```

```
            "CipherData":{
              "CipherValue":"nBZdvu0Go+Qp...[encrypted symmetric key]...hoCJR5NVxOdgdJcGClg=="
            }
          }
        },
        "CipherData":{
          "CipherValue":"1GV6RbZcWRAe...[encrypted content]...IjcyW6sGKAC8IqMsBBesw=="
        }
      }
    }
  ],
  "bicycle":{
    "color":"red",
    "price":19.95
  }
 }
}
```

Código Fonte 4.11: Resultado Cifra JSON

Decifra

```
var secjson = require('secjson');

var decryptOptions = {
    key: fs.readFileSync(__dirname + '/test-auth0.key')
};

secjson.jsonDecrypt(encryptJsonObject, 'store.book[1]', decryptOptions, function(err, dec) {
    console.log(JSON.stringify(dec));
});
```

Código Fonte 4.12: Decifra elemento JSON

Conclusões

A implementação do mecanismo SecJSON consistiu no desenvolvimento de uma biblioteca de software desenvolvida em *Node.js*.

Através desta implementação será possível qualquer programador intergrar as funcionalidades de segurança SecJSON nos seus sistemas/serviços, desde que estes suportem módulos *Node.js*.

A disponibilização desta biblioteca foi realizada através do gestor de bibliotecas NPM, onde está disponível através do sitio https://www.npmjs.com/package/secjson.

Capítulo 5

Validação do Sistema

No decorrer deste capítulo serão efetuados testes que visam validar o protótipo desenvolvido relativamente à solução mais utilizadas nesta área, concretamente o protocolo SSL/TLS.

Ao longo deste capitulo vamos falar sobre dois termos que poderão gerar confusão, nomeadamente transmissão e serialização. No contexto desta dissertação definimos **transmissão** como o conjunto de processos de cifra, decifra e serialização. **Serialização** representa apenas os processos de necessários para a troca de informação entre dois sistemas.

Durante os testes de transmissão serão utilizados vários ficheiros JSON, detalhados em seguida:

- Ficheiro 1 (0,00098 Mb)

  Este ficheiro é um vetor composto por diversos pares chave/valor simples, onde o tamanho de cada par é constante.

```
[
    {
      "8000": "teste8000"
