



**JOÃO PEDRO DE
OLIVEIRA MARTINS
PIRES**

**SEGURANÇA EM APLICAÇÕES DE
E-COMMERCE COM DESCONTOS
PERSONALIZADOS**



Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

**JOÃO PEDRO DE
OLIVEIRA MARTINS
PIRES**

**SEGURANÇA EM APLICAÇÕES DE
E-COMMERCE COM DESCONTOS
PERSONALIZADOS**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor João Paulo Barraca, Professor Doutorado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

Presidente

Prof. Doutor André Ventura da Cruz Marnoto Zúquete
Professor auxiliar da Universidade de Aveiro

Doutor Alfredo Miguel Melo Matos
Managing Partner da Metrify

Prof. Doutor João Paulo Silva Barraca
Assistente convidado da Universidade de Aveiro

agradecimentos

A realização desta Dissertação marca o fim de uma importante etapa da minha vida, e portanto gostaria de agradecer a todos aqueles que contribuíram de forma decisiva para a sua concretização.

Em primeiro lugar, ao meu orientador, Prof. Doutor João Paulo Barraca, agradeço pela oportunidade de realizar este projeto na empresa BeUbi, pela ajuda prestada ao longo deste percurso, através do constante acompanhamento e também por ter depositado a confiança em mim de que conseguiria concluir este projeto.

Em segundo lugar, à empresa BeUbi pela oportunidade dada para o desenvolvimento deste projeto, a ajuda disponibilizada pelos seus colaboradores, em especial ao António Howcroft e ao Rui Mendes.

Em terceiro lugar, à minha família, em especial aos meus pais, irmã e namorada que sempre me apoiaram durante este percurso, ajudando-me a superar as diversas barreiras.

Por último, gostaria de agradecer aos meus amigos e colegas de curso, pela disponibilidade e ajuda prestada para a conclusão deste percurso e também pelos bons momentos que me proporcionaram.

palavras-chave

segurança, *Optiprizer*, e-commerce, descontos, redes sociais, loja online.

resumo

Cada vez mais as redes sociais têm impacto na sociedade. Hoje em dia, é a plataforma líder de comunicação entre sujeitos de diferentes idades e é a mais utilizada para partilha de notícias, produtos ou serviços. O *e-commerce* também tem vindo a crescer e as redes sociais têm tido um papel fundamental para o seu florescimento, existindo cada vez mais estudos de marketing sobre como explorar o meio social no e-commerce. O *Optiprizer* surge nesta vertente, sendo um serviço que disponibiliza descontos a um indivíduo numa loja *online* de acordo com o seu perfil social, o seu poder de compra, os seus interesses, entre outros critérios. O objetivo deste trabalho é compreender e descrever como se pode adicionar uma camada de segurança ao *Optiprizer* de modo a que todo o processo de obtenção de desconto se possa fazer de forma segura. O *Optiprizer* é constituído por quatro fluxos primários: processo de ativação, obtenção de um desconto, adição de um produto ao carrinho com desconto e *checkout* desse mesmo produto. Assim, torna-se necessário assegurar que todas as entidades envolvidas no processo se encontrem protegidas nestes fluxos, de modo a evitar os variados ataques conhecidos sobre aplicações *web*. Para tal, desenvolvemos uma biblioteca de segurança que foi aplicada sobre o *Optiprizer* e sobre uma extensão para o *Woocommerce*, que comunica com o *Optiprizer* e implementa todos os fluxos, também desenvolvida por nós. Os testes funcionais realizados sobre a plataforma levam a concluir que a camada de segurança que desenvolvemos se comporta como esperado, assegurando todas as comunicações e evitando que os dados sejam adulterados por terceiros. Os resultados dos testes de carga sobre o *Optiprizer* com a nossa camada aplicada foram bastantes positivos, comprovando ainda que o consumo de serviços do *Optiprizer* com os dados protegidos através da nossa aplicação pouco influencia os tempos de resposta do servidor.

keywords

social network, e-commerce, *Optipricer*, discounts, security, online store.

abstract

Nowadays, social networks have an important role on our society, where people of different ages can communicate with each other using them to share important news, products or services. The e-commerce market also raised how long of this years, where people trust mucher on online stores to buy their products. Today, people buy much more online than in person. Has been proved that social networks has a big influence on e-commerce market where people trust more on reviews of the social profile of a certain store instead of trusting in experts. Marketing studies were made for evaluate the influence of social network on e-commerce market. The *Optipricer* product borned in that way. *Optipricer* is a product which combines the e-commerce world with social network. The main target of *Optipricer* is to increase a certain business, inserting that business in the social network. *Optipricer* rewards users which share a certain product in their own social media, providing a discount coupon which adapts to him: the value of discount is calculated based on user's interests, user's friends and user's influence on social networks. In that way, *Optipricer* tries to insert a certain product in online market. The main target of our project is to provide security on *Optipricer*, creating a layer which secure the communications between a certain store and *Optipricer*. The main activities to obtain and apply a discount from *Optipricer* are the follow: activate the store into *Optipricer*, obtain a discount remotely, add a product to a cart with that discount and checkout that product with discount. In that way, we developed a plugin for *Woocommerce* which tests the security of all communications applying our secure library. The functional and unit tests performed in our application proved that our security layer was well designed, which secure all communications between different entities. We also develop load tests to analyse the performance of our security library and we conclude that *Optipricer* with our security layer applied has a good performance, since our library doesn't influence so well the behavior of *Optipricer* plataform.

Índice

1. INTRODUÇÃO	1
2. E-COMMERCE E SEGURANÇA NA WEB	5
2.1 E-COMMERCE E REDES SOCIAIS	5
2.2 SEGURANÇA NA WEB	8
2.3 ATAQUES SOBRE APLICAÇÕES WEB	21
3. MECANISMOS CRIPTOGRÁFICOS	25
3.1 CIFRAS SIMÉTRICAS	25
3.2 CIFRAS ASSIMÉTRICAS.....	32
4. CASOS DE USO E INTERAÇÃO	41
4.1 DESCRIÇÃO DOS CASOS DE USO.....	41
4.2 ANÁLISE DE FLUXOS	42
4.2.1 Fluxo de ativação.....	43
4.2.2 Fluxo de obtenção de desconto	46
4.2.3 Fluxo de adição de um produto ao carrinho com desconto	54
4.2.4 Fluxo de checkout de um produto com desconto	56
5. COMUNICAÇÃO SEGURA NO OPTIPRICER	61
5.1 FLUXO DE ATIVAÇÃO.....	62
5.2 FLUXO DE OBTENÇÃO DE DESCONTO.....	65
5.3 FLUXO DE ADIÇÃO DO PRODUTO AO CARRINHO COM DESCONTO.....	70
5.4 FLUXO DE CHECKOUT DE UM PRODUTO COM DESCONTO	72
5.5 BIBLIOTECA DE SEGURANÇA.....	74
6. AVALIAÇÃO E RESULTADOS	77
6.1 TESTES FUNCIONAIS.....	77
6.2 TESTES DE CARGA.....	79
6.3 ANÁLISES DE ATAQUES / ERROS.....	83
6.4 TESTES UNITÁRIOS.....	90
7. SÍNTESE CONCLUSIVA	93
8. BIBLIOGRAFIA	95

Índice de figuras

Figura 1 - Plataformas de e-commerce mais utilizadas [5].....	6
Figura 2 - Popularidade das redes sociais	6
Figura 3 - Funcionamento de aplicações clássicas Web VS aplicações baseadas em AJAX	8
Figura 4 - Método de cifra do OTP Cookie Encryption [21].....	11
Figura 5 - Método de decifra do OTP Cookie Encryption [21]	11
Figura 6 - Diagrama de sequência do protocolo HTTPS	15
Figura 7 - Estudo de utilização de serviços Web [40]	17
Figura 8 - Termos pesquisados no Google (REST e SOAP) desde 2005 [41]	17
Figura 9 - Método "TokenPassword" para autenticação de Web Services	19
Figura 10 - Ataque Cross-site Scripting	21
Figura 11 - Cifra e Decifra no modo CBC [52]	26
Figura 12 - Dados a serem cifrados pelo algoritmo de benchmarking de cifras	28
Figura 13 – Gráfico com os tempos de cifra + decifra para os três melhores algoritmos de cifra por bloco, em todos os modos de cifra por bloco e algoritmos de cifra contínua VS tamanho do bloco a cifrar	28
Figura 14 – Gráfico com o tempo de cifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco a cifrar	29
Figura 15 – Gráfico com o tempo de decifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco que foi cifrado	30
Figura 16 – Gráfico com o tempo de cifra + decifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco a cifrar	31
Figura 17 - Gráfico com os tempos de assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar com chaves de 1024 bits	35
Figura 18 - Gráfico com os tempos de assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar com chaves de 2048 bits	36
Figura 19 - Gráfico com os tempos de verificação da assinatura para os diversos algoritmos de assinatura e síntese VS tamanho do bloco assinado, com chaves de 1024 bits	36
Figura 20 - Tempos de verificação da assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco assinado, com chaves de 2048 bits	37
Figura 21 – Gráfico com tempos de assinatura + verificação para os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar, com chaves de 1024 bits	38
Figura 22 – Gráfico com tempos de assinatura + verificação para os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar, com chaves de 2048 bits	39
Figura 23- Diagrama de casos de utilização do Optipricer	42
Figura 24 - Diagrama de Sequência do fluxo de ativação.....	44
Figura 25 - Documentação da API de contato do Optipricer	45

Figura 26 - Proposta de arquitetura para o fluxo de ativação	46
Figura 27 - Diagrama de Sequência de obtenção de um desconto	47
Figura 28 - Dados a serem enviados na comunicação page View() e createCoupon()	48
Figura 29 - Dados do cupão criado	49
Figura 30 - Dados do cupão criado assegurados.....	49
Figura 31 - Método da API do Optipricer para obtenção de descontos	51
Figura 32 - Documentação da API de obtenção de desconto (dados campo data)	52
Figura 33 - Documentação da API de obtenção de desconto (resposta com o cupão).....	53
Figura 34 - Diagrama final do fluxo de obtenção de um desconto.....	54
Figura 35 - Diagrama de Atividade de adição de um produto ao carrinho	55
Figura 36 - Diagrama de Sequência do processo de checkout (Redeem de cupões).....	57
Figura 37 - Dados a serem enviados no redeem	57
Figura 38 - Documentação da API de redeem de cupões.....	59
Figura 39 - Diagrama de arquitetura final do fluxo de checkout de cupão (redeem)	60
Figura 40 - Menu no backoffice do Wordpress para configurações do Optipricer	62
Figura 41 - Menu inicial de opções do Optipricer.....	62
Figura 42 - Ecrã de definições para submissão de informações de contato	63
Figura 43 - Ecrã de definições para configuração dos parâmetros do Optipricer na loja	63
Figura 44 - Definições da loja para os descontos provenientes do Optipricer	64
Figura 45 - Página de produto para obtenção de desconto, com o plugin do Optipricer, no Woocommerce ...	65
Figura 46 - Dados do produto e da loja “impressos” na página, a serem enviados para o Optipricer	66
Figura 47 - Login no Facebook a partir da aplicação do Optipricer (SWE)	67
Figura 48 - Pedido de permissões do Facebook para acesso a dados do perfil, por parte da aplicação do Optipricer.....	67
Figura 49 - Pedido de permissões para partilha do produto na rede social	67
Figura 50 - Página de produto durante a obtenção de desconto.....	68
Figura 51 - Desconto obtido pelo Optipricer.....	69
Figura 52 - Adição de um produto ao carrinho com desconto	70
Figura 53 - Carrinho de compras que contém produto com desconto	71
Figura 54 - Checkout de um produto com desconto	72
Figura 55 - Checkout concluído do produto com desconto	73
Figura 56 - Backoffice do Optipricer.....	73
Figura 57 - Função secureContent da biblioteca de segurança	75
Figura 58 - Dados retornados pela biblioteca de segurança no processo de cifra	76
Figura 59 – Função getContent da biblioteca de segurança.....	76
Figura 60 - Headers do método POST request api/coupon.....	77
Figura 61 - Dados não protegidos pelo pedido createCoupon.....	78
Figura 62 - Dados protegidos (cifrados) pelo pedido createCoupon.....	78
Figura 63 - Dados não protegidos enviados na resposta do createCoupon pelo Optipricer	79

Figura 64 - Dados protegidos (assinados) enviados na resposta do createCoupon pelo Optipricer	79
Figura 65 - Comportamento da aplicação para os diferentes cenários.....	80
Figura 66 - Gráfico de testes de carga (pedidos p/ seg) para os cenários A1, A2, A3 e A4,	81
Figura 67 - Gráfico de testes de carga (pedidos p/ seg) para os cenários B1, B2, B3 e B4	81
Figura 68 - Pedidos falhados (erro 500) para testes de carga com os cenários A1, A2, A3 e A4.....	82
Figura 69 - Pedidos falhados (erro 500) para testes de carga com os cenários B1, B2, B3 e B4.....	82
Figura 70 - Conteúdo para criar cupão manipulado via JavaScript.....	83
Figura 71 - Resultado retornado pelo Optipricer quando o conteúdo para criação de cupão é manipulado	84
Figura 72 - Adição de um produto ao carrinho com a cookie manipulada (1)	85
Figura 73 - Adição de um produto ao carrinho com a cookie manipulada (2)	85
Figura 74 - Checkout de um produto com um cupão de desconto que não pertence ao utilizador (1)	86
Figura 75 - Checkout de um produto com um cupão de desconto que não pertence ao utilizador (2)	86
Figura 76 - Cupão manipulado pela loja (processo de checkout).....	87
Figura 77 - Checkout de um cupão manipulado pelo administrador da loja (1)	88
Figura 78 - Checkout de um cupão manipulado pelo administrador da loja (2)	88
Figura 79 - Checkout com cupão expirado (1).....	89
Figura 80 - Checkout com cupão expirado (2).....	89
Figura 81 - Testes unitários efetuados sobre a nossa plataforma.....	90

Lista de acrónimos

API – Application Programming Interface	SOAP – Simple Object Access Protocol
HTTP – HiperText Transfer Protocol	REST – Representational State Transfer
CMS – Content Management System	CRUD – Create, Remove, Update, Delete
AJAX – Asynchronous JavaScript and XML	CORS – Cross Origin Resource Sharing
XML – Extensible Markup Language	JSON – JavaScript Object Notation
HTTPS – HTTP com TLS	JSONP – JSON com padding
DOM – Document Object Model	ECB – Electronic CodeBook
IEFT – Internet Engineering Task Force	CBC – Cipher Block Chaining
RFC – Request for Comments	OFB – Output Feedback Block
HMAC – keyed-hash message authentication code	NOFB – Output Feedback Block c/ blocos de n bits
OTP – One-time Pad	IV – Initialization Vector
SSL – Secure Sockets Layer	DES – Data Encryption Standard
TLS – Transport Layer Security	3DES – Triple-DES
URL – Uniform Resource Locator	AES – Advanced Encryption Standard
HTML – HyperText Markup Language	WEP – Wired Equivalent Privacy
MITM – Man In The Middle	WPA – Wi-fi Protected Access
CA – Certification Authority	RSA – Rivest, Shamir e Adleman
SSH – Secure Shell	DSA – Digital Signature Algorithm
TCP – Transmission Control Protocol	DSS – Digital Signature Standard
SOA – Simple Object Access	SHA – Secure Hash Algorithm

1. Introdução

As redes sociais têm atualmente um grande impacto social, sendo líderes nos processos de comunicação generalizada entre sujeitos de diferentes idades e estatutos sociais. Os utilizadores usufruem desta tecnologia para estabelecerem contacto com os seus amigos e familiares, para se informarem, partilharem e acederem a vários produtos e serviços¹.

Por outro lado, o e-commerce enquanto tecnologia de comércio *online*, também tem tido uma grande influência no mercado, existindo cada vez mais plataformas de lojas *online*, como o *Shopify*, o *Woocommerce* (baseado em *Wordpress*), o *Prestashop*, o *Magento*, entre outras. Hoje em dia, é cada vez mais usual fazerem-se compras *online*, dado à sua simplicidade, comodidade, acessibilidade e segurança oferecida pelos diferentes meios de pagamento e confiança assegurada pela loja, prestadora do serviço.

Para um negócio, ter uma loja *online* não é sinónimo de ter um grande volume de vendas. Presentemente, as redes sociais podem ter grande impacto num negócio, principalmente *online*. Estudos revelam que uma partilha numa rede social pode ter mais impacto para as vendas de um certo produto do que críticas, pontuações ou mesmo marketing “boca a boca”.

De forma a potenciar um negócio *online*, inserindo-o no mundo social, surgiu o *Optipricer*. O *Optipricer* é um produto que visa recompensar os utilizadores ao partilharem no seu meio social um determinado produto de uma dada loja. Deste modo, este serviço tende a adaptar-se ao utilizador, ou seja, gera um cupão de desconto para um produto de acordo com o poder de compra do cliente e a sua influência nas redes sociais. O *Optipricer* é uma ferramenta que funciona fora do meio da loja, ou seja, disponibiliza uma API (*Application programming interface*) para comunicar com a mesma. Assim, é necessário a criação de diversas extensões para diferentes plataformas de *e-commerce*, atuando estas sobre a loja e ajudando o cliente a adquirir um desconto.

Uma grande vantagem do *Optipricer* operar fora da loja é poder monitorizar todos os movimentos sociais de várias lojas e do cliente, criando análíticas e estatísticas sobre o comportamento dos utilizadores, a influência destes nas redes sociais, os produtos com maior peso no meio social, entre outros dados estatísticos.

O objetivo desta dissertação é procurar estudar uma forma de adicionar uma camada de segurança ao *Optipricer*, integrando-a no mesmo e numa extensão numa loja *online*. Para isso, o *Optipricer* será integrado

¹ Obtido através do getSocial (<http://getsocial.io/>)

no *Woocommerce*. É requisitado que no final as comunicações entre as duas entidades possam ser feitas de forma segura. Como o *browser* do cliente é uma entidade que intervém no processo, devemos garantir que a informação que passa por este não possa ser manipulada. Desta forma, devemos certificar que o *browser* do cliente é um canal seguro de troca de informação. Para tal, abordaremos as diferentes técnicas de segurança conhecidas para proteger comunicações sobre o protocolo HTTP (comunicações em serviços Web) e para impedir ou detetar a manipulação de dados da página Web e de *cookies* por parte do cliente.

A extensão do *Optipricer* para diversas plataformas de e-commerce incorpora a implementação de quatro fluxos primários: ativação do produto, obtenção do desconto remotamente, adição do produto ao carrinho com desconto e checkout do carrinho que contém o produto com o cupão de desconto emitido pelo *Optipricer*. Nestes fluxos é obrigatório garantir a segurança dos dados, bem como a privacidade de certas informações da loja e a autenticidade do conteúdo produzido pelo *Optipricer*, como por exemplo a emissão de cupões, evitando expressamente a sua manipulação por parte de terceiros.

Com base nos estudos efetuados, desenvolvemos uma biblioteca de segurança que protege informação através de mecanismos criptográficos, aplicando-a aos *plugins* do *Optipricer* para o *Magento* e o *Woocommerce*. Esta biblioteca pode ser aplicada noutros *plugins* do *Optipricer* para lojas *online* e também noutros contextos para proteção de informação. Além disso, o nosso projeto assegura a obtenção de um desconto remoto de forma segura numa loja *online*, através dos seus cupões de desconto, aplicando-os automaticamente ao carrinho de compras, ao contrário do que se verifica noutros trabalhos na área, onde os cupões teriam que ser aplicados manualmente ao carrinho pelo utilizador.

Inicialmente, estudámos todo o processo e todas as comunicações entre uma dada loja e o *Optipricer*, remodelando os casos de uso e diagramas de sequência que retratam todas as interações entre ambos. Deste modo, analisámos as possíveis vulnerabilidades do *Optipricer* e ataques passíveis de serem efetuados sobre a plataforma. Assim, refizemos toda a API de modo a recriar um processo seguro de troca de mensagens, com mecanismos de segurança adequados, selecionados por nós.

Além da implementação de uma biblioteca de segurança, desenvolvemos um *plugin* para o *Woocommerce* que visa testar todas as interações entre uma dada loja *online* e o *Optipricer*, comprovando a segurança das mesmas. Esta extensão também teve efeitos comerciais, o *Optipricer* passou a suportar mais uma plataforma de lojas *online*: o *Woocommerce*.

Por fim, de modo a validar todo este processo, desenvolvemos testes unitários que visam testar a nossa aplicação, testes funcionais sobre a nossa aplicação, de modo a testar todas as interações entre uma dada loja e o *Optipricer*, e ainda testes de carga de modo a avaliar o performance do *Optipricer* com a nossa biblioteca de segurança incluída.

No final, o *Optipricer* acolheu as nossas modificações, e a sua infraestrutura ficou mais completa e segura.

Tendo em atenção o projeto desenvolvido este documento encontra-se dividido em sete secções:

Inicialmente, faremos uma introdução sobre o problema, referindo o estado do e-commerce e das redes sociais na atualidade.

Seguidamente, referiremos alguns trabalhos desenvolvidos no âmbito esta área, bem como explicitaremos os principais conceitos relacionados com a segurança na Web e ataques / vulnerabilidades conhecidas.

O terceiro capítulo tem como tópico central a criptografia, analisando diferentes algoritmos e o seu comportamento em termos de performance.

No quarto capítulo, analisaremos a arquitetura do *Optipricer*, descrevendo cada fluxo, as mensagens que são trocadas nestes, potenciais ataques e vulnerabilidades, e soluções sucintas para os mesmos. Posto isto, apresentaremos uma arquitetura do produto final, com a adição da nossa camada de segurança.

No quinto capítulo, faremos a análise da implementação do produto por nós delineado, enunciando o comportamento da extensão do *Optipricer* para o *Woocommerce* em cada fluxo de comunicação.

No sexto capítulo, apresentaremos os resultados obtidos, correndo testes de carga antes e depois da aplicação da camada de segurança e também testes unitários sobre a camada de segurança bem como sobre o comportamento da aplicação com a introdução de ataques e erros nas mensagens.

Por último, tiraremos algumas conclusões sobre o trabalho desenvolvido, referindo as potencialidades do produto, bem como o trabalho futuro que pensamos poder desenvolver.

2. E-commerce e segurança na Web

Neste capítulo iremos apresentar o estado de arte referente à influência das redes sociais no e-commerce. Seguidamente, visto que no e-commerce é obrigatório referenciar segurança, faremos uma análise sobre segurança na Web, explicitando técnicas que se usam para proteção da informação e proteção dos clientes. Para isso, aludiremos a linguagens de programação Web e ainda técnicas de partilha de informações e de sessão entre clientes e servidores. Por fim, refletiremos sobre os diversos ataques conhecidos a aplicações Web.

2.1 E-commerce e Redes Sociais

A utilização da *Internet* cresce de dia para dia. Os utilizadores gostam de partilhar informação que se torna facilmente acessível para outros, utilizando a *Web* para esse feito. Estudos demonstram que entre 2000 e 2010, a taxa de crescimento de utilizadores da *Internet* ronda os 445%. Os utentes utilizam-na para entretenimento, trabalho, pesquisa, compras *online*, entre outras funcionalidades. Muitas destas potencialidades estão disponíveis através do protocolo HTTP (*HiperText Transfer Protocol*) [1].

O protocolo HTTP foi desenhado para ser *stateless*, ou seja, não tem um estado definido [2]. Este trabalha sobre a forma de pedido-resposta: inicialmente, o cliente envia um pedido ao servidor, e de seguida, o servidor processa o pedido e devolve a resposta. Para as comunicações seguintes, esta ligação entre cliente e servidor é esquecida. Deste modo, cada pedido do cliente é independente dos pedidos anteriores [3].

Uma das grandes utilizações da *Internet* são as compras *online*. Se há 20 anos atrás se falava que a compra e venda de produtos ou serviços era dependente de deslocações, hoje em dia fala-se em e-commerce, que também é uma atividade de compra e venda de produtos ou serviços mas via *Internet*. Para a criação de uma loja *Online*, é necessário considerar os dispositivos legais que atuam sobre o comércio *online*, a plataforma utilizada para construir a loja, ferramentas de marketing que farão lançar a loja no mundo online, os métodos de pagamento, e o mais importante, a segurança da loja [4].

A segurança no e-commerce não se refere apenas à segurança nas transações entre a loja e o cliente, mas também à proteção da integridade do negócio *online* [5]. Os negócios de e-commerce, de acordo com Holcombe [6], devem obedecer a quatro requisitos de segurança primários: 1) **privacidade**, as informações trocadas entre a loja e o utilizador devem ser asseguradas sobre partes não autorizadas, 2) **integridade**, os dados trocados não devem ser adulterados, 3) **autenticação**, o cliente e a loja devem provar a sua identidade e 4) **não-repudição**, requisição de uma prova de como as informações trocadas foram realmente recebidas.

A escolha da plataforma de e-commerce não é facilitada. Para *designers web*, devem ser cumpridos dois objetivos primários: a implementação de requisitos e a sua fácil atualização, e uma interface de fácil utilização. As plataformas de e-commerce dividem-se em comerciais, como *Bigcommerce*, *Volution*, *Shopify*, entre outras e *open-source*, como *Magento*, *Woocommerce*, *OpenCart* e *osCommerce*. A Figura 1 mostra um gráfico de um estudo realizado em 2014, enunciando as plataformas mais utilizadas de e-commerce. Nesta observamos que

o *Woocommerce* é a plataforma mais utilizada, pois é baseada em *Wordpress*, uma das plataformas CMS (*Content Management System*) mais poderosa e mais segura do mercado [4].

Com o rápido desenvolvimento da economia e o crescimento do e-commerce, os utilizadores do mercado *online* atingiram um estado de saturação. O desenvolvimento acelerado do meio social *online*, trouxe novas oportunidades aos comerciantes, permitindo a criação de um novo modelo de negócio, que se centra na combinação das redes sociais com o comércio online [7]. As redes sociais vieram revolucionar o modo como as pessoas comunicam e mantém relações entre si [8]. Hoje em dia, as pessoas gastam mais do seu tempo nas redes sociais do que a consultar o seu *email*, consumindo cerca de 4,6 horas semanais, contra as 4,4 horas semanais gastas no *email* [9]. O mundo social na web permite que certos indivíduos possam estabelecer ou manter relações com outros que partilham os mesmos interesses, por exemplo em contexto de trabalho (*LinkedIn*), música ou política (*MySpace*), académicos (o objetivo inicial do *Facebook*), entre outros [10]. De acordo com os autores de [11], o *Facebook* tem sido de longe a rede social mais popular. Esta rede social permite que os seus utilizadores possam ter o seu perfil, comentem e visualizem os perfis dos outros, criem ou se juntem a grupos do seu interesse e aprendam sobre os hábitos e *hobbies* de outros utilizadores como por exemplo, gostos musicais, filmes, desportos, entre outros [10]. A Figura 2 ilustra quais as redes sociais mais populares, num estudo desenvolvido entre 2012 e 2014 [11].

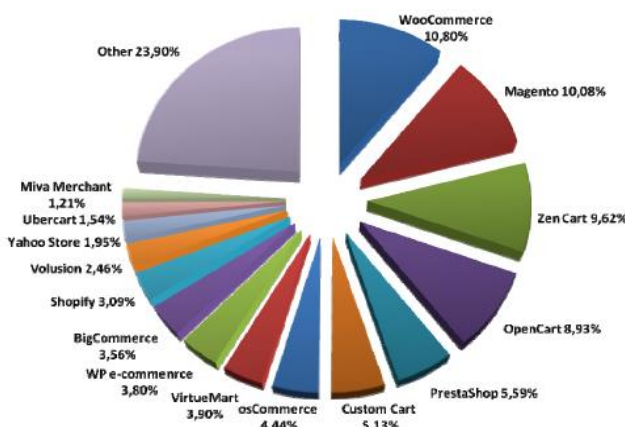


Figura 1 - Plataformas de e-commerce mais utilizadas [4]

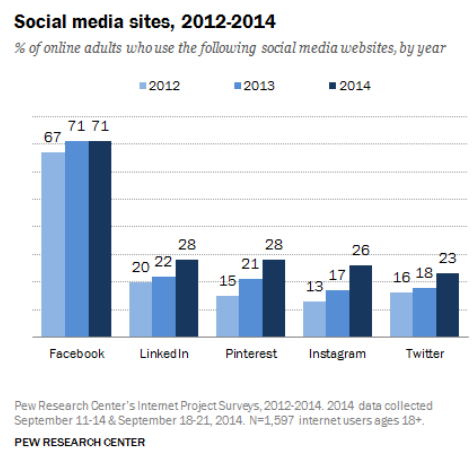


Figura 2 - Popularidade das redes sociais

Social e-commerce define-se como um método de *marketing* no e-commerce, que se centra na produção de conteúdo social. Devido ao grande crescimento de utilizadores nas redes sociais, as comunicações e interações entre utilizadores também aumentam. Deste modo, os utilizadores podem avaliar produtos, partilhar a sua experiência de compra, providenciando boa publicidade e *marketing* ao comércio *online* [7].

Para diversos segmentos de consumidores, as compras físicas sempre foram sociais: alguns autores referem o seguinte: “posso colocar um item no teu cesto de compras dizendo «isto é perfeito para ti»” [8]. Hoje em dia,

as empresas utilizam as redes sociais da mesma forma, como um lugar onde se pode transacionar produtos com os clientes e onde os clientes podem comprar em conjunto entre si.

Como já referido na introdução deste trabalho, o *Optipricer* centra-se na recompensa de utilizadores que partilhem no seu mural², numa certa rede social, a compra de um determinado artigo que efetuaram numa loja *online*, interligando desta forma o mundo social com o e-commerce. Existem produtos que se assemelham ao *Optipricer*, como é o caso do *getSocial*³, que consiste em adicionar botões de redes sociais (*Facebook*, *Twitter*, *Instagram*, entre outros), para a partilha de conteúdo da loja *online*, traçando estatísticas comerciais. Outro projeto semelhante ao *Optipricer* é o *addShoppers*⁴, que contém vários produtos adequados ao perfil do vendedor: estatísticas sociais, partilha de carrinhos de compras, partilha de um produto para obtenção de um cupão de desconto, entre outros⁵.

O *Optipricer* centra-se na geração de cupões, existindo também outras aplicações que disponibilizam cupões de desconto para os seus utilizadores. Uma delas, já referida anteriormente, designa-se por *addShoppers*. Nesta aplicação, de acordo com o artigo do blog [12], o cupão de desconto tem que ser criado na loja, e o respetivo código tem que ser adicionado no *backoffice* do *addShoppers*, de forma a quando a partilha, a aplicação do *addShoppers* possa disponibilizar o código do cupão. A comunicação com o *addShoppers*, a partir da loja é feita em *JavaScript*. Este método apresenta diversas desvantagens: uma vez gerado o cupão, com respetiva partilha, o código deste pode ser partilhado e reutilizado vezes sem conta; não há possibilidade de geração de descontos dinâmicos, ou seja, o valor do desconto é sempre o mesmo; e por último, o cupão não é aplicado automaticamente no carrinho de compras, o que torna a aplicação menos interativa e mais difícil de utilização. Para a plataforma *Woocommerce* existe um *plugin* de aplicação de descontos a partir de cupões: *Smart Coupons* [13]. Esta extensão centra-se na criação de cupões inteligentes, que se adequam ao utilizador, ou seja, para cada produto há a possibilidade de existirem múltiplos cupões de desconto, podendo uns ser mais benéficos na compra do que outros, dependendo da quantidade do produto obtido, por exemplo.

Uma vez que as aplicações que geram descontos dinâmicos numa loja *online* lidam monetariamente com o utilizador e com a loja, torna-se necessário estudar as diversas tecnologias associadas à Web, como é o caso do *JavaScript* e do AJAX (*Asynchronous JavaScript and XML*), mas também protocolos e sua segurança, como é o caso do HTTP e do HTTPS (HTTP com uma *Transport Layer Security*, TLS), protocolos de estabelecimento de sessão entre utilizadores e servidores e a sua segurança, como é o caso de *cookies* da web, e ainda dos possíveis ataques em ambientes de *e-commerce*. Nos próximos parágrafos faremos uma síntese sobre as virtualidades da conhecida linguagem de programação Web, o *JavaScript*, e a utilização do AJAX como motor de comunicação assíncrono com servidores.

² Mural define-se como um espaço do utilizador onde se pode visualizar as principais alterações ao seu perfil, os seus eventos, os seus *posts* entre outras informações.

³ <http://getsocial.io>

⁴ <http://addshoppers.com>

⁵ Os produtos do *addShoppers*: <http://addshoppers.com/apps/>

2.2 Segurança na Web

O *JavaScript* é uma linguagem “*client-side scripting*” que foi desenvolvida pela Netscape no final dos anos 90. Ao contrário de linguagens de script de servidor, como é o caso do *PHP*, *Perl*, *Ruby* ou *Python*, o *JavaScript* corre sobre o *browser* do cliente. Geralmente, esta linguagem é utilizada para manipular o DOM da página Web [14], mas existem outras utilizações, nomeadamente para correr jogos de consola, aplicações para *smartphone*, entre outras [15]. Deste modo, visto ser uma linguagem que corre no *browser* do cliente, a manipulação do código não afeta outros utilizadores, ou seja, todas as alterações feitas no código por parte do cliente apenas irão ser prejudiciais ou benéficas para ele.

Uma das utilidades do *JavaScript* é tornar as aplicações Web dinâmicas, recorrendo às suas comunicações assíncronas (AJAX). O AJAX contempla a utilização de código *JavaScript* para comunicações assíncronas onde os dados são enviados no formato XML (*Extensible Markup Language*) Esta ferramenta veio revolucionar o funcionamento clássico de aplicações Web, uma vez que anteriormente estas eram síncronas, dependendo do procedimento comum “pedido-resposta”, do processo HTTP. Hoje, com a utilização do AJAX, uma aplicação Web pode substituir o procedimento comum do processo HTTP por chamadas AJAX, evitando o recarregamento constante do DOM (*Document Object Model*) da página Web, bem como o processamento dos dados por parte dos servidores [16]. A Figura 3 ilustra a comparação de funcionamentos entre aplicações Web clássicas e aplicações Web que utilizam AJAX.