    },
    {
      "8001": "teste8001"
    },
```

```
    {
      "8002": "teste8002"
    },
    {
      "8003": "teste8003"
    }
  ]
```

Código Fonte 5.1: Ficheiro de teste 1

- Ficheiro 2 (6,58008 Mb)

  O ficheiro 2 é um vetor composto por diversos documentos JSON complexos. A primeira posição do vetor possui uma imagem codificada em formato *Base64*.

```
[
    {
      "8000": "teste8000",
      "picture": "iVBORw0KGgoAAAANSUhEUgAAGQAAABLACAYAAAAPmRUPAAAACXBIWXMAA
      AsTAAAAlEQVR4nOzdQjS [...] base64 picture [...] /cmk0QJAYSkvdoRPUj6A"
    },
    {
      "8001": "teste8001"
    },
    {
      "8002": "teste8002"
    },
    {
      "8003": "teste8003"
    }
  ]
```

Código Fonte 5.2: Ficheiro de teste 2

- Ficheiro 3 (15,5498 Mb)

  Este ficheiro é idêntico ao ficheiro 2 diferindo apenas na imagem codificada.

```
[
    {
      "8000": "teste8000",
      "picture": "GgoAAAANSUhEUgAAGQAAABLACAYAAAPRUAPAJAYSkvAAAACXBIWXAszZZ
      ALEwEAAgAERY4nOZKjS [...] base64 bigger picture [...] /c0daRUoRU67sgM"
    },
    {
      "8001": "teste8001"
```

```
  },
  {
    "8002": "teste8002"
  },
  {
    "8003": "teste8003"
  }
]
```

Código Fonte 5.3: Ficheiro de teste 2

A máquina utilizada para as validações tinha como principais características:

- Processador Intel® Pentium® CPU G2130 (@ 3,20 GHz)
- 8 GB RAM
- Sistema operativo Windows 8 de 64 bits.

5.1 Transmissão entre dois nós (Cliente - Servidor)

O objetivo deste teste será avaliar a performance da utilização do SecJSON na transmissão de informação entre dois nós. Posteriormente será comparado com a performance apresentada pelo SSL/TLS.

Procedimento SecJSON:

1. Cifra SecJSON de um documento JSON;
2. Serialização do documento cifrado através do protocolo HTTP;
3. Decifra do documento cifrado pelo nó servidor;

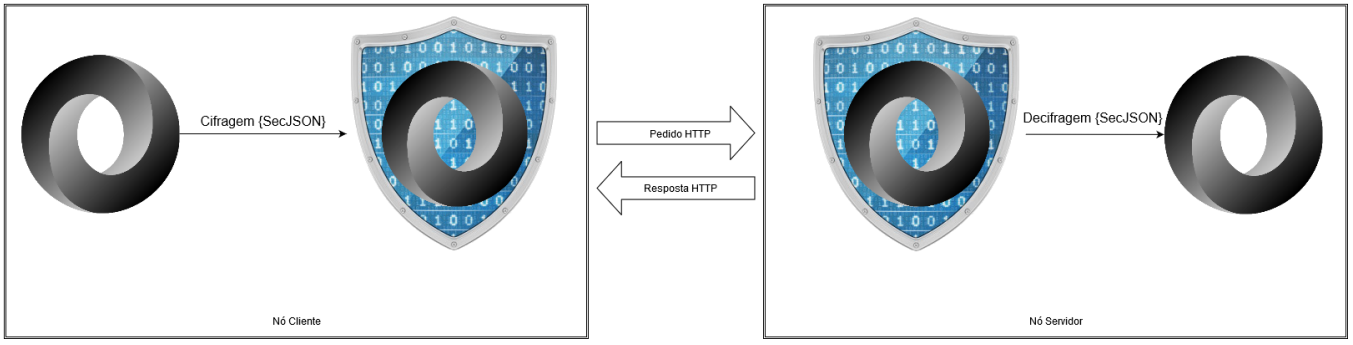

FIGURA 5.1: Transmissão SecJSON entre dois nós

Procedimento SSL:

1. Transmissão do objeto cifrado através do protocolo HTTPS;

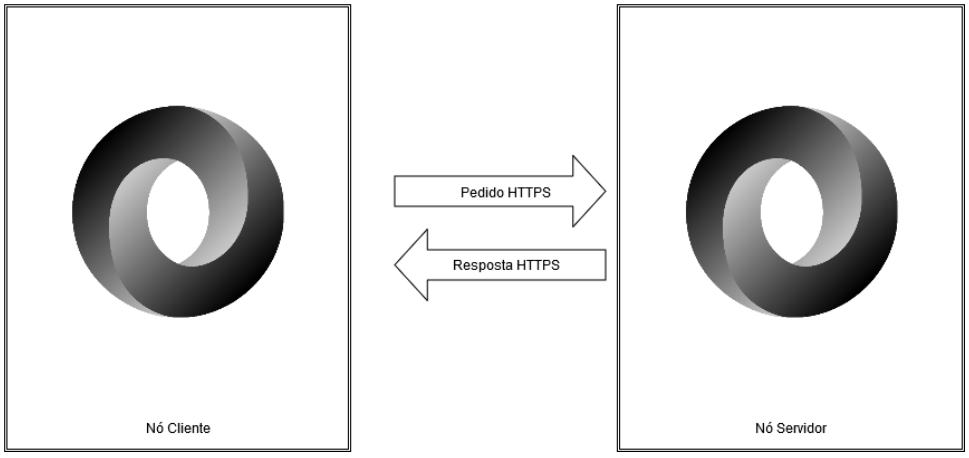

FIGURA 5.2: Transmissão SSL entre dois nós