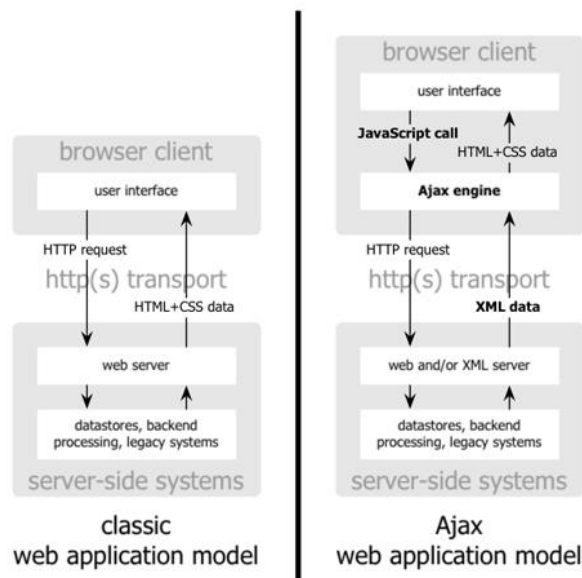


Figura 3 - Funcionamento de aplicações clássicas Web VS aplicações baseadas em AJAX

Embora as aplicações Web baseadas em AJAX sejam mais complexas, estas tornam-se mais dinâmicas para o utilizador, providenciando uma experiência na Web atrativa. Através das suas chamadas ao servidor, evitam o constante recarregamento de páginas e a espera pelos mesmos recarregamentos, no caso de aplicações Web

clássicas, sendo que os carregamentos mais demorados podem-se traduzir em páginas em branco enquanto carregadas, sem nenhum feedback para o utilizador. Assim, com o modelo de aplicações Web utilizando o AJAX, enquanto um utilizador navega numa página Web, certos conteúdos podem ir sendo carregados dinamicamente, sem que ele se aperceba do mesmo, tornando a aplicação mais leve e mais rápida.

Como definido anteriormente, o protocolo HTTP não tem estados definidos. A definição de estados no protocolo HTTP foi inserida mais tarde com a introdução de *Web Cookies*. As *cookies* foram inicialmente implementadas na Netscape em 1994. Em 1995, as *cookies* já eram suportadas pelo *browser Internet Explorer*, tornando esta especificação um *standard* [2]. Ao longo dos anos seguintes, foram feitos estudos e tentativas para tornar *cookies* uma norma *standard RFC (Request for Comments)*, sendo que em fevereiro de 1997, surgiu a especificação original “*HTTP State Management Mechanism*”, como norma IETF (*Internet Engineering Task Force*) RFC 2109. Visto não ser uma norma final, esta apenas surgiu em outubro do ano 2000, descrita como RFC 2965 [17].

Normalmente, as aplicações Web utilizam *cookies* para manter um estado na ligação. Uma *cookie* é um pedaço de informação que guarda o estado do cliente. Quando o servidor precisa de saber o estado do cliente, o servidor cria uma *cookie* que contém informações do cliente (autenticação por exemplo), enviando-a de volta para o cliente. Este armazena a *cookie* no seu *browser*. Mais tarde, o cliente reenvia a *cookie* a cada pedido ao servidor, dando-lhe autenticação [3]. A *cookie* é utilizada para autenticar utilizadores, manter um *tracking* de uma sessão e relembrar informação específica de utilizadores, como por exemplo carrinhos de compra, sendo esta muito útil em aplicações de e-commerce. As *cookies* estão sujeitas a ataques devido às vulnerabilidades do protocolo HTTP, tais como, “roubo” de sessão, envenenamento de *cookies* e replicação de *cookies* [18].

As *cookies* são muito utilizadas no mundo do e-commerce, nomeadamente para o armazenamento do carrinho de compras (não há necessidade de o utilizador efetuar o login no site para adição de produtos ao carrinho), e também para o armazenamento de informação de sessão do utilizador (também utilizado por muitos outros *websites*). Sendo um dos grandes requisitos numa loja *online* a segurança do *website* (como discutida anteriormente), é necessário mencionar estudos sobre segurança em *cookies*, visto que estas lidam com dados sensíveis do utilizador e da loja.

Para implementação de um protocolo de segurança em *cookies* entre o *browser* do cliente e o servidor, segundo [3], existem quatro princípios básicos que devem ser respeitados:

- 1) **Autenticação**, o servidor deve conseguir verificar que um cliente foi autenticado dentro de um certo período de tempo. Portanto, um cliente não pode forjar uma *cookie* válida. Numa sessão típica existem duas fases: login e pedidos seguintes. Na primeira fase, o cliente deve autenticar o servidor (com por exemplo um certificado de chave pública, SSL (Secure Socket Layer)) e vice-versa (nome do utilizador e password). Numa segunda fase (pedidos seguintes), o cliente envia a *cookie* segura ao servidor para ser autenticada. Se validada, a sessão continua estabelecida.

- 2) **Confidencialidade**, os conteúdos da *cookie* segura devem apenas ser lidos pelo servidor. Existem dois níveis de confidencialidade: *low-level confidentiality* que impede que qualquer parte (exceto servidor ou o cliente) consigam ler uma *cookie* e *high-level confidentiality*, onde apenas o servidor consegue ler o

conteúdo da *cookie*. Este conceito é utilizado por exemplo quando o servidor quiser armazenar informações do cliente na *cookie* que não lhe deseja revelar.

3) **Integridade**, o servidor deve conseguir detetar quando a *cookie* foi manipulada.

4) **Anti replicação**: no caso de um atacante replicar uma *cookie* forjada, um protocolo de *cookie* seguro deve detetar que a *cookie* é inválida. Nesse caso, o atacante estaria autenticado como sendo o cliente da *cookie* replicada.

Existem diversos protocolos de segurança em *cookies*. O primeiro foi proposto pelos autores de [19] em 2001 e consiste na inclusão de uma assinatura (HMAC) de diversos elementos que incorporam a *cookie* (*username* do cliente, tempo de validade da *cookie* e dados da *cookie*), com uma chave do servidor. Segundo [3], este protocolo apresenta diversas fraquezas: não providencia *high-level confidentiality*, onde os dados seguem visíveis na *cookie* e são validados apenas com a respetiva assinatura (HMAC, *keyed-Hash Message Authentication Code*), é vulnerável a ataques de replicação, uma vez que um atacante pode “roubar” uma *cookie* de um utilizador e utilizá-la para seu benefício, e por último, o seu mecanismo de defesa contra ataques de grande volume não é escalável.

O protocolo “*One-Time Pad (OTP) Cookie Encryption*” surgiu logo depois em 2002 [20]. Este protocolo tinha como objetivo proteger dados sensíveis do cliente, nomeadamente no âmbito do *e-commerce*, como dados de cartões de crédito. “*OTP Cookie Encryption*” é baseado no mecanismo OTP proposto em 1926 por Vernam [21] cujo objetivo era a cifra de comunicações (com e sem fio). O método proposto por [20] utilizava chaves geradas aleatoriamente para cada *cookie*, armazenando a chave de cifra, um id (referente à *cookie*) e o tempo de validade da mesma na base de dados. Os métodos de cifra e decifra propostos por este protocolo estão detalhados na Figura 4 e na Figura 5 [20].

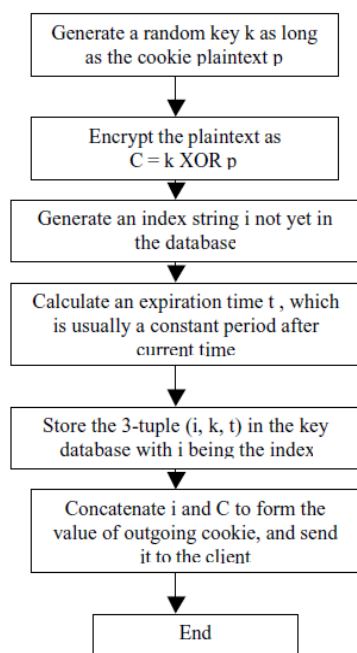


Figura 4 - Método de cifra do OTP Cookie Encryption [20]

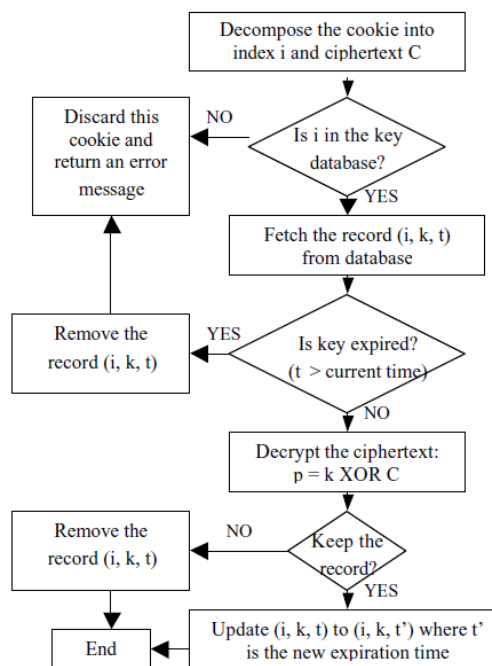


Figura 5 - Método de decifra do OTP Cookie Encryption [20]

Esta abordagem apresenta três desvantagens [18]: a primeira diz respeito à performance do protocolo, visto que sempre que é necessário analisar uma *cookie*, é obrigatória a consulta à base de dados e procura da respetiva chave; a segunda advém da remoção de uma chave antiga de cifra, provocando a invalidação de todas as *cookies* cifradas com aquela chave; por último, esta solução é utilizada para evitar a partilha de informação sensível com o cliente, mas poderá ser importante manter certas informações acessíveis ao cliente, o que este protocolo não assegura. Os nossos estudos sobre este protocolo acrescentam como desvantagem o facto de um servidor X não poder partilhar o conteúdo de uma certa *cookie* com um servidor Y, visto que apenas o servidor X sabe como interpretar essa mesma *cookie*.

Em 2005, surgiu uma nova proposta para implementar segurança em *cookies* [3]. Esta tinha por base utilizar uma chave diferente para cada *cookie*, que consistia num HMAC entre uma chave partilhada, o nome do utilizador e o tempo de validade da *cookie*. Esta chave era utilizada para cifrar os dados da *cookie* e criar um HMAC que além dos dados da *cookie* continha a chave partilhada. Este protocolo pode-se tornar lento com a aplicação consecutiva de HMAC e não evita ataques de replicação visto que duas *cookies* idênticas contêm o mesmo HMAC [18].

Por último, em 2009 surgiu uma ideia [18], que se centra na utilização de um *Reverse Proxy* protegido por uma *firewall*, entre o cliente e o servidor. Este consegue interpretar se as *cookies* do utilizador são válidas, se inválidas não as envia para o servidor, podendo evitar ataques de replicação. A *firewall* permite evitar comunicações externas (não aceites) com este *proxy* sendo que apenas aceita certos pacotes de utilizadores autorizados e aplicações web permitidas. Portanto, o *Reverse Proxy* tem como objetivo proteger e validar as *cookies*, cifrando-as e aplicando um HMAC para validar o conteúdo cifrado.

Findada esta síntese sobre protocolos de segurança em *cookies*, cabe-nos agora estudar que outras abordagens existem para a partilha de dados entre cliente e servidor. São tipicamente referidas diferentes técnicas e é ainda sugerida uma nova: o protocolo *Cookieless* [2].

Em primeiro lugar, estes autores [2] referenciam a técnica URL (*Uniform Resource Locator*) *Rewriting*, que consiste em enviar dados do utilizador através de uma *query string* no URL. Deste modo, os dados do cliente são enviados codificados no URL para o servidor, como parâmetros. Esta abordagem traz certas desvantagens: é um método pouco transparente e não automatizado, ou seja, o *website* tem que estar preparado para incluir sempre no URL os parâmetros do utilizador. Como problemas de utilização desta prática, ainda acrescentamos: URL's de grande dimensão, a codificação de caracteres estranhos e falta de privacidade caso o utilizador queira partilhar um certo URL com outro utilizador, poderá estar a partilhar do mesmo modo os seus dados, sem se aperceber.

Uma aproximação do protocolo anterior é a utilização URL *Rewriting* via *JavaScript*. Em contrapartida, nesta tática, a alteração do URL é efetuada através de *JavaScript*. Além das desvantagens do URL *Rewriting*, podemos acrescentar as seguintes: é necessário acrescentar programação adicional para incluir sempre, no URL, a sessão do cliente quando este navegar para uma outra página do servidor e nem todos os *browsers* (clientes) aceitam *JavaScript*.

Outra aproximação possível consiste em utilizar o endereço IP do cliente como identificador de sessão do mesmo. Hoje em dia, considera-se que vários utilizadores podem partilhar o mesmo endereço IP público quando se encontram na mesma rede, sendo portanto uma abordagem a não utilizar. Com a versatilidade dos dispositivos móveis, um utilizador troca facilmente de endereço IP sem se aperceber, sendo portanto mais uma desvantagem na utilização desta abordagem. Em suma, esta abordagem não garante persistência de sessão sendo, desse modo, uma técnica a não utilizar.

Por outro lado, é possível utilizar *frames*, que são usadas por algumas páginas HTML. As *frames* são traduzidas por parcelas, onde cada uma representa uma página HTML com um URL identificativo. Estas parcelas irão ser apresentadas todas juntas numa página web. A navegação em cada *frame* é independente, ou seja, a navegação numa determinada *frame* não afeta o comportamento das outras, apenas aquele pedaço da página é modificado. Deste modo, uma das *frames* poderá guardar a sessão do utilizador e manter-se intacta. As *frames* podem conter código em *JavaScript* que armazena informações de sessão e podem comunicar entre elas. Deste modo, pode existir uma *frame* estática que armazena a informação do cliente, enquanto a navegação do site é feita a partir de outra *frame*. A utilização de uma *frame* como estática força a lógica do site estar dividida em programação do lado do servidor e scripts do lado do cliente. Outra desvantagem é a utilização dos botões de retroceder e avançar, o que pode quebrar a lógica do *website*.

O protocolo HTTPS pode ser igualmente utilizado para manter uma associação segura entre o utilizador e o *website*. HTTPS é um protocolo baseado numa sessão TLS. Este protocolo será discutido e analisado mais adiante. Quando o pedido é efetuado para um URL em HTTPS, uma sessão TLS é criada entre o cliente e o servidor, e posteriormente reutilizada nos pedidos seguintes. O servidor pode assim confiar que todos os pedidos seguintes provêm do mesmo cliente. Esta abordagem poderá constituir uma boa solução, mas correr o

website em HTTPS é computacionalmente exigente, e além disso nem todos os *websites* correm sobre HTTPS, visto que um certificado para suportar HTTPS é dispendioso.

A autenticação via HTTP é uma outra possibilidade. Depois de um utilizador estar autenticado, o *browser* guarda em cache informações como o nome do utilizador e a sua *password* e estes são incluídos nos pedidos seguintes. A cada utilizador do *website* pode ser associado um ID que o servidor pode utilizar para associar os pedidos aos utilizadores. Para *websites* que requerem autenticação dos utilizadores, este tipo de autenticação pode ser o indicado. Assim, para outros *websites* que permitem utilizadores anónimos esta abordagem não irá funcionar, visto que a autenticação na página, utilizando este método, é sempre um requisito.

Por último, estes autores referem a utilização de *headers* opcionais pré-definidas pelo protocolo HTTP como uma possível solução. As *headers Referer* e *From* podem ser utilizadas para manter a persistência do utilizador. *From* é uma *header* opcional que, se presente, deve conter o endereço *email* do utilizador. Geralmente, nenhum *browser* envia esta *header*, não podendo portanto ser utilizada para manter sessão. *Referer* é uma outra *header* opcional que deve conter a URL da página da qual foi redirecionado para a página corrente. Utilizar esta *header* é uma forma de manter sessão entre utilizador e servidor, mas da mesma forma que muitos utilizadores não aceitam *cookies* devido a razões de privacidade, também muitos poderão configurar os seus *browsers* ou *firewalls* para não enviar a *header Referer*.

Terminado o estudo de outras abordagens existentes para conservação de sessão entre cliente e servidor, existe uma nova técnica [2], consertado num processo denominado de *Cookieless*, que se centra na utilização da *tag* HTML BASE para emular o URL *Rewriting* (especificado acima), utilizando-a para armazenar informações de sessão do utilizador. Este método apresenta as mesmas desvantagens da técnica URL *Rewriting*.

Além das técnicas enunciadas anteriormente, podemos acrescentar uma outra abordagem: *Browser Fingerprinting* [22]. Este algoritmo consiste em recolher características comuns e menos comuns presentes nos *browsers* tais como o *user-agent*, os *headers* aceites pelo protocolo HTTP, se o *browser* do utilizador aceita *cookies*, resolução do ecrã, *plugins* do *browser*, entre outros... Esta técnica não é muito fiável visto que existe uma pequena probabilidade de colisões, ou seja, dois *browsers* podem ser considerados iguais. A semelhança de outros *browsers* pode ser maior se o utilizador não disponibilizar determinadas informações do seu *browser*, com certas políticas de privacidade ativas, aumentando assim a probabilidade de colisões.

Concluída esta síntese sobre *cookies* e outras abordagens de estabelecimento de sessão no protocolo HTTP, é pertinente abordar agora os dois protocolos principais de comunicação na Web, o HTTP e o HTTPS, o seu funcionamento, as suas vulnerabilidades e possíveis ataques que poderão ocorrer com um e outro protocolo nas redes de comunicações.

A internet suporta dois protocolos bem conhecidos de transporte de informação: o HTTP e o HTTPS. Estes protocolos têm diferentes custos de mercado e disponibilizam diferentes graus de segurança. Por um lado temos o protocolo HTTP, com uma utilização praticamente gratuita por parte dos servidores, mas que não oferece qualquer segurança para aplicações Web. Por outro lado, podemos considerar o protocolo HTTPS que apesar do seu elevado custo justifica a segurança oferecida pelo mesmo, providenciando três garantias de segurança, já mencionadas anteriormente (autenticação, integridade de mensagens e confidencialidade das mesmas) [23].

Hoje em dia, o protocolo HTTP é utilizado por bastantes aplicações Web, devido à sua grande simplicidade e eficiência. No entanto, este protocolo torna-se bastante vulnerável a ataques [18]. Os ataques mais comuns são de injeção como *HTTP request smuggling*, *HTTP response splitting* e *cache poisoning* [1]. Neste protocolo os pacotes circulam em claro, ou seja, se interceptados por terceiros, tornam-se legíveis e compreensíveis, podendo ser adulterados. A técnica de escutar comunicações entre duas entidades e de as manipular denomina-se “*Man In The Middle Attack*” (MITM). Esta consiste em interceptar comunicações entre dois ou mais interlocutores que comunicam entre eles através de um canal inseguro. O ataque é bastante simples: consiste em visualizar ou mesmo modificar dados entre um cliente e um servidor. Se o ataque for bem-sucedido, os intervenientes não se apercebem que a comunicação foi forjada. Combater este ataque é um desafio e, para tal, devem ser desenvolvidas técnicas criptográficas para assegurar as comunicações vulneráveis [24].

Uma das formas de realizar um ataque MITM consiste em utilizar técnicas de *ARP spoofing* que consistem no envenenamento das tabelas de ARP de um determinado *router* e as de um dado utilizador, interceptando, deste modo, as comunicações entre ambos e redireccionando-as para estas entidades [25]. Outro método possível para atingir MITM denomina-se *DNS Spoofing*. Este consiste em envenenar o cliente, com um servidor DNS falso, que resolve os endereços DNS como o atacante desejar, enganando o utilizador e interceptando as suas comunicações [26]. As explorações do ataque MITM são diversas, desde interceptar, interpretar e manipular comunicações sobre o protocolo HTTP, interceptar sessões SSH (*Secure Shell*), com vista a usufruir de um computador ou servidor a que o atacante não tem acesso [27] e interceptar comunicações de *Web Services* se estas correrem sobre o protocolo HTTP, por exemplo entre um servidor e um dispositivo mobile. O conceito de *Web Service* será discutido mais adiante.

De modo a mitigar os ataques MITM, surgiu o protocolo HTTPS. Em 1994, a Netscape apresentou um protocolo que visava assegurar comunicações sensíveis, o SSL. Cinco anos depois, o IETF adotou-o como norma *standard* nomeando-a TLS [28], com vista a assegurar o protocolo HTTP, surgindo assim o protocolo HTTPS. Este protocolo, ao invés de utilizar a porta 80 predefinida pelo protocolo HTTP, utiliza o porto 443, encontrando-se as comunicações sobre esta porta asseguradas por uma camada TLS de cifra / autenticação entre os protocolos HTTP e TCP (*Transmission Control Protocol*) [29]. A maioria das comunicações na Web correm sobre o protocolo HTTPS, incluindo transações de e-commerce, bancos online e *email*. Este protocolo suporta uma infraestrutura de chave pública composta por milhares de CAs (*Certification Authority*)⁶. Assim, estas CAs responsabilizam-se por assinar digitalmente certificados de chave privada que associam um domínio a uma chave pública [30]. Se um certificado for assinado por uma CA de confiança, então podemos concluir que o certificado é de confiança [31], e consequentemente o *website* que o contém também é. De uma forma sucinta, a Figura 6 ilustra o funcionamento do protocolo HTTPS, com as mensagens trocadas [32]:

⁶ CAs: entidades que são da confiança dos *browsers* e que garantem a identificação de um servidor na Web

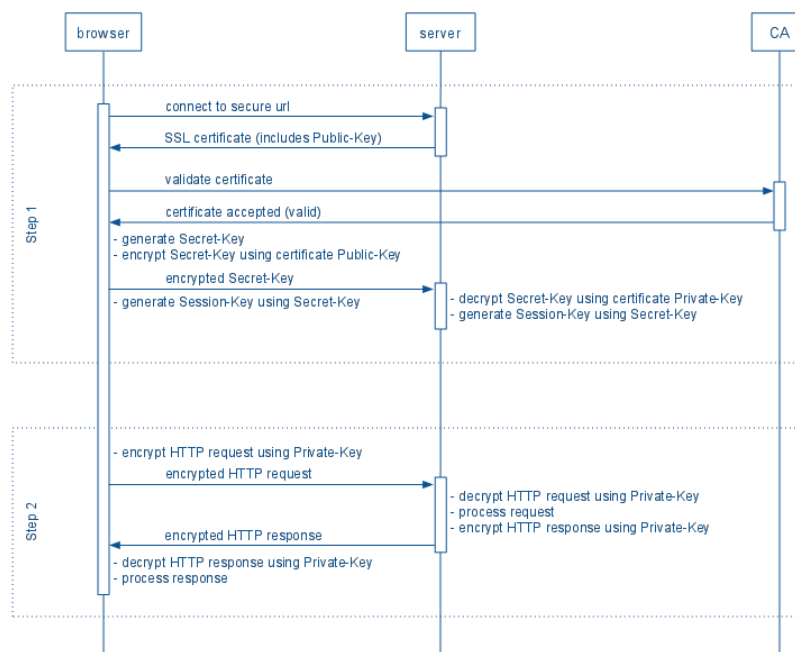


Figura 6 - Diagrama de sequência do protocolo HTTPS

No diagrama presente na Figura 6, verificamos como se comporta o protocolo HTTPS quando um utilizador inicia a sua navegação num *website* seguro. Inicialmente, o servidor envia um certificado SSL, que inclui a sua chave pública ao cliente, e o cliente (*browser*) trata de validá-lo com a CA correspondente, verificando se é de confiança. Se tal ocorrer, o certificado é aceite e a navegação no *website* é segura. De seguida, o *browser* gera uma chave secreta (segredo), cifra-a com a chave pública do servidor, e envia-a para o cliente. Esta chave será utilizada para gerar uma chave de sessão, e esta última corresponderá à chave de cifra e decifra de comunicações entre ambas as entidades. Quando o servidor recebe a chave partilhada cifrada, este trata de a decifrar com a sua chave privada, e gera uma chave de sessão a partir da mesma. Depois destes passos concluídos, é possível comunicar em segurança e privacidade entre ambas as entidades, mitigando o ataque MITM.

No entanto, segundo [27] e [29], é possível ocorrerem ataques do tipo MITM, se os certificados utilizados por um determinado *website* não forem de confiança ou se o utilizador aceitar certificados não confiáveis em comunicações futuras do mesmo *website*. Um certificado não confiável significa que não foi assinado por uma CA de confiança como, por exemplo, um certificado auto assinado. Este é facilmente gerável e pode ser utilizado para adulterar uma ligação HTTPS, tornando-a vulnerável, isto se o utilizador aceitar. Se um atacante conseguir interceptar o certificado enviado pelo servidor e enviar um certificado falso, saberá como cifrar e decifrar mensagens que circulam entre as duas entidades, pois o *browser* utiliza um certificado falso para a cifra da chave partilhada e o atacante consegue decifrá-la.

Sobre HTTP e HTTPS, além de páginas, são disponibilizados serviços, tipicamente denominados por serviço Web. Segundo W3C, o serviço Web define-se como um sistema de *software* que suporta interações de máquina para máquina numa determinada rede [33]. A utilização de serviços *Web* tem crescido nos últimos anos,

nomeadamente em plataformas de e-commerce, com vista a serem reduzidos custos de produção e comercialização, a diminuir o tempo de instalação do sistema e *debugging*, e ainda a melhorar a agilidade e flexibilidade da aplicação, na passagem do negócio para o mercado [34]. Os protocolos de comunicação de serviços Web mais conhecidos são o SOAP e o RESTful, e serão descritos e analisados a seguir.

O conceito de *Web Service* surgiu em 1999 e foi rapidamente adotado pela Microsoft, propondo o SOAP como protocolo base de comunicações em serviços Web. O SOAP (*Simple Object Access Protocol*) baseia-se na notação SOA (“Arquitetura Orientada a Serviços”), proposto em 1996, num artigo publicado por Yefim Natis, analista no Gartner. SOA é uma arquitetura que visa correlacionar serviços e consumidores de serviços que são modelados por *software*, dentro de uma grande empresa. Tipicamente, estes serviços são acedidos por um consumidor (*software*), “chamando” pelo seu nome, através de uma interface, operando num modo de pedido-resposta [35]. O SOAP define-se como um protocolo baseado em XML para troca de mensagens e chamadas de serviços remotos. Este não define nenhum protocolo novo de transporte, apenas opera sobre protocolos de transporte existentes, como é o caso do HTTP, SMTP (*Simple Mail Transfer Protocol*) ou MQSeries [36]. Uma mensagem SOAP é constituída por uma estrutura muito simples: um elemento XML com dois elementos “filhos”, *header*, onde estão presentes as *headers* da mensagem, como por exemplo o tipo de serviço (como POST ou GET) e *body* que representa o corpo da mensagem [36].

O protocolo RESTful (*Representational State Transfer*) de serviços Web começou a ser utilizado mais tarde, mas hoje é a arquitetura mais empregada na implementação de API’s de serviços Web, devido à sua simplicidade de construção e de utilização, acabando como domínio do mercado por parte do SOAP, verificado em anos anteriores. O conceito REST foi inicialmente proposto no ano 2000 por Roy Fielding, na sua tese de doutoramento [37], sugerindo o acesso simplificado a serviços Web, reutilizando tecnologias existentes, ao invés de adicionar novas camadas ao serviço de comunicação, como é o caso do SOAP. Deste modo, o serviço RESTful reutiliza o protocolo HTTP, para se servir de operações CRUD (*Create, Remove, Update, Delete*) [38]. O RESTful foi desenhado para ser um serviço Web orientado ao recurso, focando-se no objeto, diferenciando-se do SOAP, que foi sugerido como serviço orientado à ação. Os serviços Web do tipo RESTful atuam sobre a camada HTTP, podendo usufruir dos seus métodos (por exemplo GET, POST ou DELETE), projetando diferentes operações CRUD sobre um ou vários objetos [34]. Segundo os autores de [34], os pontos fortes de serviços RESTful são os seguintes:

- 1) **Interface uniforme**, os serviços Web são implementados a partir de métodos HTTP (por exemplo GET, POST, PUT ou DELETE);
- 2) **Utilização de JSON / POX**, o serviço Web pode utilizar dados formatados em JSON, POX ou texto plano, ao invés de dados tipo XML adotado pelo SOAP;
- 3) **Simplicidade nos requisitos do software**, este serviço utiliza URI para nomear cada recurso, facilitando ao utilizador o consumo do serviço Web a partir do *browser*, sem a necessidade de instalar novo software;
- 4) **Stateless** (sem estado definido), visto correr sobre o protocolo HTTP, a metodologia RESTful também é *stateless*, o que melhora a escalabilidade dos servidores;

5) **Utilização de Cache**, o utilizador guarda uma resposta a um pedido de serviço e utiliza-a novamente, convocando a cache do *browser*, sem necessidade de requerer um novo pedido ao serviço Web.

Com vista a analisar o estado de mercado relativo a serviços Web, foi efetuado um estudo em maio de 2010, que teve como base 2000 APIs Web listadas no serviço de diretórios *Programmable Web* [39]. A Figura 7 ilustra este estudo, que concluiu que 74% dos serviços usam REST, enquanto apenas cerca de 15% dos 2000 serviços estudados usam SOAP. Ainda é feita uma comparação a um estudo mais antigo, datado de 2008, que concluía que cerca de 60% dos serviços analisados utilizavam APIs REST contra os 25% serviços SOAP. Deste modo, podemos chegar à conclusão de que houve um acréscimo da utilização de serviços Web REST, e um decréscimo de APIs SOAP. A Figura 8 mostra o interesse demonstrado pelo público, através das pesquisas no *Google*, que nos mostram que ao longo dos anos o termo REST API tem sido cada vez mais pesquisado.

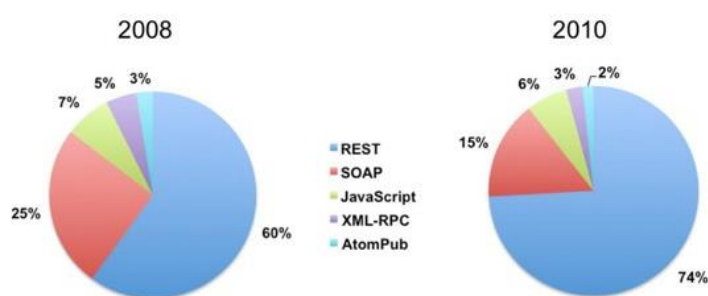


Figura 7 - Estudo de utilização de serviços Web [39]

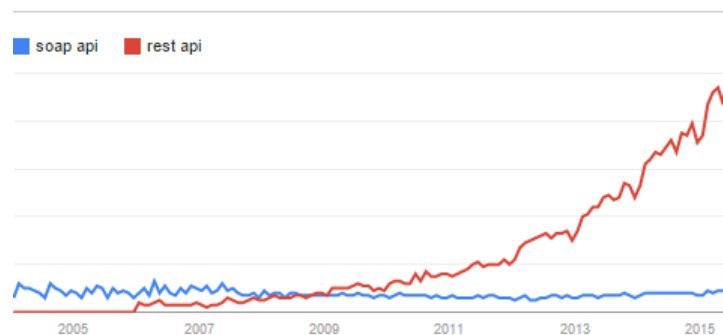


Figura 8 - Termos pesquisados no Google (REST e SOAP) desde 2005 [40]

Na altura da sua criação, as indústrias e comunidades envolvidas estavam preocupadas com aspetos de segurança e performance [41]. Devido à simplicidade e flexibilidade dos serviços mencionados, a performance dos mesmos é positiva. Quanto a aspetos de segurança os protocolos referidos apresentavam certas lacunas, sendo passíveis de ataques MITM, como mencionado anteriormente.

Relativamente ao protocolo de serviços Web RESTful, o método mais comum para proteção de dados consiste na utilização do protocolo HTTPS, cifrando assim as comunicações de forma “automática”, e invisível ao utilizador e evitando ataques MITM. Mesmo assim, foram efetuados outros estudos para a proteção de dados em *Web Services*, quando não existe a possibilidade de utilização do protocolo HTTPS ou quando existe um

MITM que é conhecido pelas duas entidades, no meio das comunicações, e que serve para reencaminhar as mensagens entre as duas entidades, mas que não deve poder manipular ou visualizar as mesmas, como é o caso do fluxo de obtenção de desconto do *Optipricer*, de que falaremos mais adiante.

Para implementar um protocolo seguro para consumo de serviços *Web* devemos ter presentes dois pontos. O primeiro relaciona-se com a autenticação de utilizadores para o recurso, ou seja, que utilizadores devem ter acesso a um determinado recurso protegido e como pode ser acedido de forma segura. Para isso, foram propostos dois protocolos conhecidos de autenticação em Web Services do tipo REST, o *HTTP Basic Authentication* e o *Digest Authorization*, e foi sugerido um outro, o “*UsernameToken*” [34]. O segundo ponto diz respeito à proteção das mensagens trocadas entre o utilizador e o serviço *Web*, ou seja, as mensagens não devem ser inteligíveis ou manipuladas por terceiros.

O modelo de Autenticação básica em HTTP (*HTTP Basic Authentication*) é do tipo desafio-resposta sendo utilizado por servidores HTTP com vista a validar a autenticação de *Web browsers*. Quando um cliente tenta aceder a um recurso protegido, a sua identificação será testada pelo servidor através de um desafio. O acesso ao recurso apenas será permitido se o utilizador responder corretamente ao desafio. *HTTP Basic Authentication* é um protocolo muito simples de autenticação de utilizador para acesso a recursos, mas não consegue garantir a autenticidade das entidades envolvidas. Como desvantagem, este protocolo utiliza codificação de base64, não cifrando e transmitindo *passwords* em texto plano. Portanto, este método de segurança é vulnerável a ataques de replicação.

A técnica “*Digest Authorization*” é referida pelos autores como sendo semelhante ao modelo anterior mas mais segura. Este protocolo utiliza a função de síntese MD5 para autenticar os clientes. Inicialmente o cliente requer o acesso ao recurso partilhado e recebe um texto aleatório gerado pelo servidor. O cliente gera uma assinatura (HMAC) com a sua *password* da *string* aleatória recebida pelo servidor e envia a mensagem. O servidor procura a *password* do cliente respetivo e gera a assinatura com base no algoritmo MD5, utilizando a *password* do cliente como chave (HMAC) e compara o resultado com a mensagem recebida do cliente. Se equivalerem, então o cliente está autorizado a ter acesso ao recurso. Embora muito mais segura que a técnica mencionada anteriormente, os autores referem uma vulnerabilidade desta técnica, designada por “*offline password guessing attack*”, que consiste em tentar “adivinhar” a *password* com múltiplas tentativas através da *hash* gerada anteriormente. Pela pesquisa realizada, acrescentamos ainda que o protocolo de síntese utilizado não está livre de colisões, podendo duas *passwords* sintetizadas equivalerem-se. Para resolver este último problema, sugere-se a utilização de um protocolo de *hashing* mais seguro, como é o caso do SHA-256.