Os dois procedimentos foram repetidos dezenas de vezes para cada um dos ficheiros referidos anteriormente. Os resultados obtidos podem ser observados nos seguintes gráficos:

Ficheiro 1

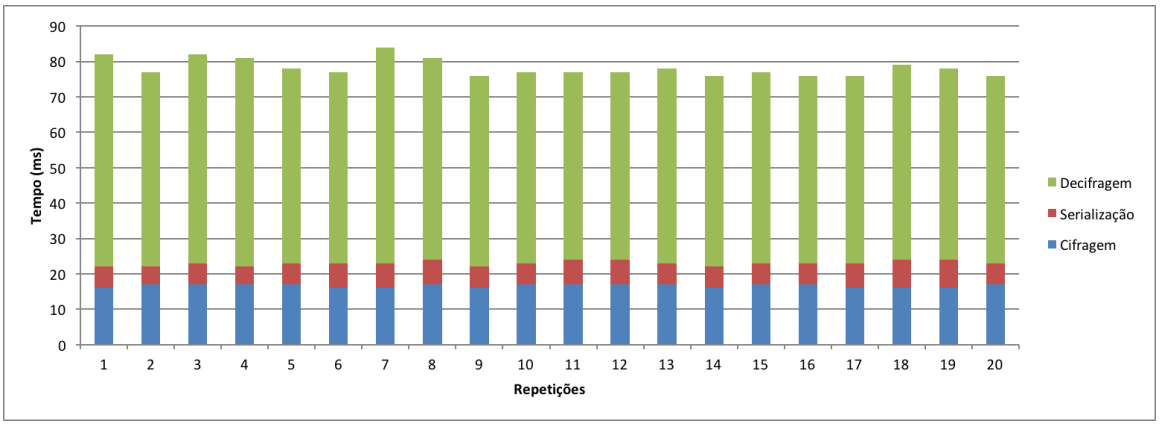

FIGURA 5.3: Transmissão SecJSON do Ficheiro 1

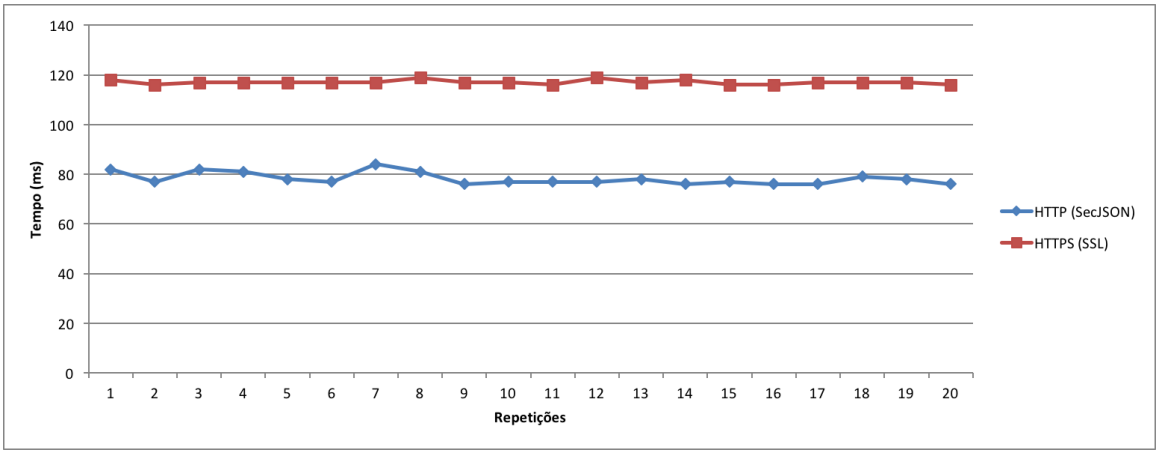

FIGURA 5.4: Transmissão SSL do Ficheiro 1

Ficheiro 2

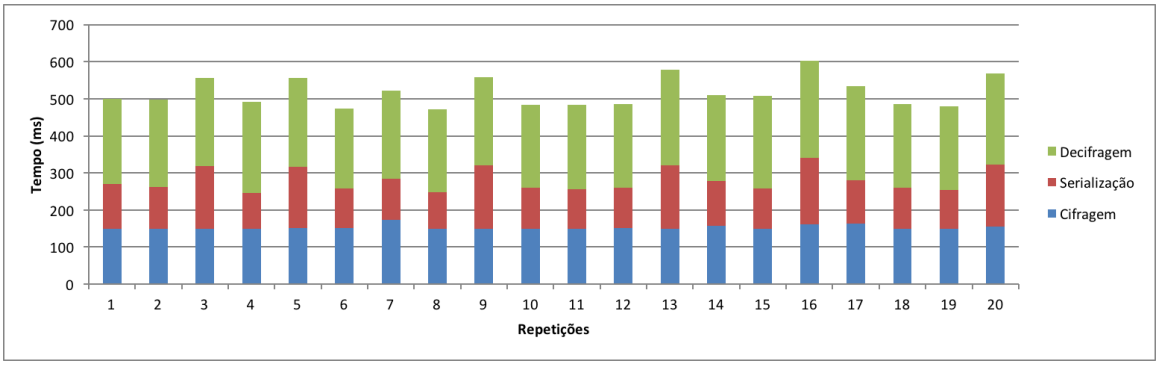

FIGURA 5.5: Transmissão SecJSON do Ficheiro 2

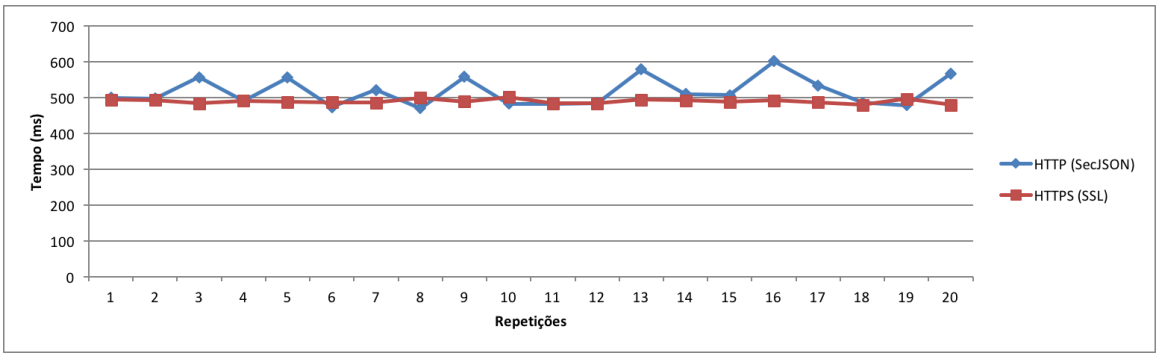

FIGURA 5.6: Transmissão SSL do Ficheiro 2

Ficheiro 3

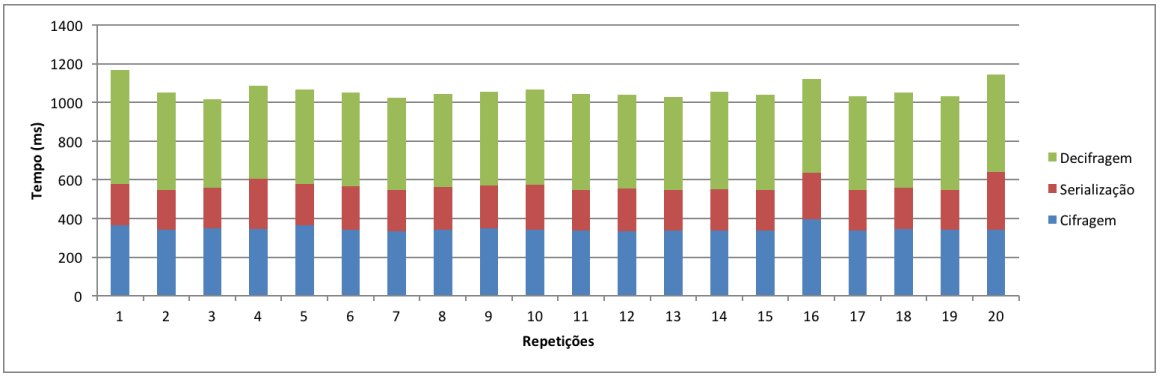

FIGURA 5.7: Transmissão SecJSON do Ficheiro 3

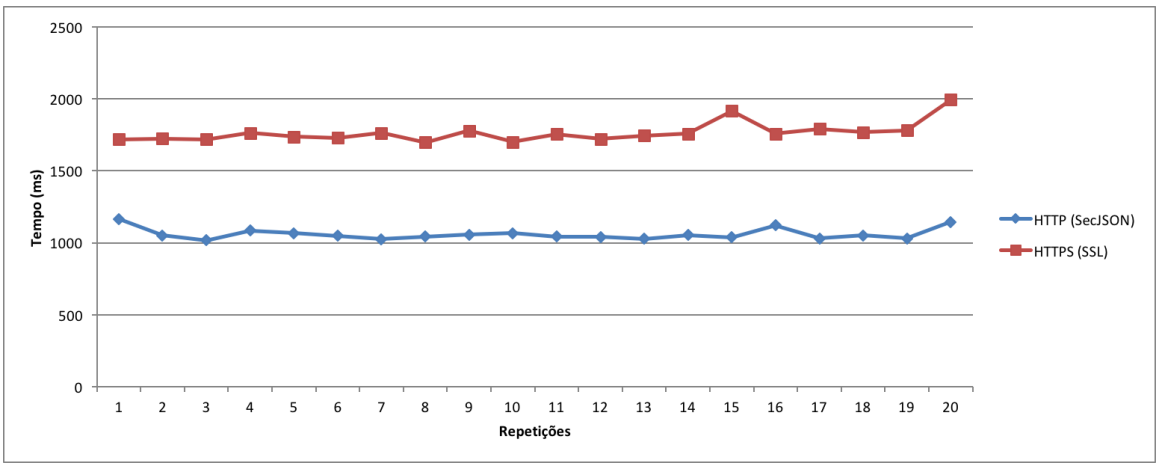

FIGURA 5.8: Transmissão SSL do Ficheiro 3

Através dos resultados anteriores poderemos calcular o tempo médio para a serialização e processos criptográficos de cada ficheiro e comparar a sua evolução em função do tamanho do *payload*.

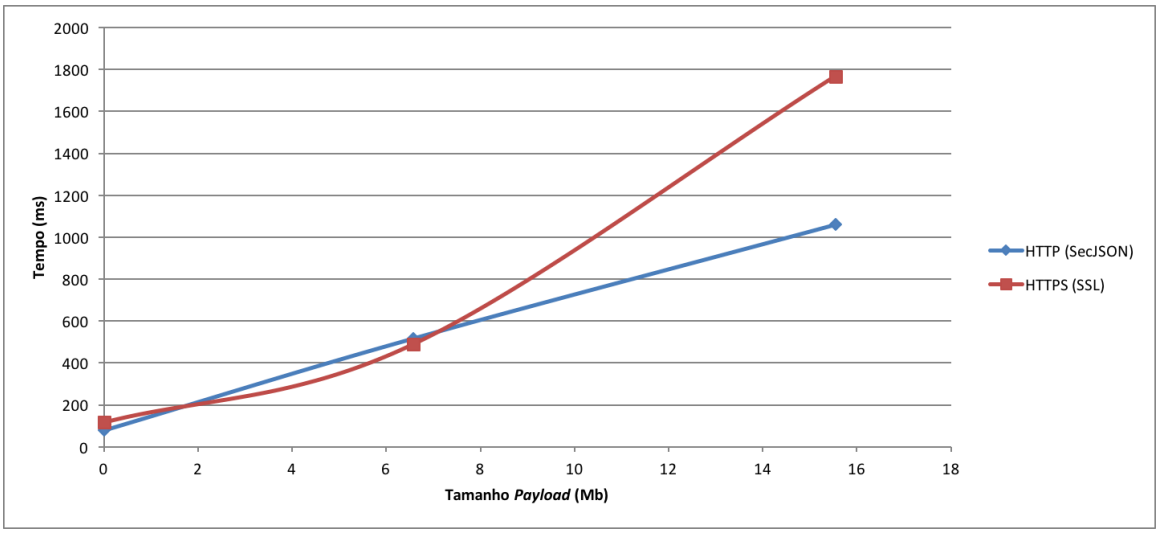

FIGURA 5.9: Transmissão em função do tamanho do *Payload*

Análise de resultados

Os resultados da transmissão entre dois nós mostram uma performance média superior do SecJSON em ficheiros de pequena dimensão e grande dimensão (ficheiros 1 e 3) mas ligeiramente inferior para ficheiros de média dimensão (ficheiro 2).

5.2 Transmissão para multiplos nós

Neste tipo de validação será efetuada uma transmissão do objeto para múltiplos nós (serviços), registando os tempos de transmissão de um nó (cliente) para quatro nós (servidor). A informação que se destina ao nó servidor n está na posição n do vetor JSON do ficheiro em análise.

O objetivo deste teste será avaliar a possibilidade de aplicar múltiplas cifras sobre o mesmo documento JSON e transferir o mesmo para múltiplos destinatários que terão, unicamente, acesso à informação a si destinada. Além disso será ainda comparado ao processo de enviar a mesma informação utilizando o protocolo SSL/TSL.

Para calcular o tempo total foram registados os tempos individuais de cifra (c), serialização (s) e decifra (d) e calculado o seu somatório.