Por fim, para providenciar autenticação em serviços *web* do tipo RESTful, foi proposto um novo protocolo ao qual é dado o nome de “*UsernameToken*”, que é comum utilizar-se em serviços *Web* do tipo SOAP no seu protocolo de segurança, o *WS-Security* [34]. O método proposto está enunciado na Figura 9. Inicialmente o cliente tenta aceder a um recurso e o servidor responde como não autorizado e envia um texto gerado aleatoriamente (*nonce*). O cliente terá que gerar duas chaves de sessão para se autenticar (*r1* e *r2*). Ambas as chaves são geradas a partir da sua *password*. O *r1* corresponde a uma síntese do campos *nonce*, *created* e *password* do utilizador através da função SHA1. O campo *nonce* corresponde a uma *string* aleatória criada

pelo utilizador, e o campo *created* representa a data de criação da resposta, por parte do utilizador. O *r2* é computado a partir da síntese de *h1*, *nonce*, *nc*, *cnonce*, *method* e *uri* pelo algoritmo MD5. O campo *h1* corresponde a uma síntese de *username*, *realm* (utilizado para identificar um grupo de recursos na internet) e *password* do utilizador. O campo *nc* corresponde a um id da mensagem, garantindo a não replicação de pedidos (se o servidor receber uma mensagem com os mesmos dados e *nc* então a mensagem foi replicada). O *method* representa o método do serviço Web e por fim o *uri* corresponde ao URL do serviço Web. No final, estes dados são enviados ao servidor e o servidor trata de ir buscar a *password* correspondente ao cliente, a partir da sua base de dados, e computa da mesma forma o *r1* e *r2*, comparando-os com os dados recebidos. Se ambos corresponderem, então o acesso ao recurso é garantido ao utilizador.

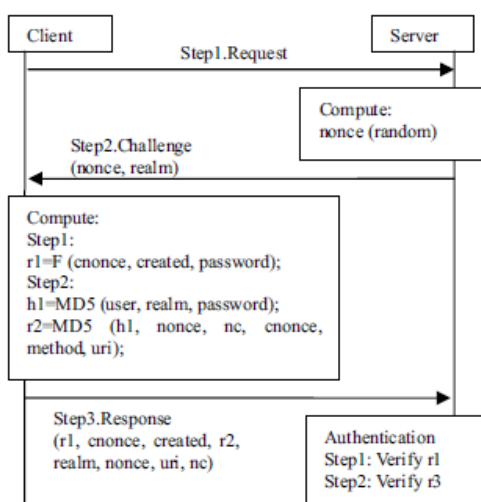


Figura 9 - Método "TokenPassword" para autenticação de Web Services

No entanto, um serviço de autenticação não é suficiente para garantir a segurança dos serviços Web. Se um cliente consumir um serviço de uma API, sobre o protocolo HTTP os dados circularão em pleno, podendo ser manipulados e vistos por um atacante no meio das comunicações. Deste modo, se o protocolo HTTPS não estiver ativo é necessário promover um protocolo seguro de circulação de dados de modo a proteger os dados sensíveis das comunicações. Assim, os autores o problema foi abordado e sugeridas duas possíveis abordagens criptográficas [38]: uma para dados sensíveis que não podem ser manipulados, mas que não é revelante que sejam visíveis, com a utilização de assinaturas digitais (criptografia assimétrica) e outra para dados sensíveis e privados, com a utilização de cifras, dando uso a criptografia simétrica e assimétrica.

O protocolo proposto para assegurar serviços Web utiliza diversos cabeçalhos que poderão definir algoritmos de síntese e de assinatura e algoritmos de cifra simétrica e assimétrica. Deste modo, o protocolo proposto é maleável e o cliente pode adotar um método específico de cifra ou um algoritmo de assinatura e o servidor consegue igualmente validar a mensagem recebida.

O método de assinaturas digitais proposto para assegurar serviços Web do tipo RESTful contém um cabeçalho, a assinatura dos dados presentes no *body* da mensagem. Para assinatura de mensagens, primeiramente é gerado uma síntese (*digValue*) dos seguintes dados: URL, algoritmo de assinatura, algoritmo de síntese, *id* do certificado a utilizar para validar a mensagem e ainda outros cabeçalhos a proteger. O valor da assinatura é gerado a partir da cifra do conteúdo do *digValue* com o algoritmo de assinatura especificado nos cabeçalhos e com a chave privada do emissor. Depois da mensagem recebida pelo recetor, este trata de gerar a síntese (*digValue*) da mesma forma que foi gerada anteriormente pelo remetente e valida-a com a síntese recebida. Se esta for válida prossegue na validação da mensagem, decifrando o valor da assinatura emitida pelo cliente (obtendo o *digValue*) e compara-o ao *digValue* recebido. Se ambos equivalerem então a mensagem é válida.

Se um conjunto de entidades desejar privacidade no *Web Service* implementado, deverá utilizar o segundo método [38], técnicas de cifra segundo criptografia simétrica e assimétrica. De uma forma sucinta, a criptografia assimétrica será utilizada para cifrar uma chave de sessão, e esta irá ser usada na cifra da mensagem a proteger (criptografia simétrica). Neste caso, um emissor da mensagem começa por gerar aleatoriamente uma chave partilhada, *skey*. De seguida, a mensagem a ser enviada é cifrada com essa chave. De igual forma, com a mesma chave, são cifradas todas as *headers* da mensagem e incluídas no *body* da mesma. A chave partilhada é cifrada a partir da chave pública do recetor (desta forma apenas este a pode decifrar), e incluída como *header* no pedido. Assim, revela-se mais eficiente do que cifrar a mensagem integral com criptografia assimétrica, visto que a cifra de dados de grandes dimensões torna-se dispendiosa. O mesmo não se pode dizer da utilização de chaves partilhadas, onde o processo é bastante mais rápido. A cifra da chave partilhada utilizando criptografia de chave pública é um processo rápido, visto que a chave de sessão é de pequenas dimensões. O processo de obtenção da mensagem em claro é efetuado de forma inversa: primeiramente é obtida a chave partilhada, depois decifra-se esta com a chave privada; de seguida é utilizada a chave de sessão obtida para a decifra da mensagem. Neste método de segurança, são incluídas diferentes *headers* no pedido: o *id* do certificado a utilizar para a chave privada, a chave partilhada (ou de sessão) cifrada, o algoritmo utilizado no processo de cifra simétrica (por exemplo AES128-CBC) e ainda o algoritmo utilizado no método de cifra assimétrica (por exemplo RSA-SHA1).

Além dos métodos propostos para eliminar ataques sobre comunicações Web, existe ainda uma outra técnica que merece a nossa atenção: o JSON Web Token (JWT) [42].

JWT é um formato de representação de dados compactado, em formato JSON, com intuito de proteger informação em ambientes com espaço limitado, tais como cabeçalhos de HTTP ou parâmetros de *Query URI*. Normalmente, esta técnica é utilizada para autenticação em serviços Web, em que os pedidos são autenticados através dos seus cabeçalhos HTTP. O JWT pode ser codificado numa estrutura JWS (*JSON Web Signature*) [43] e/ou numa estrutura JWE (*JSON Web Encryption*) [44]. A primeira técnica recorre a técnicas criptográficas para validar conteúdo com recurso a assinaturas digitais, criando um HMAC sobre os dados, em notação JSON. O JWS não evita a visibilidade dos dados, visto que os dados na ligação não vão cifrados, apenas protege a integridade dos mesmos. Quanto à segunda técnica (JWE), além de proteger os dados, evitando que estes sejam manipulados, assegura a privacidade dos mesmos, visto que os dados na mensagem são cifrados. No entanto, JSON Web Token apresenta uma grande vulnerabilidade [45]. Se o algoritmo de uma

mensagem assinada for substituído por “none”, a técnica de validação da mesma não assegura a integridade dos dados, retornando sempre como uma mensagem válida. Uma das formas de resolver este problema é evitar que se proceda à função de validação do JWT quando o algoritmo da mensagem é “none”, validando assim se o campo do algoritmo da mensagem se encontra corretamente preenchido.

Em suma, os métodos propostos mitigam ataques MITM a serviços Web RESTful, que operam sobre o protocolo HTTP. Embora as comunicações possam ser escutadas, a informação que circula nas mesmas não é perceptível pelo atacante.

Finda esta resenha sobre protocolos e serviços Web, iremos proceder à análise dos diferentes ataques conhecidos sobre aplicações Web, e que mecanismos de segurança existem para os banir. O ataque mais comum é o MITM, de que temos vindo a falar, tendo sido já discutido na secção dos protocolos da Web, tal como os mecanismos de segurança conhecidos para os diferentes serviços se protegerem de possíveis ataques.

2.3 Ataques sobre aplicações Web

Um ataque muito conhecido na Web e utilizado a partir de *JavaScript* denomina-se *Cross-site scripting (XSS)*. Este ataque ocorre quando páginas que são carregadas dinamicamente (com acesso a uma base de dados por exemplo) mostra conteúdo que não está propriamente autenticado. Deste modo, os atacantes comportam-se como utilizadores normais, fazendo o *upload* de código em *JavaScript* para o *Website* através de formulários com erros e que habilitam *Cross-site scripting*. Posteriormente, a vítima visita o *Website*, e o código *JavaScript* criado pelo atacante é carregado e executado na página, sem que a vítima se aperceba. A imagem seguinte mostra como se processa um ataque XSS [46]:

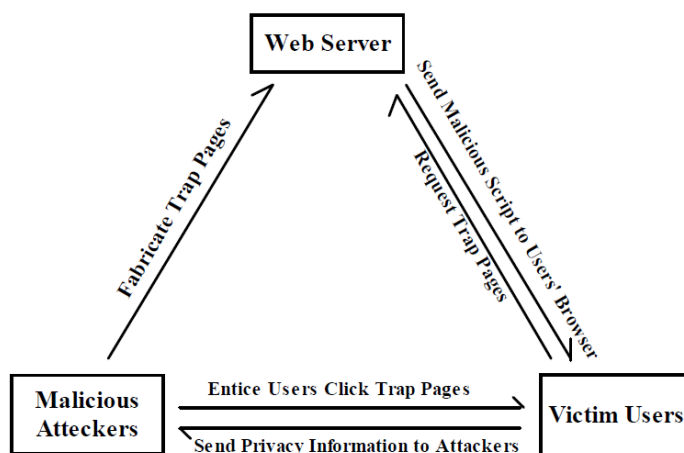


Figura 10 - Ataque Cross-site Scripting

Segundo a Figura 10, um atacante fabrica um código em *JavaScript* malicioso e envia-o para um servidor Web através do preenchimento de um formulário, por exemplo um comentário num *blog*. Quando a vítima visita o *WebSite* o *script* malicioso é carregado (como no exemplo anterior, a partir do comentário do *blog*), e desta forma o atacante consegue obter informações da vítima.

As *Cross-site Scripting* são a vulnerabilidade mais comum nas aplicações Web, tendo sido identificadas em mais de 80% dos *websites* (2006). XSS é um ataque que não atua no servidor, este é apenas o hospedeiro. Este ataque atua sobre o *browser* da vítima. Deste modo, o ataque *Cross-site Scripting* permite o furto de *web cookies*, que podem ser reutilizadas para aceder a contas da vítima, nomeadamente do banco, *email*, blogs, redes sociais, ... Para minimizar o efeito de furto de *cookies*, uma das soluções passa por incluir na *cookie* o *user-agent* do cliente, devidamente sintetizado, de forma a evitar que um atacante possa aceder à área pessoal da vítima, não tendo o mesmo *user-agent*. Este conceito de sessão é utilizado no *Facebook*, onde é feita a gestão de sessões por localização (através do IP) e por máquina / *browser* através do *user-agent*. Foi revelado que este tipo de ataques pode assumir o controlo total sobre o *browser* da vítima, como um *trojan*. Existem duas formas de os utilizadores serem vítimas de ataques XSS: através do ataque não persistente (quando o utilizador clica num link que remete para uma página *falsa*) ou por ataque persistente (quando o utilizador visita uma página comum no seu quotidiano e é injetado com código *JavaScript* malicioso) [47].

Com vista a minimizar este ataque, os programadores, devem-se precaver e evitar formulários mal programados, não permitindo a injeção de código em *JavaScript* nos mesmos. Uma política implementada pelos *browsers* da internet visa minimizar este tipo de ataque. A “*Same-origin Policy*” proíbe a partilha de dados de origens diferentes. Uma origem é constituída por um *host* (*WebSite*), o número da porta e o protocolo da aplicação. Basta um destes últimos ser diferente para a “*Same-origin Policy*” atuar: rejeita o acesso a documentos por parte de um *website* a outro, visto estes pertencerem a origens diferentes. Desta forma, a “*Same-origin Policy*” protege os utilizadores de modo a que ao visitarem *websites* maliciosos, estes não afetem a sua navegação em *websites* de confiança [48].

A definição de “*Same-Origin Policy*” foi reforçada [49], definindo-se como um modelo de “*sandbox*”, e apenas o *website* que armazena certa informação no *browser* pode ler ou modificar essa informação. Estes autores ainda acrescentam que a afirmação anterior não pode ser interpretada literalmente, visto que a web confia nas ligações internas entre os *websites*. Deste modo, o conjunto de “*Same-Origin Policies*” apenas interfere nas características individuais do *browser*, como *cookies*, acesso *JavaScript* a documentos ou *plugins* (*Flash*, *Adobe Reader* e *Java*). Estas políticas estão preparadas para permitir algumas exceções que possam ser benéficas para a cooperação entre *websites*. Estas exceções podem-se traduzir em comunicações assíncronas com serviços Web por parte de um *browser*, devendo este permitir que sejam consumidos serviços de *websites* externos, de uma origem diferente. Existem diversas técnicas que providenciam a possibilidade de um determinado *browser* comunicar com um serviço Web externo: utilização de *proxies* como intermediários da mensagem, *JSONP* e ainda *Cross-Origin Resource Sharing*.

De acordo com [49], podemos definir *proxy* como um website que se comporta como intermediário entre duas entidades, um cliente e um servidor, responsável por reencaminhar os dados provenientes do cliente para o servidor remoto de onde se quer consumir um determinado serviço Web. Assim, é necessário que o *proxy* se encontre na mesma origem do cliente, de modo a estes poderem comunicar entre si [48]. Esta técnica é funcional, mas mais lenta, visto que necessita de duas comunicações extra para comunicar com o servidor fora da origem.

Por outro lado, é possível utilizar JSONP (JSON com *padding*). O servidor web que disponibiliza o serviço deve suportar esta tecnologia, criando uma função que imprime o conteúdo a ser retornado pelo serviço em formato JSON (*JavaScript Object Notation*) ao invés de o retornar [48]. Esta função é convocada a partir do consumo de uma URL externa (como um link), utilizando esse *link* como fonte para um código *JavaScript* remoto. No entanto, se um atacante adulterar o *site* que disponibiliza o serviço Web, de modo a retornar código *JavaScript* malicioso (semelhante a um ataque XSS), o *browser* do cliente irá executar esse código. Contornando a “Same-Origin Policy”, utilizando JSONP, o programador introduziu uma vulnerabilidade de segurança.

Por último, existe a tecnologia *Cross-Origin Resource Sharing* (CORS): uma extensão ao protocolo HTTP para suportar pedidos de domínios diferentes [50]. Deste modo, o servidor remoto (de origem diferente) define se uma determinada origem pode aceder aos seus recursos ou não, sendo essa decisão forçada pelo *browser*. O servidor pode especificar que recursos são acessíveis do exterior e requerer autenticação do cliente para acesso a certos recursos, com o uso de credenciais (*cookies* ou autenticação sobre HTTP). Tecnicamente, o CORS adiciona “*request headers*” aos pedidos de modo a disponibilizar ao servidor certas informações como a origem ou a necessidade de credenciais, aos quais o servidor responde com “*response headers*” especificando o que o *browser* necessita de forçar. Esta especificação preserva a proteção de ambientes que não usam CORS, utilizando uma política de negação por defeito. Uma desvantagem da utilização desta abordagem é a dificuldade que um servidor remoto tem em distinguir pedidos provenientes de dois componentes diferentes dentro da mesma origem.

Além dos ataques mencionados acima, existe um outro, que tem como objetivo impedir que uma certa empresa ou serviço disponibilize um serviço normal aos seus clientes. Este ataque denomina-se DoS (Denial of Service) [31].

Concluída esta síntese sobre as diversas tecnologias da Web, os ataques conhecidos e protocolos de segurança para os combater, iremos procurar esclarecer a sua relevância para o nosso projeto.

Em primeiro lugar, a análise efetuada sobre segurança em *cookies* pode vir a ser útil se quisermos associar um desconto a um utilizador e mapeá-lo através de uma *cookie*. Visto que o detentor da *cookie* é o dono do desconto, e esta contém dados sensíveis para o utilizador e não pode ser forjada, é necessário garantir a sua segurança. Em segundo lugar a análise efetuada sobre protocolos de transporte na Web e serviços Web e a segurança dos mesmos, é relevante visto que o *Optipricer* disponibiliza uma API RESTful, com diversos recursos para obtenção de desconto, validação do mesmo, entre outros, e assim é necessário assegurar a informação que poderá circular entre a loja e o *Optipricer*, evitando que esta seja adulteradas. Por último, a análise efetuada sobre os ataques existentes em plataformas Web é-nos útil para estudar diferentes métodos que visam mitigar esses ataques, para que não ocorram na nossa plataforma.

O capítulo seguinte apresenta uma breve resenha teórica sobre técnicas criptográficas e descreve um estudo prático que efetuámos sobre os diferentes algoritmos existentes para cada técnica, que visa medir a performance de cada um e que nos ajudou na escolha da técnica mais rápida para o nosso projeto.

3. Mecanismos Criptográficos

Neste capítulo iremos apresentar um estudo empírico que efetuamos sobre diversos mecanismos criptográficos de modo a determinar quais os algoritmos mais adequados. Para isso, começaremos por lembrar alguns conceitos criptográficos, enunciando, seguidamente, os algoritmos mais utilizados.

Etimologicamente, criptografia significa a arte ou ciência de escrever de forma a ocultar conteúdo de uma mensagem (do grego *kryphós*, oculto, + *graph*, raiz de *graphein*, escrever). O objetivo da criptografia é, portanto, permitir que um conjunto de entidades (normalmente duas) possam trocar informação inteligível para terceiros [31].

A criptografia baseia-se no uso de cifras. Uma cifra é uma técnica criptográfica que permite, de uma forma específica, ocultar informação. Deste modo, uma cifra transforma texto claro em texto cifrado (ou criptograma). O processo inverso denomina-se de decifra, o qual permite transformar um texto cifrado no texto original em claro. Se a decifra for mal utilizada, o resultado é imprevisível, ou seja, não corresponderá ao texto previamente cifrado. A operação da maioria das cifras e decifras é especificada através de um algoritmo e de uma chave. O algoritmo define o modelo genérico de transformação de dados; a chave é um parâmetro do algoritmo que permite variar o seu comportamento de forma complexa [31].

3.1 Cifras simétricas

As cifras simétricas usam um valor comum, um *segredo* ou chave partilhada entre duas ou mais entidades. Só os detentores dessa chave secreta podem decifrar informação previamente cifrada com a mesma chave. Deste modo, as cifras simétricas permitem garantir a confidencialidade de dados conhecidos apenas por uma entidade ou garantir a confidencialidade de dados trocados entre duas ou mais entidades, partilhando a mesma chave.

As cifras simétricas têm como principal vantagem a sua eficiência. A desvantagem que apresentam prende-se com o facto de que num universo de N interlocutores, se se pretender ter uma chave secreta para cada par de interlocutores, serão necessárias $N * (N-1) / 2$ chaves. O principal problema no uso destas chaves é a sua distribuição segura, ao conjunto limitado de entidades que devem usar essas chaves [31].

O modo mais comum de utilização de cifras simétricas é o uso de cifras por bloco. Estas definem-se como cifras monoalfabéticas onde cada carácter do alfabeto original e do resultante é formado por conjuntos com dezenas, centenas ou mesmo milhares de bits. Ou seja, a informação, tanto em claro como cifrada, é sempre vista como uma sequência de blocos de dimensão constante de bits [31].

Existem diversos modos de cifra por bloco. Os mais conhecidos são o **ECB** (Electronic Codebook), o **CBC** (Cipher Block Chaining), o **CFB** (Cipher Feedback Mode) e o **OFB** (Output Feedback Block). Seguidamente

iremos detalhar como exemplo a utilização do modo de cifra por bloco **CBC**. Os restantes podem ser consultados no apêndice A.

No modo de cifra por bloco CBC, o primeiro bloco cifrado é gerado a partir do primeiro bloco, aplicando um *XOR* com um vetor de inicialização (IV). Este bloco (cifrado) é utilizado na cifra do bloco seguinte, sendo aplicado um *XOR* com o segundo bloco a cifrar. Deste modo, cada bloco sucessivo é cifrado a partir do bloco anterior cifrado, com a aplicação de um *XOR*, sendo desta forma cada bloco dependente do anterior, exceto o primeiro [51].

Na função de decifra, ao primeiro bloco cifrado é aplicado um *XOR* com o vetor de inicialização de onde resulta o primeiro bloco de texto. Nos blocos seguintes, é aplicado um *XOR* entre o bloco a decifrar e o bloco cifrado anterior a este [51].

No modo de cifra **CBC**, cada operação de cifra num bloco depende do bloco anterior (exceto a primeira que depende de um IV). Assim, o processo de cifra não pode ser feito em paralelo. No entanto, o processo de decifra já pode ser executado em paralelo, visto que os blocos cifrados já estão disponíveis para serem utilizados no processo de decifra [51]. Neste modo de cifra por bloco, o IV pode não ser secreto, mas não deve ser previsível, devendo ser gerado aleatoriamente.

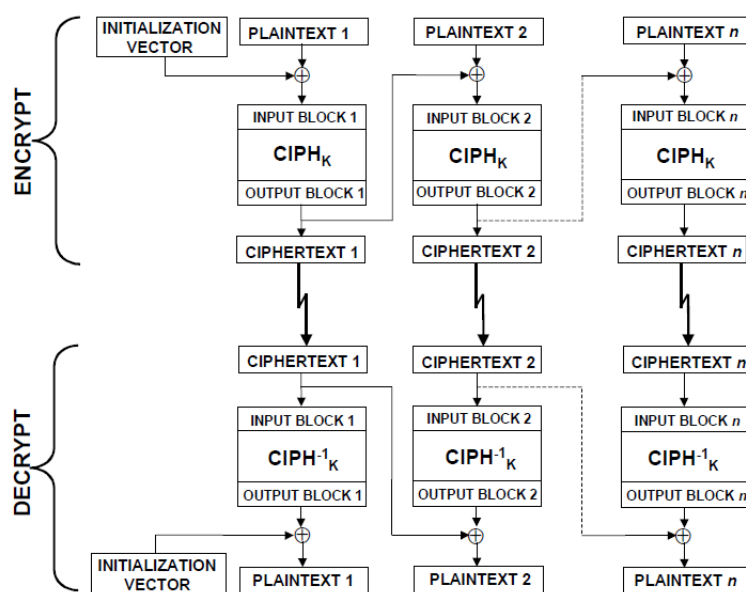


Figura 11 - Cifra e Decifra no modo CBC [51]

Este modo de cifra tem sido o mais utilizado na proteção de informação com recurso a criptografia simétrica, visto que reduz substancialmente o risco de replicação [31].

Com os diferentes modos de cifra por bloco podem ser utilizados diversos algoritmos. De seguida, irão ser descritos os algoritmos mais comuns.

O primeiro algoritmo reconhecido foi o **DES (Data Encryption Standard)**. **DES** é um algoritmo de permutação do tipo rede de *feistel* que foi implementado nos anos 70 do século XX. Este algoritmo de cifra utiliza uma chave de 56 *bits* e blocos de 64 *bits*, que pode ser facilmente comprometida utilizando métodos de *brute-force*. Foi um algoritmo muito utilizado, mas hoje em dia não é mais usado, tendo em conta as suas vulnerabilidades e *performance* [52].

De modo a combater as vulnerabilidades do **DES**, apareceu o **Triple-DES (3DES)**, desenvolvido entre 1997 e 1998. Este consiste numa atualização ao algoritmo **DES**, utilizando uma combinação de 3 operações **DES**: primeiro cifra, depois decifra e por último volta a cifrar, sempre com chaves diferentes, resultando numa chave de 168 *bits*. No entanto, esta modificação não melhorou a segurança do algoritmo, visto que em 1998 Stefan Lucks conseguiu quebrar este algoritmo em 2^{90} operações [53].

Em 1997, apareceu também o algoritmo **AES (Advanced Encryption Standard)** ou **Rijndael**, com o objetivo de combater as fraquezas do algoritmo **DES**, tal como no **Triple-DES**. Uma variante do algoritmo originalmente proposto por Joan Daemen e Vincent Rijmen foi aceite pela “*Federal Information Processing Standards*” (FIPS). Inicialmente, este algoritmo foi designado para ser utilizado com chaves de 128 *bits*, mas uns anos mais tarde, foi melhorado para aceitar chaves de 192 ou 256 *bits*. Neste momento é dos algoritmos mais populares devido à sua rapidez, segurança, fácil implementação e por não ser muito intenso em termos de memória, deixando para trás algoritmos como o **DES** e o **Triple-DES** [54].

Além dos algoritmos especificados anteriormente, existem outros também potentes, mas que ficam aquém do **AES**, como é o caso do **Blowfish**, que é um algoritmo *open-source*, não patenteado [54], o **CAST-128** e **CAST-256**, que foram aprovados pelo governo do Canadá [55], sendo ambos reconhecidos como norma RFC [56], [57], e o **GOST**, utilizado pelo governo da Rússia [58].

Além de cifras por bloco, existem cifras contínuas. O algoritmo mais comum e mais utilizado na internet para cifras contínuas é o **RC4 (Rivest Cipher 4)**. Este é comercial, ou seja, é pago. O **RC4** é utilizado na proteção de redes *wireless*, como é o caso da proteção WEP (*Wired Equivalent Privacy*) e WPA (*Wi-fi Protected Access*), de modo a assegurar as comunicações. Foi escolhido para redes sem fios visto ser um algoritmo resistente à transmissão de erros, ou seja, um erro na cifra apenas afeta poucos *bytes* da mensagem [54]. Como referenciado pelos autores em [59], este algoritmo apresenta várias vulnerabilidades pelo que não deve ser utilizado.

Uma vez concluída esta breve resenha sobre os diversos algoritmos de cifra simétrica e os modos de cifra contínua e por bloco, nos parágrafos seguintes iremos apresentar o estudo que efetuámos sobre estes algoritmos, que visa em reconhecer quais os algoritmos com melhor comportamento temporal no processo de cifra e decifra.

O estudo foi feito com diversos tamanhos de blocos (512, 1024 e 2048 *bytes*), com o tamanho máximo de chave para cada algoritmo de cifra. Os dados a cifrar pelo programa de análise temporal de cifras são os presentes na Figura 12 (em formato JSON). Estes dados correspondem a uma réplica dos campos enviados pela loja para o *Optipricer*, de modo a este criar um cupão.

```

1 {
2   "min": 10,
3   "max": 30,
4   "location": "Aveiro, Portugal",
5   "text": "",
6   "discount_render": 1,
7   "discount_offset": 3600,
8   "product_id": 1,
9   "name": "teste",
10  "link": "http://teste.te/teste/teste/2",
11  "description": "Lorem ipsum dolor sit amet, cu quo dicant iriure iracundia,
12  taffert copiosae abhorreant vim in, possim tritani disputationi at sea.
13  Antiopam omittantur ut nec.",
14  "image_url": "http://teste.te/teste/teste/2",
15  "price": 50,
16  "product_brand": "blabla",
17  "product_barcode": "123456789",
18  "categories": ["a", "b", "c"]
19 }

```

Figura 12 - Dados a serem cifrados pelo algoritmo de benchmarking de cifras

Estes dados são replicados até atingirem o tamanho do bloco especificado, utilizando um *padding*.

Para medições de tempo é utilizada a função *microtime* do PHP. A máquina utilizada para estes testes opera com um CPU Intel Core i5-3317U CPU 1.70GHz, com 6GB de RAM, no sistema operativo *Ubuntu* 14.10. Este programa corre em PHP, sobre consola, utilizando a biblioteca *MCrypt*.

O gráfico seguinte apresenta os resultados obtidos no processo completo (cifra e decifra), para diversos algoritmos combinados com os diferentes modos de cifra por bloco, para três tamanhos distintos de blocos. Geralmente, no eixo dos *xx* dos gráficos está representado o tamanho do bloco a cifrar (em *bytes*) e no eixo dos *yy* o tempo médio de cada cifra / decifra, em milissegundos. As barras representam os algoritmos de cifra combinados com o modo de cifra (por bloco ou cifra contínua). Os tempos apresentados nos gráficos são médios, ou seja, para cada processo, são efetuadas 10000 cifras ou decifras e depois é calculada a média por cifra ou decifra.

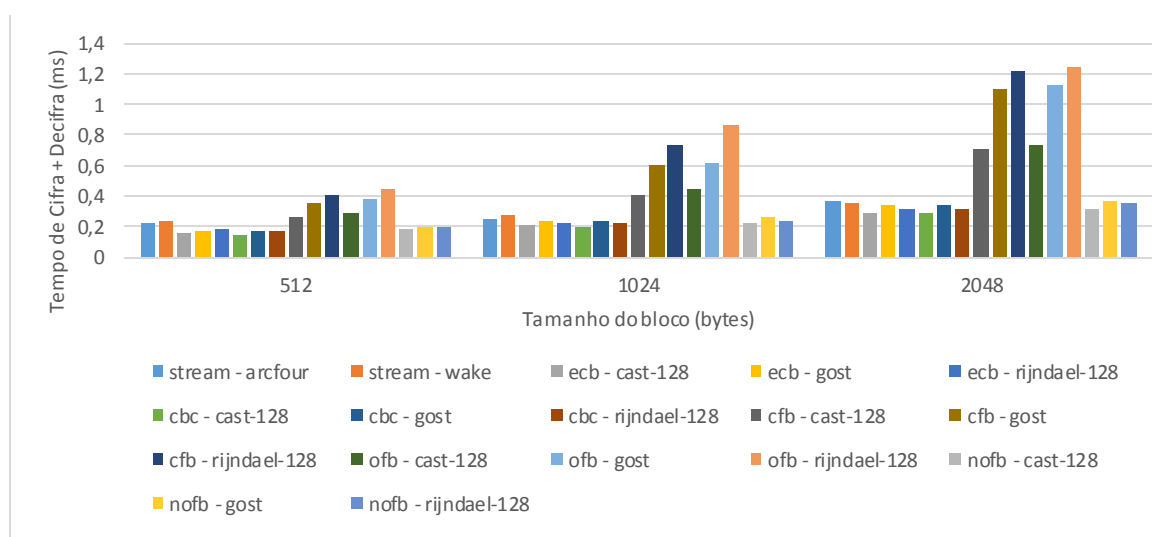


Figura 13 – Gráfico com os tempos de cifra + decifra para os três melhores algoritmos de cifra por bloco, em todos os modos de cifra por bloco e algoritmos de cifra contínua VS tamanho do bloco a cifrar

Com base na Figura 13, podemos concluir que o modo CBC é o mais rápido na cifra por bloco e que, geralmente, os algoritmos de cifra por bloco são mais rápidos que os algoritmos de cifra contínua. O modo de cifra por bloco ECB também é igualmente rápido mas não deve ser utilizado uma vez que reproduz padrões. Analisando os algoritmos mais rápidos, no geral, os algoritmos de cifra por bloco no modo CBC não ultrapassam os 0,2ms para blocos de 512 bytes, nos restantes modos de cifra por bloco, os tempos disparam, sendo que, por exemplo, o algoritmo Rijndael-128 no modo CFB ocupa cerca de 0,4ms para o processo completo, para o mesmo tamanho de bloco. O modo OFB também é igualmente mais lento, sendo que para o mesmo algoritmo e para o mesmo tamanho de bloco (512 bytes), o tempo que ocupa no conjunto de operações (cifra e decifra) ultrapassa ligeiramente os 0,4ms. Para o caso do modo OFB com n bits (NOFB), para os mesmos critérios, os tempos do processo dos diferentes algoritmos aproximam-se dos 0,2ms, como é o caso do modo de cifra por bloco CBC. Mesmo assim, o modo CBC é o mais rápido no processo conjunto de cifra e decifra. Diferenciando o tamanho do bloco, os algoritmos de cifra por bloco no modo CBC pouco se deixam afetar, visto que, por exemplo para 2048 bytes, o tempo de resposta do algoritmo Rijndael-128 ronda os 0,3ms. Já no modo CFB, os tempos disparam, para os mesmos critérios analisados anteriormente, ultrapassando os 1,2ms (cerca de 1ms de diferença). Para o modo de cifra por bloco OFB, os tempos de resposta são semelhantes aos do modo CFB. Já no modo de cifra por bloco OFB com n bits (NOFB), para blocos de 2048 bytes, o tempo de resposta é semelhante ao modo CBC, apresentando-se com uma boa performance: o algoritmo Rijndael-128 demora quase 0,4ms para o método conjunto de cifra e decifra. Em suma, os modos ECB, CBC e NOFB são os modos mais rápidos no processo de cifra e decifra de mensagens. Contudo, o modo ECB não deve ser utilizado pelo que já referimos anteriormente, introduz padrões, remetendo-nos para o modo CBC que é o mais utilizado na cifra de mensagens.

De seguida, faremos uma análise dos diferentes algoritmos de cifra por bloco no modo CBC. Nesta análise iremos apresentar gráficos para os métodos cifra, decifra e conjunto (cifra e decifra).