$$cifra = \sum_{i=1}^{4} c_i \tag{5.1}$$

$$serializao = \sum_{i=1}^{4} s_i \tag{5.2}$$

$$decifra = \sum_{i=1}^{4} d_i \tag{5.3}$$

Procedimento SecJSON:

1. Cifra SecJSON de quatro documentos JSON com diferentes chaves, destinadas a diferentes destinatários;
2. Serialização do documento cifrado através do protocolo HTTP;
3. Cada nó servidor decifra o documento a ele destinado;

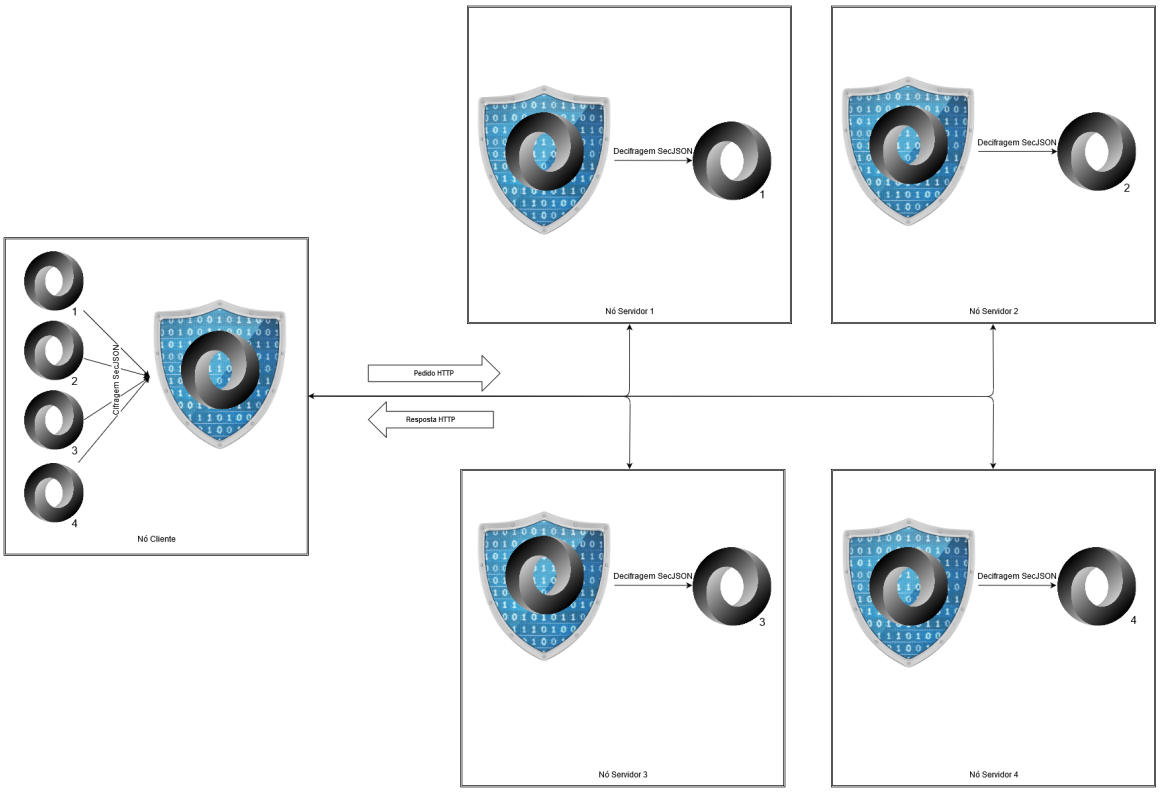

FIGURA 5.10: Transmissão SecJSON entre multiplos nós

Procedimento SSL:

1. Transmissão do documento cifrado através do protocolo HTTPS;

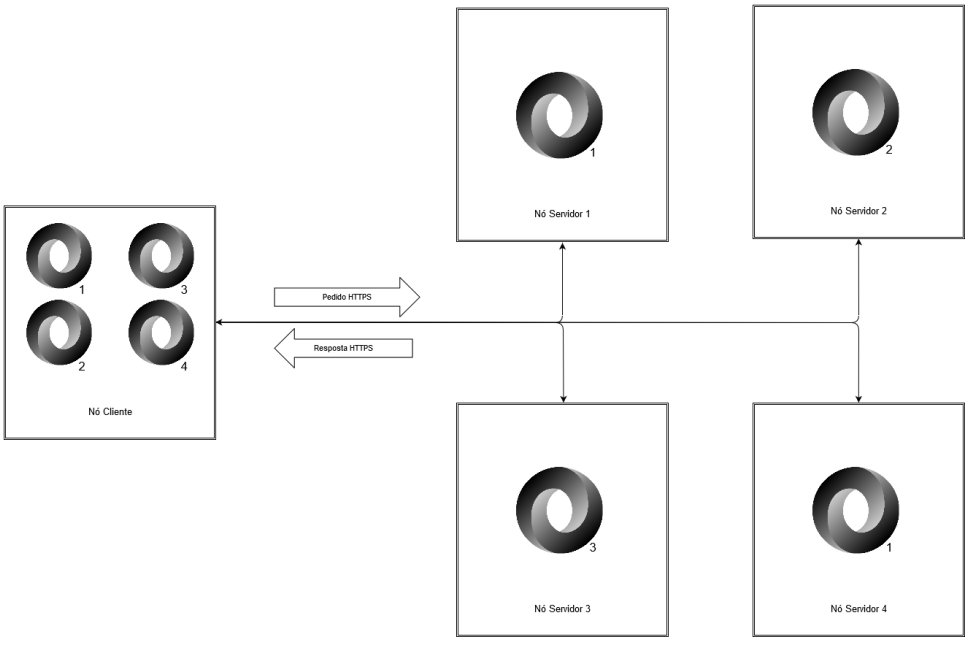

FIGURA 5.11: Transmissão SSL entre multiplos nós

De forma análoga ao teste anterior, os dois procedimentos foram repetidos vinte vezes para cada ficheiros e os resultados obtidos serão apresentados de seguida:

Ficheiro 1

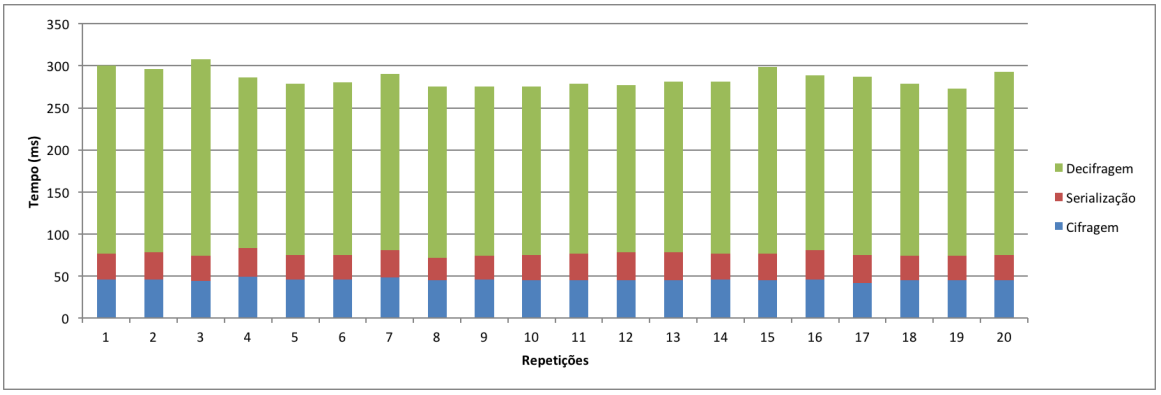

FIGURA 5.12: Transmissão SecJSON do Ficheiro 1

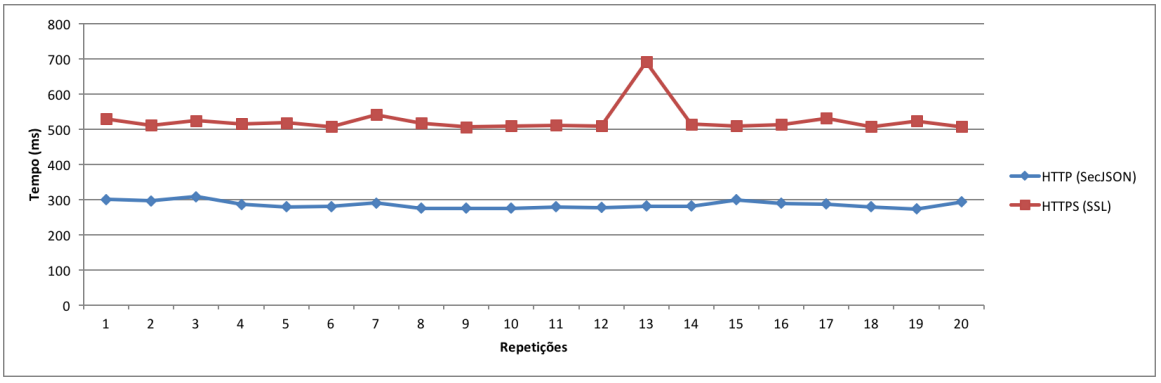

FIGURA 5.13: Transmissão SSL do Ficheiro 1

Ficheiro 2

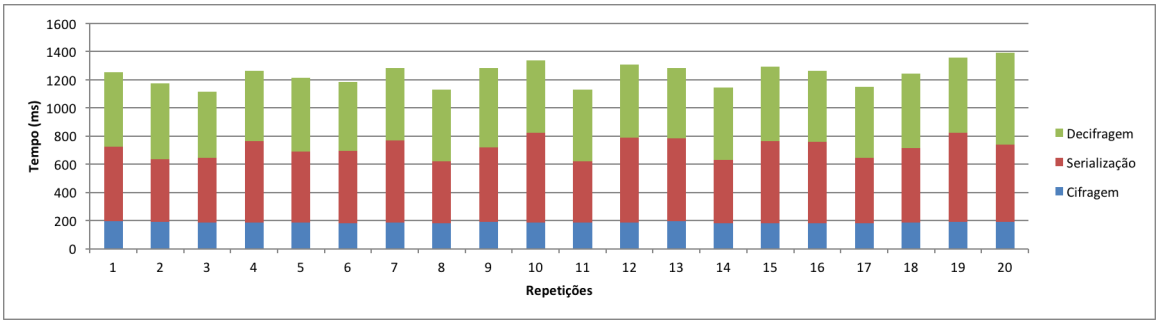

FIGURA 5.14: Transmissão SecJSON do Ficheiro 2

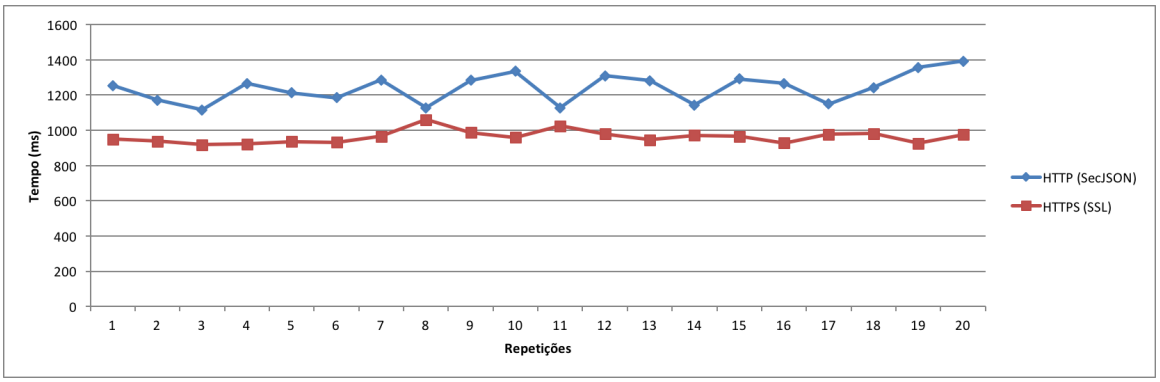

FIGURA 5.15: Transmissão SSL do Ficheiro 2

Ficheiro 3

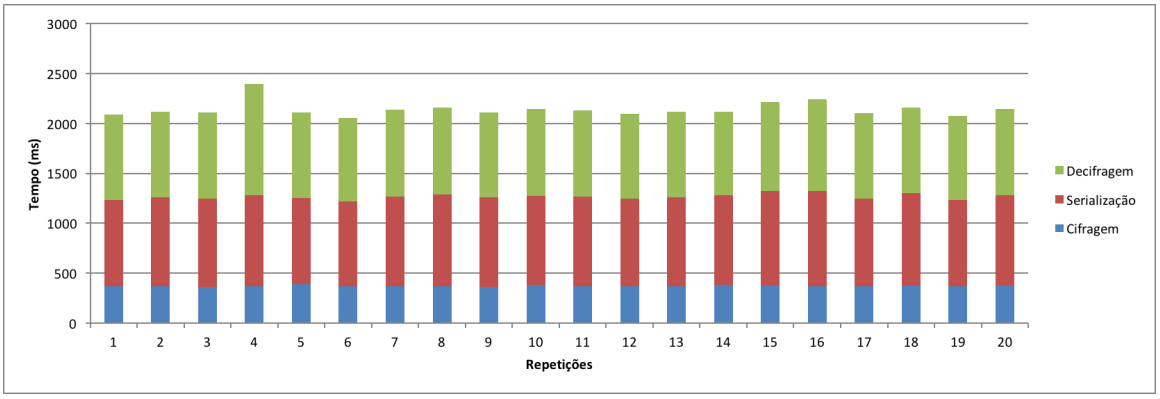

FIGURA 5.16: Transmissão SecJSON do Ficheiro 3

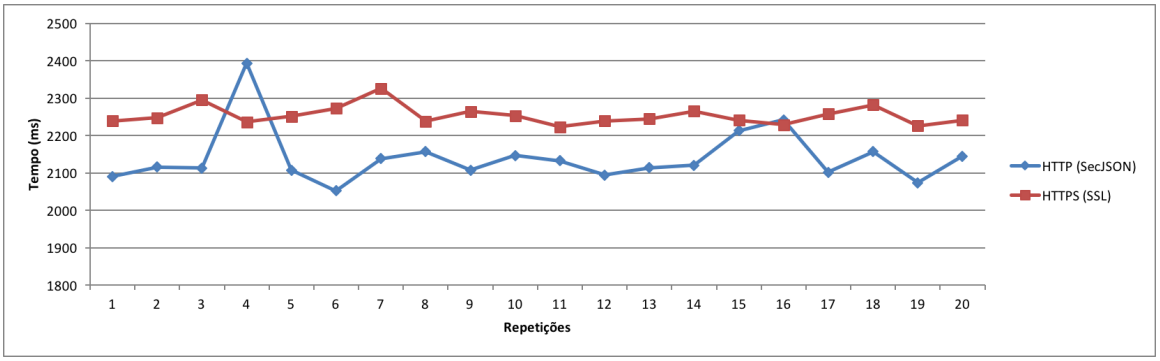

FIGURA 5.17: Transmissão SSL do Ficheiro 3

Uma vez mais, foi calculado o tempo médio para a serialização e processos criptográficos de cada ficheiro para comparar a sua evolução em função do tamanho do *payload*.