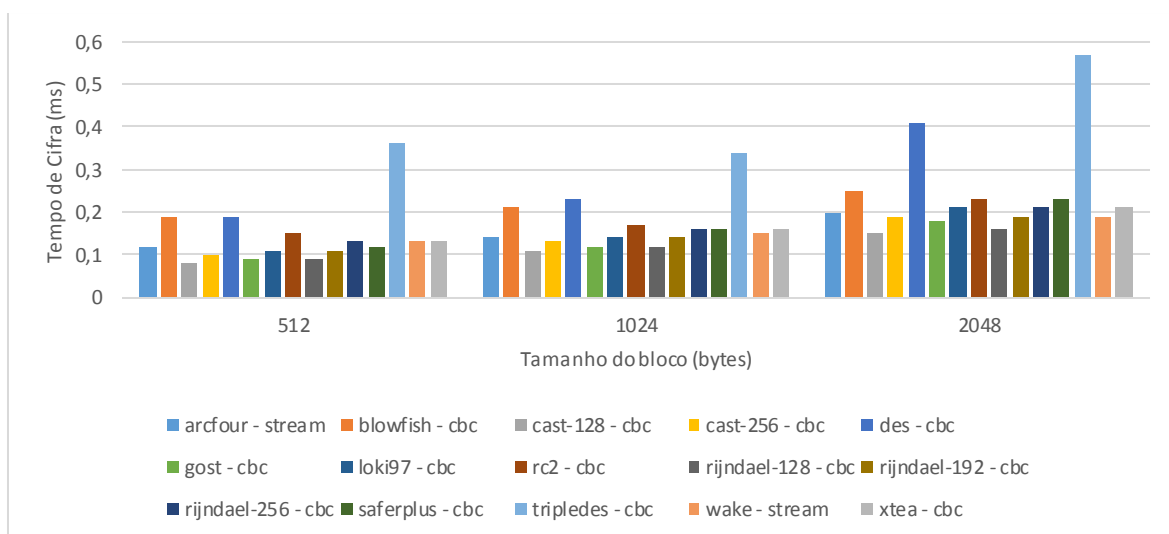


Figura 14 – Gráfico com o tempo de cifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco a cifrar

Observando a Figura 14, neste modo CBC, podemos verificar que os algoritmos mais rápidos no processo de cifra são (1) CAST-128, (2) Rijndael-128 e (3) Gost. Neste modo, os algoritmos de cifra por bloco são bem mais rápidos que os algoritmos de cifra contínua, para a técnica de cifra, em que apenas alguns algoritmos são mais lentos do que o modo de cifra contínua, como é o caso do DES ou do 3DES. O mesmo não se verifica para outros modos de cifra por bloco, onde os algoritmos de cifra contínua são mais rápidos para o processo de cifra. Esta análise encontra-se em anexo.

Analisando o gráfico sobre os vários algoritmos de cifra por bloco, para o processo de cifra, podemos concluir que o algoritmo CAST-128 é o mais rápido, demorando menos de 0,1ms para blocos de 512 bytes e cerca de 0,15ms para blocos de 2048 bytes para o processo de cifra. Para este algoritmo, o tamanho do bloco a cifrar (mensagem) pouco influencia os tempos de resposta. O mesmo se verifica para o algoritmo Rijndael-128, apresentando tempos semelhantes para o processo de cifra, relativamente ao algoritmo CAST-128. Já algoritmos como o DES ou o 3DES apresentam resultados medíocres já que o tamanho do bloco influencia negativamente o tempo de cifra. Por exemplo para o algoritmo DES, o tempo de cifra chega mesmo a duplicar, passando de cerca de 0,2ms para aproximadamente 0,4ms, para blocos de 512 bytes e 2048 bytes, respetivamente. No entanto, geralmente, os algoritmos de cifra por bloco utilizados hoje em dia apresentam bons tempos de cifra. Os algoritmos de cifra contínua têm também uma boa performance sendo que o Arcfour demora pouco mais de 0,1ms a cifrar um bloco de 512 bytes, e quase 0,2ms a cifrar um bloco de 2048 bytes.

Seguidamente iremos apresentar o gráfico dos tempos obtidos no processo de decifra, utilizando o mesmo modo de cifra por bloco (CBC).

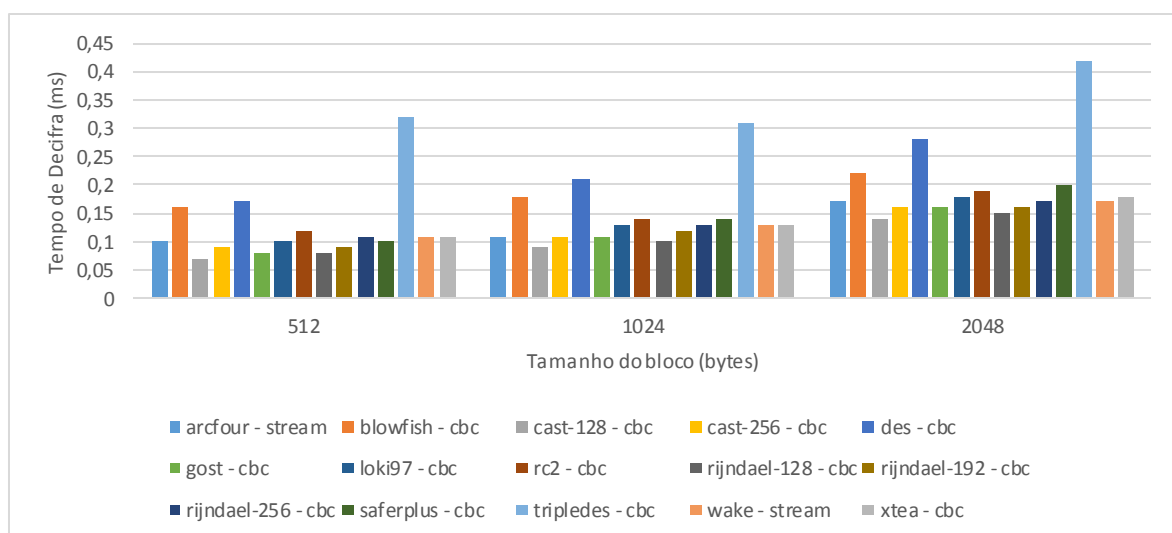


Figura 15 – Gráfico com o tempo de decifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco que foi cifrado

Pela análise do gráfico presente na Figura 15, apuramos que neste modo CBC os algoritmos mais rápidos no processo de decifra são: (1) CAST-128, (2) Rijndael-128 e (3) Gost. Estes algoritmos são muito velozes a decifrar sendo que o algoritmo mais rápido é o CAST-128 que realiza uma decifra de uma mensagem de 512 bytes em pouco mais de 0,05ms, e mensagens de 2048 bytes o mesmo algoritmo consegue decifrá-las em menos de 0,15ms, variando o tempo de decifra em apenas cerca de 0,1ms com o aumento significativo do tamanho da mensagem. Tal não se verifica para outros algoritmos, como é o caso do DES ou 3DES, que, como averiguámos no processo de cifra, são igualmente lentos neste processo. Por exemplo, o algoritmo 3DES, no geral, a sua performance é deplorável em comparação com outros algoritmos. Para uma mensagem de 512 bytes, este algoritmo necessita de mais de 0,3ms para a decifrar, e para mensagens de 2048 bytes o processo de decifra ocupa 0,4ms. O algoritmo Blowfish é igualmente lento, mas mais constante visto que o aumento significativo do tamanho do bloco pouco influencia o tempo de resposta: considerando blocos de 512 bytes, este algoritmo consegue decifrá-los em cerca de 0,15ms, e blocos de 2048 bytes o tempo apenas aumenta em pouco mais de 0,05ms, correndo o processo em aproximadamente 0,2ms. Sendo o algoritmo Rijndael-128 um dos mais utilizados, importa fazer a sua análise.

Este algoritmo é capaz de decifrar mensagens de 512 bytes em menos de 0,1ms, e em mensagens de 2048 bytes, a decifra das mesmas demora cerca de 0,15ms. Neste caso, com o aumento excessivo do bloco a decifrar (de 512 para 2048 bytes), os tempos do algoritmo Rijndael-128 apenas aumentam em cerca de 0,05ms. Em suma, este algoritmo é extremamente rápido no processo de decifra, tendo apenas à sua frente o CAST-128. Fazendo uma análise aos algoritmos de cifra contínua, estes no geral apresentam boa performance, visto que os seus tempos de resposta não diferem muito dos algoritmos mais rápidos. Analisando o Arcfour (Alleged RC4), podemos dizer que este algoritmo consegue decifrar mensagens de 512 bytes em perto de 0,1ms, e em mensagens de 2048 bytes o tempo de decifra pouco ultrapassa os 0,15ms.

Na Figura 16 apresentamos os resultados da soma dos valores dos gráficos anteriores, ou seja, o tempo de cifra e decifra de diversos algoritmos no modo de cifra por bloco CBC, com diferentes tamanhos de bloco.

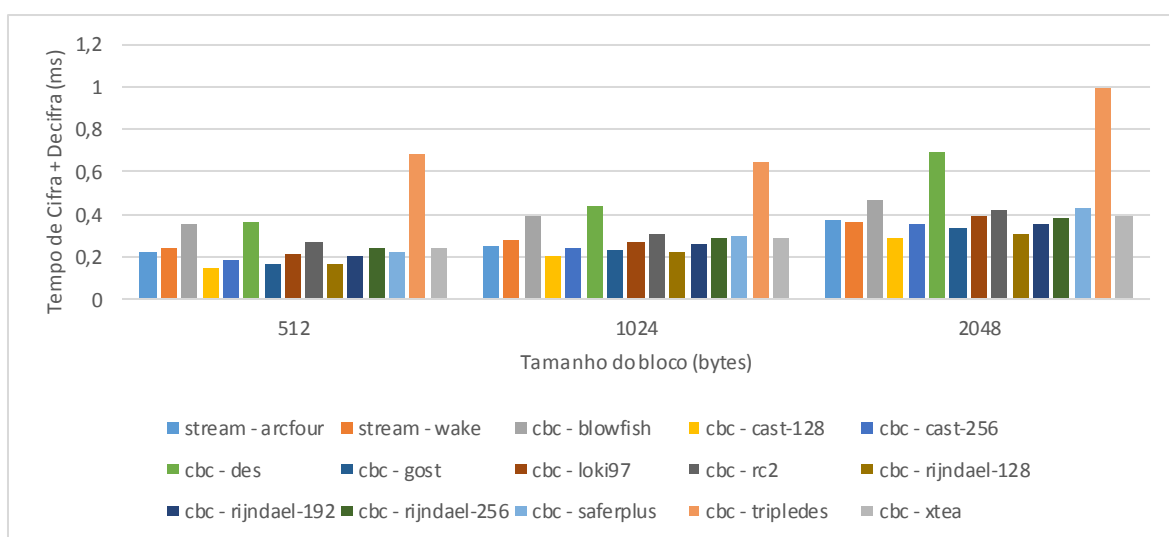


Figura 16 – Gráfico com o tempo de cifra + decifra de diversos algoritmos com o modo de cifra por bloco CBC e cifras contínuas VS tamanho do bloco a cifrar

Podemos concluir que os algoritmos de cifra mais rápidos são: (1) CAST-128, (2) Rijndael-128 e (3) Gost. Estes resultados já seriam de esperar, tendo em conta as análises efetuadas anteriormente. O algoritmo mais rápido, o CAST-128, para o processo completo consegue ser muito rápido, sendo que, para blocos de 512 bytes, este algoritmo demora menos de 0,2ms para cifrar e decifrar mensagens. Se variarmos o tamanho do bloco, ele pouco influencia no tempo de resposta, sendo que este algoritmo demora cerca de 0,3ms para o processo completo. Se analisarmos o algoritmo Rijndael-128, os tempos são bastante próximos do algoritmo CAST-128, tanto para mensagens de 512 bytes como para mensagens de 2048 bytes. Como examinado nos gráficos anteriores, confirma-se que os algoritmos mais lentos são o Blowfish, DES e 3DES, sendo que este último chega a demorar perto de 1ms para o processo completo em blocos de 2048 bytes. Para 512 bytes, os tempos não são muito favoráveis neste algoritmo, estando um pouco abaixo dos 0,7ms. Já relativamente aos algoritmos de cifra contínuos, é expectável que tenham um bom rendimento, tal como percebemos pela análise anterior, sendo que o algoritmo Arcfour, para o processo completo, atuando sobre blocos de 512 bytes consome cerca de 0,2ms e para blocos de 2048 bytes, o tempo do processo é quase o dobro, não aumentando significativamente (quase 0,4ms). Os gráficos que conjugam outros modos de cifra por bloco podem ser encontrados no apêndice B.

Embora o algoritmo CAST-128 seja mais rápido que o Rijndael-128, este último é mais seguro visto suportar chaves até 256 bits, ao contrário do CAST-128 que apenas suporta chaves até 128 bits. Podemos concluir que a rapidez do algoritmo CAST-128 deve-se ao facto de este utilizar chaves mais curtas, evidenciando a qualidade do Rijndael-128, que mesmo utilizando chaves de 256 bits, consegue diferir muito pouco do CAST-128, no processo completo, como observámos anteriormente. Além disso, o Rijndael-128 é um algoritmo reconhecido pelo NIST (*National Institute of Standards and Technology*), sendo este utilizado pelo governo dos Estados Unidos. Pesando estes factos todos, utilizaremos o algoritmo Rijndael-128 no nosso projeto, dado a sua segurança, o seu reconhecimento e a sua performance. O modo de cifra que iremos utilizar irá ser o modo CBC, pois é o modo mais rápido, um dos mais seguros e também o mais utilizado.

3.2 Cifras assimétricas

As cifras assimétricas, também designadas cifras de chave pública, usam um par de chaves distintas mas relacionadas: uma pública utilizada para cifrar e uma privada utilizada para decifrar. Algumas cifras ainda permitem a operação inversa, ou seja, cifrar com a chave privada e decifrar com a chave pública, o que não tem interesse para esconder mas sim para garantir a autenticação dos dados. As chaves assimétricas podem ser personalizadas, ou seja, associadas a pessoas, serviços ou servidores. A componente privada só deve ser conhecida e usada pela entidade a que está associada, ou seja, não deve ser partilhada. Já a componente pública deve ser ampla e publicamente divulgada para poder ser usada por qualquer entidade [31].

Em termos de vantagens, este tipo de cifras exige menos chaves para efetuar interações seguras, pois permite uma relação de muitos para um. Num universo de N interlocutores, só é preciso utilizar N chaves diferentes.

A sua principal desvantagem é a eficiência: são muito pouco eficientes porque se baseiam em operações matemáticas complexas. Deste modo, a utilização deste tipo de chaves requer um uso computacional exigente, o que pode influenciar os tempos de resposta de um determinado servidor [31].

Um modo de utilização de cifras assimétricas são as **Assinaturas Digitais**. O objetivo destas é validar a autoria de uma mensagem perante terceiros. Uma mensagem assinada digitalmente deverá ser associada a uma e só uma entidade e a assinatura deverá poder ser validada universalmente. Deste modo, a criptografia assimétrica é a que melhor se adequa a este fim visto que o par de chaves tem um caráter pessoal [31].

De uma forma simples, quem pretende assinar a mensagem cifra o conteúdo da mensagem com a sua chave privada, de modo a que o universo de interlocutores possa decifrar a mesma com a chave pública do autor da mensagem. O criptograma correspondente não serve para esconder a mensagem mas sim para a validar. As assinaturas digitais são sempre diferentes e únicas consoante o documento que autenticam. Devido à grande complexidade das funções de cifra assimétrica, é efetuada uma síntese do documento antes de ser cifrado (por exemplo utilizando o algoritmo de *hashing* SHA-1). Deste modo, obtém-se assinaturas mais reduzidas (e não do tamanho do documento, provocando maior eficiência nas funções de assinatura e validação) [31].

Existem diversos algoritmos de cifra assimétrica, para assinaturas digitais, destacando-se o RSA e o DSA. O tamanho mínimo de chave a ser utilizada pelos algoritmos de assinatura deve ser de 1024 bits, para aplicações comerciais [54].

O RSA foi apresentado em 1977, devendo-se o seu nome às iniciais dos seus criadores - Rivest, Shamir e Adleman. Este algoritmo baseia-se na complexidade de factorização e cálculo de logaritmos modulares (de grandes números) [31].

Em 1991, o NIST apresentou uma proposta de assinaturas digitais (**Digital Signature Standard, DSS**). O DSS utiliza o **Digital Signature Algorithm (DSA)** derivado do algoritmo de assinatura ElGamal. Este algoritmo é bastante mais lento do que o **RSA**, nomeadamente na verificação de assinaturas. No entanto, podem-se usar valores pré-computados para acelerar a geração de assinaturas [31].

Nas assinaturas digitais são utilizadas funções de síntese (*hashing*), de modo a sintetizar a assinatura e aumentar o desempenho dos algoritmos de assinatura [31].

As funções de síntese não são propriamente funções criptográficas, mas são úteis para complementá-las. As funções de síntese produzem valores de dimensão constante a partir de textos de funções variáveis. O modo de funcionamento destas funções consiste na aplicação sucessiva de uma função de compressão, que trata e produz dados de dimensão constante, convenientemente alinhados à dimensão do bloco de entrada da função de compressão. A função de compressão toma dois valores, um com uma síntese prévia e outro com o bloco a processar, e produz uma nova síntese. Em termos formais, uma função de síntese deve obedecer a três regras: resistência à descoberta de um texto original, resistência à descoberta de um segundo texto original e resistência à colisão [31].

As funções de síntese mais utilizadas são o MD5, o SHA (Secure Hash Algorithm) e o RIPEMD. Em 2004, um grupo de investigadores, descobriu um método expeditivo de cálculo de colisões em diversas funções,

nomeadamente MD4, MD5, HAVAL-128 e RIPEMD [60]. Visto existirem colisões nestas funções de síntese, estas não devem ser utilizadas.

Deste modo, a função de síntese mais recomendada, neste momento e face às vulnerabilidades que se conhecem, é a SHA-1. Outras funções derivadas do SHA, tais como SHA-224 ou SHA-256, também devem ser utilizadas.

Seguidamente referiremos um estudo que efetuámos sobre o comportamento temporal dos diversos algoritmos de assinatura conjugados com os algoritmos de síntese mais conhecidos. Para estes, será apresentado um gráfico, onde varia o tamanho do bloco a assinar e o tamanho da chave (1024 e 2048 bits). Para cada tamanho de chave serão apresentados gráficos de assinatura, de verificação de assinatura e do processo completo.