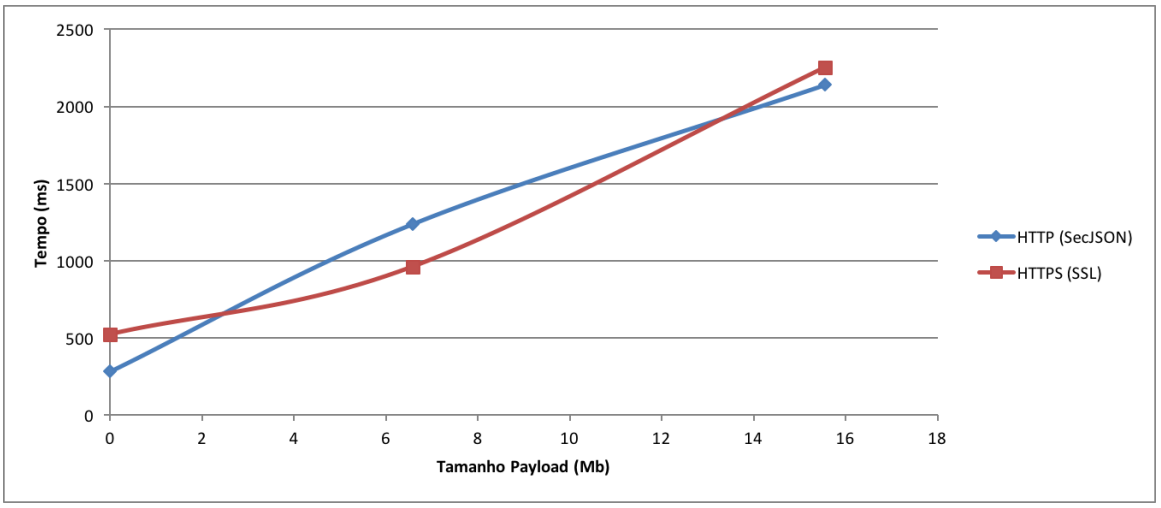

FIGURA 5.18: Transmissão em função do tamanho do *Payload*

Análise de resultados

Apesar de obter resultados similares relativamente à transmissão entre dois nós, os resultados da transmissão para multiplos nós mostram uma melhoria na performance do SecJSON para o ficheiro 1 e uma redução para os ficheiros 2 e 3.

A degradação de performance do SecJSON está relacionada com o tamanho do *payload* enviado para cada nó. O SecJSON cifra toda a informação a enviar para os quatro nós servidor e serializa o documento cifrado resultante para todos os destinatários. De forma inversa, o SSL/TLS apenas serializa a informação correspondente a cada destinatário.

5.3 Número de Nós

Com base nas validações realizadas nas secções 5.1 e 5.2 foi planeada uma validação que visa observar o comportamento do SecJSON em função do número de nós.

Com o crescimento do número de nós a informação presente no documento JSON aumenta consideravelmente, tornando-se relevante tentar perceber a evolução do SecJSON face ao SSL/TLS.

Ficheiro 1

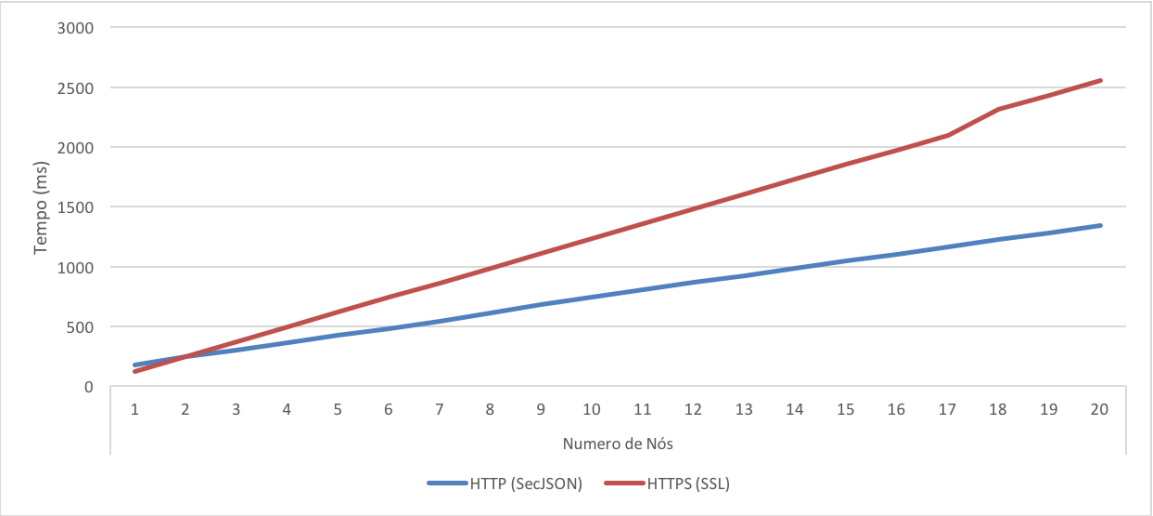

FIGURA 5.19: Evolução relativa ao numero nós - Ficheiro 1

Ficheiro 2

FIGURA 5.20: Evolução relativa ao numero nós - Ficheiro 2

Ficheiro 3

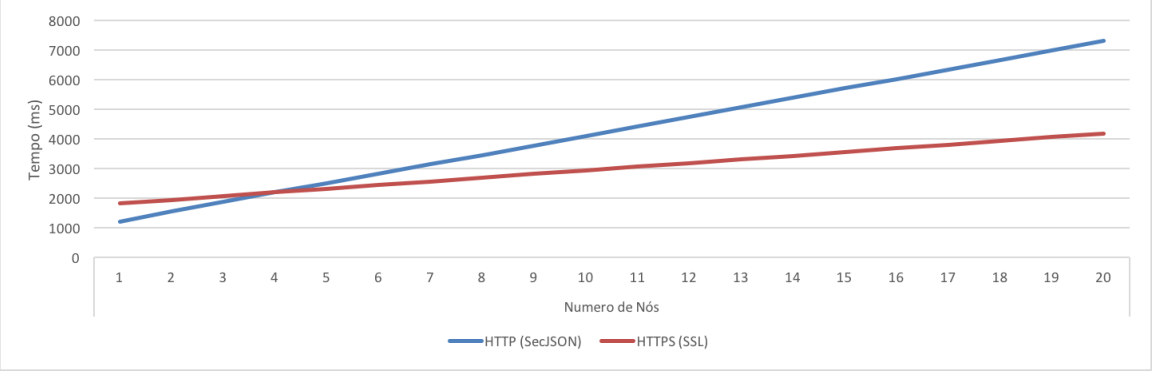

FIGURA 5.21: Evolução relativa ao numero nós - Ficheiro 3

Análise de resultados

Os resultados da performance do SecJSON em função do número de nós mostram apenas uma performance superior para o ficheiro 1, no entanto este é o único ficheiro que apresenta um tamanho constante serializado para cada nó servidor.

Nos ficheiros 2 e 3, o SSL/TLS apresenta um performance superior com o aumento do número de nós. Estes ficheiros possuem uma imagem codificada apenas na primeira posição do vetor, esta diferença justifica uma performance inicial superior do SecJSON e posterior quebra quando serializa esta informação para nós que não tem a sua chave criptográfica.

Conclusões

Durante o processo de validação, o mecanismo SecJSON apresentou-se como uma alternativa válida e complementar ao protocolo SSL/TLS.

Os casos de teste incidem principalmente na comparação de desempenho entre o SecJSON e o SSL/TLS, sendo que este foi um dos requisitos definidos na secção 3.1.1, garantir tempos de resposta nos processos criptográficos similares aos tempos apresentados pelo protocolo SSL/TLS. No entanto um sistema crítico deverá sempre acumular o maior número de mecanismo de segurança possível.

Capítulo 6

Conclusões

6.1 Conclusões Gerais

Um dos principais objetivos desta dissertação consistia no estudo, desenho, especificação e desenvolvimento de um mecanismo de segurança adicional para o formato JSON, designado ao longo deste trabalho como SecJSON, que oferecesse a possibilidade de proteger de forma granular o conteúdo de dados sensíveis neste formato. O JSON, um dos formatos de troca de informação que maior popularidade ganha nos nossos dias, e que permite a troca interoperável de informação entre múltiplos serviços e micro-serviços através da Internet, recorre essencialmente para garantir a segurança de informação a protocolos que oferecem segurança na camada de transporte, como é o caso do SSL/TLS. O SSL/TLS, apesar de ser um padrão de segurança que é largamente utilizado na Internet/WWW, possui algumas limitações. Uma das limitações mais importantes é que apenas oferece segurança entre ao nível da comunicação entre serviços e não ao nível das próprias aplicações desses mesmos serviços. Como tal, qualquer mensagem trocada através do SSL/TLS é cifrada na totalidade e decifrada igualmente no serviço receptor. Para determinadas aplicações em que as mensagens de destinem a diversos serviços ou utilizadores, em que seja necessário proteger os dados JSON com mecanismos de proteção ou

chaves critpográficas distintas, o SSL/TLS não é suficiente para responder a este tipo de requisitos.

Ao longo da revisão da literatura efectuada foi possível confirmar que existe uma ausência de mecanismos de segurança específicos para o JSON, apesar de já existirem alguns trabalhos em curso relacionados (JSON Web Key, JSON Web Signature, JSON Web Encryption e Javascript Object Signing and Encryption). No entanto, e apesar de existirem iniciativas interessantes para protecção de JSON, nenhuma delas ainda possua qualquer tipo de implementação, que permitisse que os programadores pudessem facilmente integrar esses mesmos mecanismos de segurança nos serviços que desenvolvem. Por outro lado, algumas destas iniciativas de proteção de JSON acabam por acrescentar complexidade aos próprios dados JSON ou aumentar a sua dimensão de forma considerável. O mecanismo SecJSON especificado e desenvolvido ao longo deste trabalho surge com uma solução de segurança adicional que aumenta a granularidade deste popular mecanismo de comunicação entre serviços Web. Por outro lado, outra das preocupações que foi central no trabalho desenvolvido prende-se com o facto de se tentar reaproveitar algum do trabalho que já existia anteriormente, nomeadamente na área da segurança dos serviços Web baseados em XML (WS-Security), garantindo assim que alguns dos mecanismos desenvolvidos eram já do conhecimento dos programadores de serviços e assim assegurando uma mais fácil adoção.

A especificação e implementação deste trabalho consistiu em três pontos essenciais: definição de uma sintaxe que permitisse a cifra e decifra de um documento JSON (baseando-se em princípios que já existiam na WS-Security), a implementação e disponibilização de um protótipo da sintaxe definida e validação da implementação realizada. A implementação centrou-se no desenvolvimento de uma biblioteca de software desenvolvida em Node.Js, que permite que os programadores possam facilmente integrar estas funcionalidades de segurança nos serviços desenvolvidos com esta tecnologia (uma das mais populares actualmente para desenvolvimento de serviços REST).

Ao longo do processo de validação da implementação realizada demonstrou que a solução SecJSON é uma alternativa válida e complementar ao protocolo SSL/TLS, onde apresentou desempenho superior em alguns dos cenários testados. No entanto, e apesar dos testes realizados incidirem principalmente na comparação de desempenho entre o SecJSON e o SSL/TLS, um sistema critico deverá acumular o maior número de mecanismos de segurança possível. Num cenário onde exista informação sensível redirecionada por múltiplas partes, sem que as mesmas sejam o destinatário final da informação, uma solução que combine o SSL/TLS com SecJSON permite proteger o canal de comunicação entre duas entidades e proteger a informação dirigida a uma terceira entidade. Assim, em termos de recomendação de segurança deste trabalho, apesar do SSL/TLS e o SecJSON poderem ser usados de forma estanque, a sua utilização conjunta confere um nível de segurança muito mais elevado.

Tendo em consideração o objetivo desta dissertação e a possibilidade de utilização do mesmo por terceiros, assim como o seu desenvolvimento futuro, foi disponibilizada a biblioteca do SecJSON através gestor de pacotes NPM (gestor de pacotes de Node.Js), podendo assim qualquer programador incorporar as funcionalidades SecJSON nos seus próprios serviços. Por outro lado, todo o código desenvolvido está disponível no Github para que outros possam contribuir para o seu futuro desenvolvimento. As localizações respectivas são as seguintes:

- https://www.npmjs.com/package/secjson
- https://github.com/tiagomistral/SecJSON

6.2 Contribuições

Tendo em consideração os objetivos propostos para esta dissertação assim como as questões de investigação, podem-se enumerar as seguintes contribuições:

1. Análise, desenho, especificação e desenvolvimento de um mecanismo de segurança para as mensagens trocadas entre serviços Web baseados no formato JSON, que permite a cifra parcial ou total do payload destas mensagens. A este mecanismo designamos como SecJSON.

2. Implementação e disponibilização de um protótipo do mecanismo SecJSON numa plataforma de desenvolvimento de aplicações Web.

3. Validação da implementação desenvolvida através da comparação com outros mecanismos de segurança já utilizadas.

Para além das contribuições referidas, acima foi produzido um artigo científico, *Secure Javascript Object Notation (SecJSON) - Enabling granular confidentiality and integrity of JSON documents*, tendo como base toda a implementação efetuada, conjunto de testes e explicação da abordagem seguida ao longo do desenvolvimento do mecanismo SecJSON, proporcionando assim um suporte à solução seguida e defendida. Este artigo foi submetido e aceite em duas conferências internacionais:

- SECRYPT'2016 – 13th International Conference on Security and Cryptography (*aceite como poster*)
- ICITST'2016 - International Conference for Internet Technology and Secured Transactions

6.3 Limitações e Trabalho Futuro

No âmbito de trabalho desenvolvido e tendo em conta as possíveis aplicações do SecJSON, apesar do mesmo respresentar um contributo importante para a segurança em serviços web baseados em trocas de informação que utilizem JSON, é possível apontar algumas direções de futuro no sentido de poderem ser desenvolvidos mais trabalhos de investigação nesta área. Seguidamente são propostas um

conjunto de trabalhos futuros que podem contribuir não só para a melhoria do trabalho desenvolvido, mas igualmente para o desenvolvimento de novos trabalhos de investigação.

- Numa altura em que todos os dispositivos estão cada vez mais interconectados à Internet, o advento da Internet das Coisas, ou Internet of Things (IoT) é uma área que desperta muito interesse na área da segurança de informação. São muitos os desafios de segurança que se apresentam nesta IoT e nomeadamente em termos da segurança e privacidade da informação que circula na mesma. Esta pode ser uma das áreas de aplicabilidade do SecJSON assim como do seu desenvolvimento futuro para ir de encontro aos requisitos específicos de segurança da IoT;

- Uma outra área em que o trabalho desenvolvido pode ter igualmente um desenvolvimento futuro é na área das bases de dados NoSQL. Não é apenas um requisito a segurança e a integridade da informação em trânsito, mas igualmente a segurança da informação que se encontra armazenada. Tendo em consideração que algumas destas bases de dados NoSQL usam como formato de representação de informação o JSON (o BSON – Binary JSON), a aplicação e futuro desenvolvimento do SecJSON para garantir a segurança de informação armazenada em bases de dados NoSQL seria um rumo de investigação muito interessante;

- Finalmente, e tendo em consideração frameworks de comunicação altamente distribuídos, que operam em modelos de publish/subscribe, seria ainda interessante aumentar a segurança das mesmas através da utilização do SecJSON. Um desses casos é o servidor de mensagens open source RabbitMQ, criado para suportar Advanced Message Queuing Protocol (AMQP) [40] e no qual a utilização de mecanismos de segurança granular, como é o caso do SecJSON, seriam importantes.

Bibliografia

[1] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. Xml-signature syntax and processing. *W3C recommendation*, 12:2002, 2002.

[2] Marcello Peixoto Bax and George Jamil Leal. Serviços web e a evolução dos serviços em ti. *DataGramaZero: Revista de Ciência da Informação. Rio de Janeiro*, 2(2), 2001.

[3] A Becker, D Claro, and J Sobral. Web services e xml um novo paradigma da computação distribuída. *Departamento de Informática e Estatística, Universidade Federal de Santa Catarina*, 2008.

[4] Luiz Eduardo Borges. *Python para desenvolvedores*. Novatec Editora, 2014.

[5] Adriana Manuela Ferreira Costa. Ferramenta de gestão de dados históricos. 2013.

[6] Douglas Crockford. Json specification. acedido a 14, 01, 2015, em: `http://www.ietf.org/rfc/rfc4627.txt`, 2006.

[7] Marta Angela de Almeida Sousa Cruz, Rômulo Mendes Figueiredo, and Rosangela Mourat da Rocha Ávila. Sistemas de controle de processos em ruby on rails. *Tecnologia & Cultura*, 13(18):69–77, 2011.

[8] Emerson Ribeiro de Mello, Michelle S Wangham, Joni da Silva Fraga, and Edson Camargo. Segurança em serviços web. *Livro de Minicursos do VI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. Santos: SBC*, pages 1–48, 2006.

[9] VerÔnica Lagrange Moutinho dos Reis. *Criptografia, Segurança de Dados e Privacidade-Até que ponto pode-se confiar na discrição dos computadores?* PhD thesis, UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 1989.

[10] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[11] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[12] Rúben Fonseca and Alberto Simões. Alternativas ao xml: Yaml e json. *Universidade do Minho*, Fev 2007.

[13] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[14] Alan Hevner and Samir Chatterjee. *Design science research in information systems*. Springer, 2010.

[15] Alan R. Hevner. A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19, 2007.

[16] Takeshi Imamura, Blair Dillaway, and Ed Simon. Xml encryption syntax and processing. *W3C recommendation*, 12:2002, 2002.

[17] M Jones and J Hildebrand. Json web encryption (jwe). *IETF*, Internet Draft, 2014.

[18] Michael Jones. Json web key (jwk). *IETF*, Internet Draft, 2015.

[19] Michael Jones, Paul Tarjan, Yaron Goland, Nat Sakimura, John Bradley, John Panzer, and Dirk Balfanz. Json web signature (jws). *IETF*, Internet Draft, 2011.

[20] jose.readthedocs.io. Javascript object signing and encryption (jose), acedido a 18, 05, 2016, em: `http://jose.readthedocs.io/en/latest`.

[21] Daniel Pacheco Lacerda, Aline Dresch, Adriano Proença, and JAV Antunes Júnior. Design science research: método de pesquisa para a engenharia de produção. *Gestão & Produção*, 20(4):741–761, 2013.

[22] Susan Landau. Making sense from snowden: What's significant in the nsa surveillance revelations. *Security and Privacy, IEEE*, 11:53–63, 2013.

[23] Berin Lautenbach. Introduction to xml encryption and xml signature. *Information Security Technical Report*, 9:6–18, 2004.

[24] Alan Baronio Menegotto and Fábio Mierlo. A linguagem ruby, 2013.

[25] N. Nithin and Anupkumar Bongale. Xbmrsa: A new xml encryption algorithm. *World Congress on Information and Communication Technologies*, 2012.

[26] Node.js. About node.js, acedido a 23, 03, 2015, em: `https://nodejs.org/about/`, 2014.

[27] Carlos Rodrigues, José Afonso, and Paulo Tomé. *ENTERprise information systems*, volume 220, chapter Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML, pages 162–169. Springer Berlin Heidelberg, 2011.

[28] rubygems.org. Rubygems basics, acedido a 25, 01, 2016, em: `http://guides.rubygems.org/rubygems-basics/`.

[29] Ludwig Seitz, Göran Selander, and Christian Gehrmann. Authorization framework for the internet-of-things. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–6. IEEE, 2013.

[30] Mohit Sethi, Jari Arkko, and Ari Keranen. End-to-end security for sleepy smart object networks. In *Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on*, pages 964–972. IEEE, 2012.

[31] Charles Severance. Discovering javascript object notation. *University of Michigan*, 2012.

[32] R.S. Shaw, Charlie Chen, Albert Harris, and Hui-Jou Juang. The impact of information richness on information security awareness training effectiveness. 52:92–100, 2009.

[33] H. Simon. The sciences of artificial. *MIT Press*, 3rd Edition, 1996.

[34] VASM Souza. *Uma arquitetura orientada a serviços para desenvolvimento, gerenciamento e instalação de serviços de rede*. PhD thesis, Dissertação de Mestrado, Unicamp-FEECDCA. Orientador: Prof. Dr. Eleri Cardozo, 2006.

[35] D Sudharsan, J Adinarayana, S Ninomiya, M Hirafuji, and T Kiura. Dynamic real time distributed sensor network based database management system using xml, java and php technologies. *International Journal of Database Management Systems*, 4(1):18, 2012.

[36] Stephen Thomas. *SSL an TLS Essentials*. Wiley Computer Publishing, 2000.

[37] TIOBE. Tiobe index for july 2015, acedido a 02, 07, 2015, em: `http://www.tiobe.com/tiobe_index`.

[38] Google Trends. Interesse ao longo do tempo - xml vs json, acedido a 10, 10, 2015, em: `http://www.google.com/trends/explore?q=xml+api#q=xml%20api%2C%20json%20api%2C%20xml%20webservices%2C%20json%20rest&cmpt=q&tz=Etc%2FGMT`.

[39] Nramanujam Venkatraman. It-enabled business transformation: from automation to business scope redefinition. *Sloan management review*, 35:73–73, 1994.

[40] Alvaro Videla and Jason JW Williams. *RabbitMQ in action*. Manning, 2012.