Os dados a serem assinados representam uma possível réplica de uma assinatura sobre um cupão emitido pelo *Optipricer*;

```
"qwedses2314s123s:aveiro123esd124wesdaf:20:1245212312:70"
```

Estes dados são replicados até ao tamanho do bloco, sendo aplicado um *padding*.

Para medições de tempo é utilizada a função *microtime* do PHP. A máquina utilizada para estes testes opera com um CPU Intel Core i5-3317U CPU 1.70GHz, com 6GB de RAM, no sistema operativo *Ubuntu* 14.10. Este programa corre em PHP, sobre consola, utilizando a biblioteca *OpenSSL*.

A Figura 17 apresenta os resultados obtidos no processo de assinatura, de verificação e completo (assinatura e verificação), para os dois algoritmos de assinaturas (RSA e DSA) combinados com as diferentes funções de síntese, para três tamanhos distintos de blocos. Geralmente, no eixo dos *xx* dos gráficos está representado o tamanho do bloco a assinar (em bytes) e no eixo dos *yy* o tempo médio de cada assinatura / verificação, em milissegundos. As barras representam os algoritmos de assinatura combinados com o algoritmo de síntese.

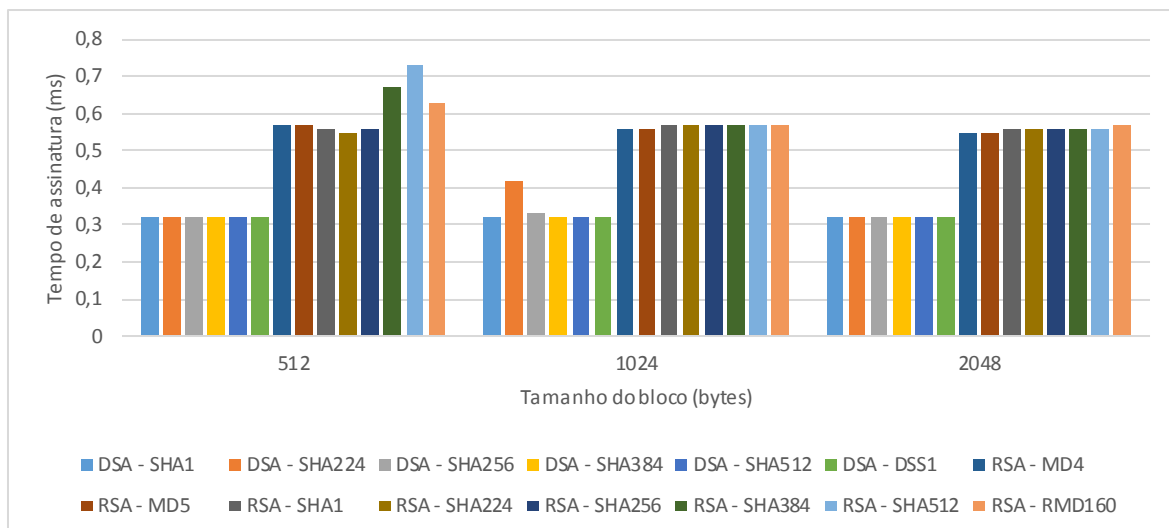


Figura 17 - Gráfico com os tempos de assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar com chaves de 1024 bits

As primeiras 6 barras representam as assinaturas com o algoritmo DSA, e as restantes com o algoritmo RSA. A variável introduzida é a função de síntese utilizada.

Com base no gráfico da Figura 17, podemos concluir que, no processo de assinatura, o DSA é bem mais rápido que o RSA, sendo que o RSA demora quase o dobro do tempo a assinar. A função de síntese escolhida pouco influencia nos tempos de resposta. As maiores oscilações dos tempos de assinatura com os diferentes algoritmos de síntese deve-se a variações de processamento do computador utilizado. Variar os blocos, nada influenciará nos tempos de assinatura, sendo que o tempo é praticamente igual para assinatura de blocos de 512 bytes e de 2048 bytes. Esta condição verifica-se porque a geração da assinatura é feita sobre uma síntese do conteúdo, ou seja, a aplicação da assinatura digital é sempre efetuada sobre um bloco de tamanho idêntico (output do algoritmo de síntese utilizado). Desta forma, o algoritmo DSA demora cerca de 0,3ms a assinar cada mensagem (de tamanho de bloco e algoritmo de síntese diferentes) enquanto o algoritmo RSA ocupa cerca de 0,5ms para proceder à assinatura de cada mensagem.

De seguida, iremos verificar como se comporta o processo de assinatura para chaves com o dobro do tamanho (2048 bits).

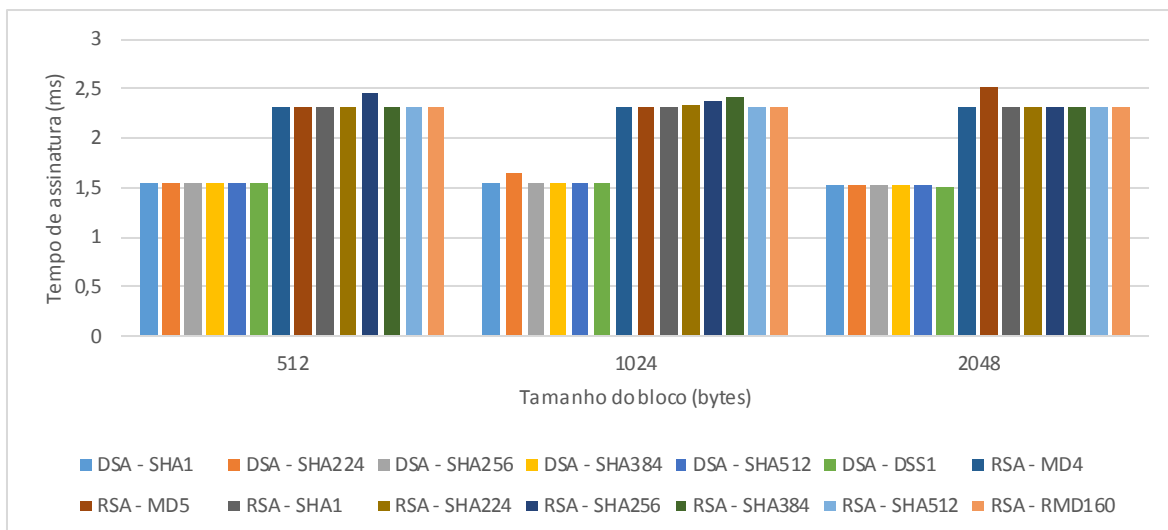


Figura 18 - Gráfico com os tempos de assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar com chaves de 2048 bits

Com chaves de 2048 bits, o algoritmo DSA continua a ser mais rápido a gerar a assinatura do que o RSA (Figura 18). No entanto, os tempos de assinatura utilizando chaves de 2048 bits são catastróficos: aumentam exageradamente, de aproximadamente 0,3ms para 1,5ms, no caso do DSA, e de aproximadamente 0,6ms para 2,3ms no caso do RSA. Como verificamos no gráfico em cima (Figura 18/Figura 17), a variação do tamanho do bloco e a variação do algoritmo de síntese nada influenciam os tempos de assinatura. O mesmo se constata com este gráfico, pois variando o tamanho do bloco ou o algoritmo de síntese os tempos de assinatura mantêm-se constantes. Deste modo, podemos concluir que apenas o tamanho da chave e o algoritmo de assinatura utilizado (DSA ou RSA) influenciam o tempo de assinatura de uma mensagem.

A Figura 19 apresenta o comportamento dos algoritmos de verificação de assinaturas para diferentes tamanhos de blocos para chaves de 1024 bits.

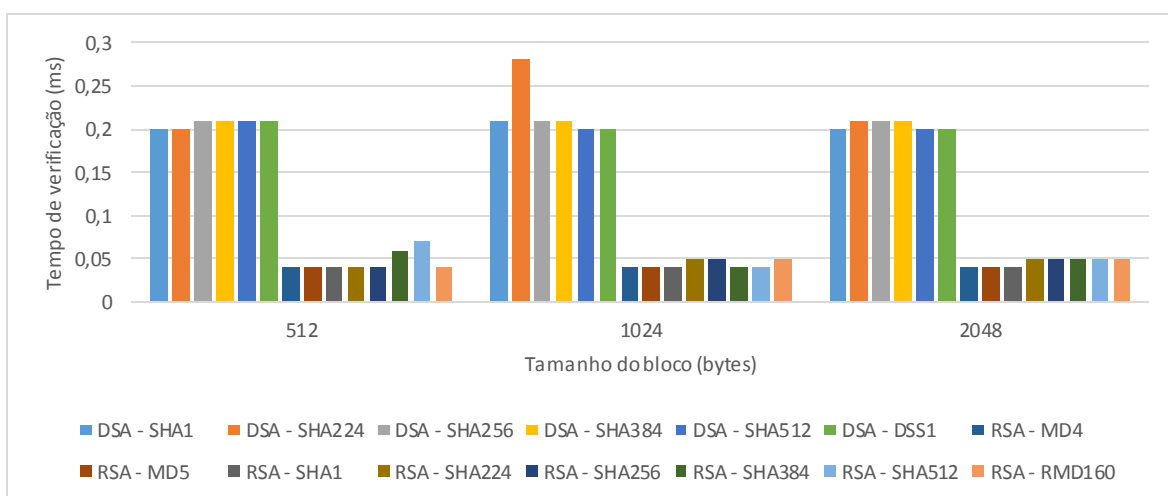


Figura 19 - Gráfico com os tempos de verificação da assinatura para os diversos algoritmos de assinatura e síntese VS tamanho do bloco assinado, com chaves de 1024 bits

Como podemos analisar pelo gráfico presente na Figura 19, o RSA é rapidíssimo a verificar a assinatura, demorando cerca de 0,05ms. Já o DSA demora cerca de 0,2ms na verificação das mesmas. Como analisado no processo anterior (assinatura de dados), o algoritmo de síntese e o tamanho do pacote a assinar pouco ou nada influenciam os tempos de assinatura. Para a verificação de assinaturas, confirma-se o mesmo: pelo gráfico da Figura 19, apuramos que os tempos de verificação pouco variam quando se altera o algoritmo de síntese utilizado e quando se altera o tamanho do bloco. As maiores variações são provocadas pela unidade de processamento do computador utilizado, podendo ser causado por outros processos concorrentes na altura do teste. Deste gráfico concluímos que o algoritmo mais rápido de verificação de assinaturas para chaves de 1024 bits é o RSA, diferenciando em cerca 0,15ms do algoritmo DSA. Concluimos ainda que ao contrário do processo de assinatura, o algoritmo mais rápido na verificação é o RSA, podendo provocar um empate no processo completo.

A Figura 20 apresenta os tempos de verificação de assinatura para diversos algoritmos de síntese e de assinatura conjugados, em diversos blocos assinados (512, 1024 e 2048 bytes) utilizando chaves de 2048 bits:

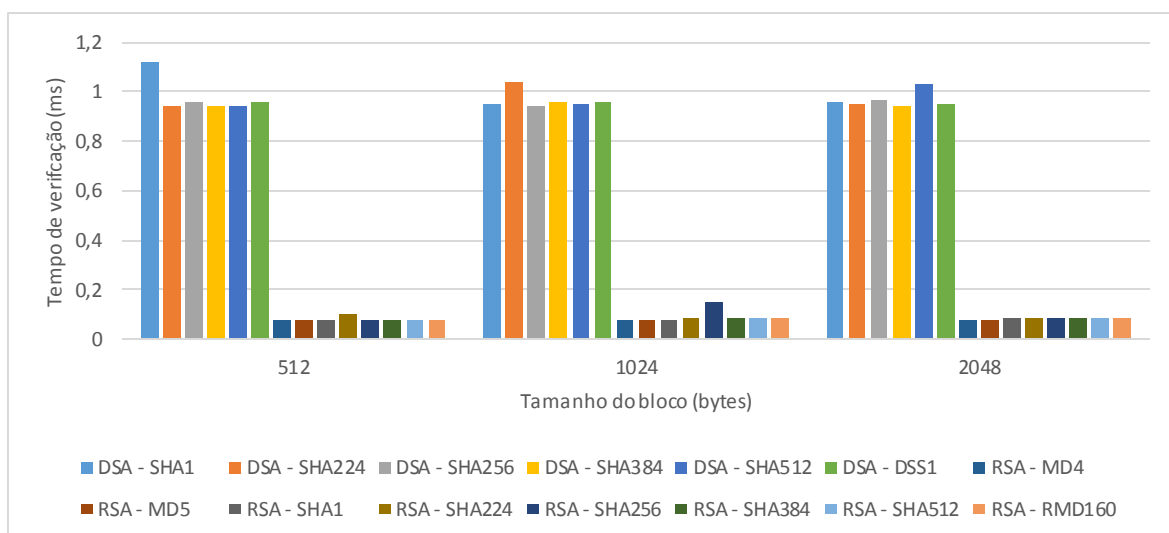


Figura 20 - Tempos de verificação da assinatura com os diversos algoritmos de assinatura e síntese VS tamanho do bloco assinado, com chaves de 2048 bits

Mesmo para chaves de 2048 bits, pelo gráfico da Figura 20, constatamos que o algoritmo RSA continua a ser bem mais rápido a verificar a assinatura, em relação ao algoritmo DSA, demorando cerca de 0,07ms. Já no algoritmo DSA o tempo de verificação de assinaturas aproxima-se dos 1ms. Para este processo, concluímos que a escolha do tamanho da chave (1024 ou 2048 bits), no caso do algoritmo RSA, pouco influencia a performance do mesmo. Já para o algoritmo DSA, a verificação de assinaturas dispara para quase 1ms, quando utilizadas chaves de 2048 bits.

Por último, a Figura 21 apresenta os resultados do processo completo, para diferentes tamanhos de bloco com chaves de 1024 bits.

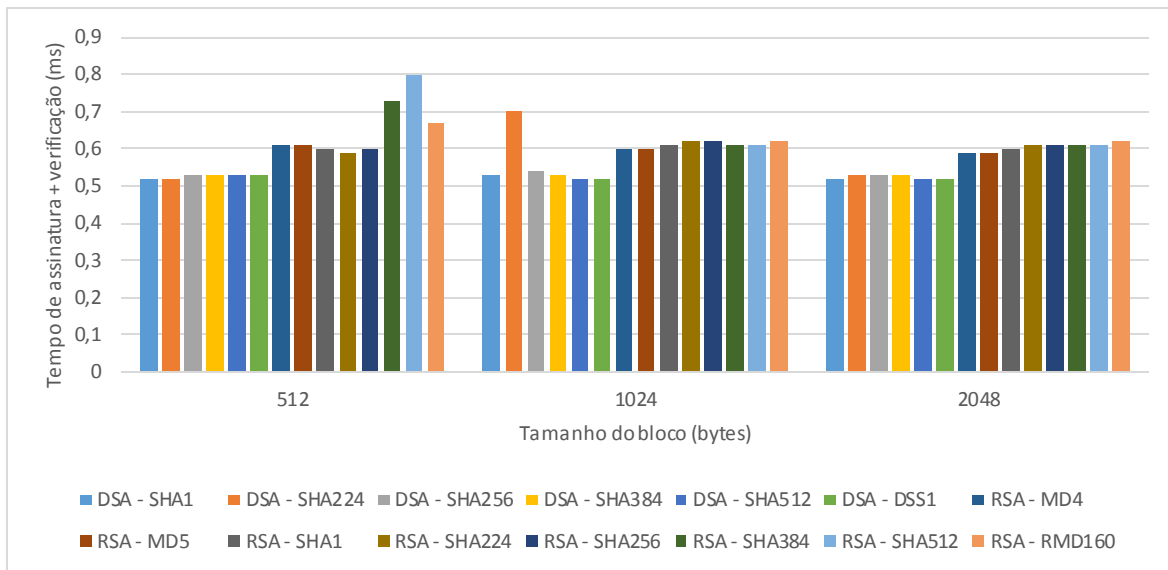


Figura 21 – Gráfico com tempos de assinatura + verificação para os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar, com chaves de 1024 bits

Pelo gráfico presente na Figura 21, compreendemos que mesmo com a rapidez do RSA a verificar a assinatura, o DSA continua a ser o algoritmo mais rápido para o processo completo de assinaturas. Como verificamos nos processos individuais, a escolha do algoritmo de síntese e o tamanho do bloco a assinar / assinado pouco ou nada influenciam os tempos do processo. Neste processo, para chaves de 1024 bits, os tempos totais de assinatura e verificação pouco se diferenciam para os dois algoritmos testados. O algoritmo DSA consegue ser um pouco mais rápido que o RSA, ocupando cerca de 0,5ms, já o RSA demora cerca de 0,6ms para realizar o processo completo de assinatura e verificação. Posto isto, podemos concluir que, para chaves de 1024 bits, o algoritmo DSA, em relação ao RSA, é muito mais rápido na geração de assinaturas, mas mais lento a verificá-las, o que resulta em tempos aproximados no processo completo (assinatura e verificação); a escolha do algoritmo de síntese pouco ou nada influencia no processo de assinatura e verificação, bem como o tamanho do bloco a assinar; mesmo com a rapidez do algoritmo RSA a verificar assinaturas, o algoritmo DSA continua a ser o mais rápido no processo completo, para chaves de 1024 bits.

A Figura 22 apresenta o comportamento dos algoritmos de assinatura RSA e DSA conjugados com diferentes algoritmos de síntese, no processo de assinatura e verificação para diferentes tamanhos de blocos utilizando chaves de 2048 bits:

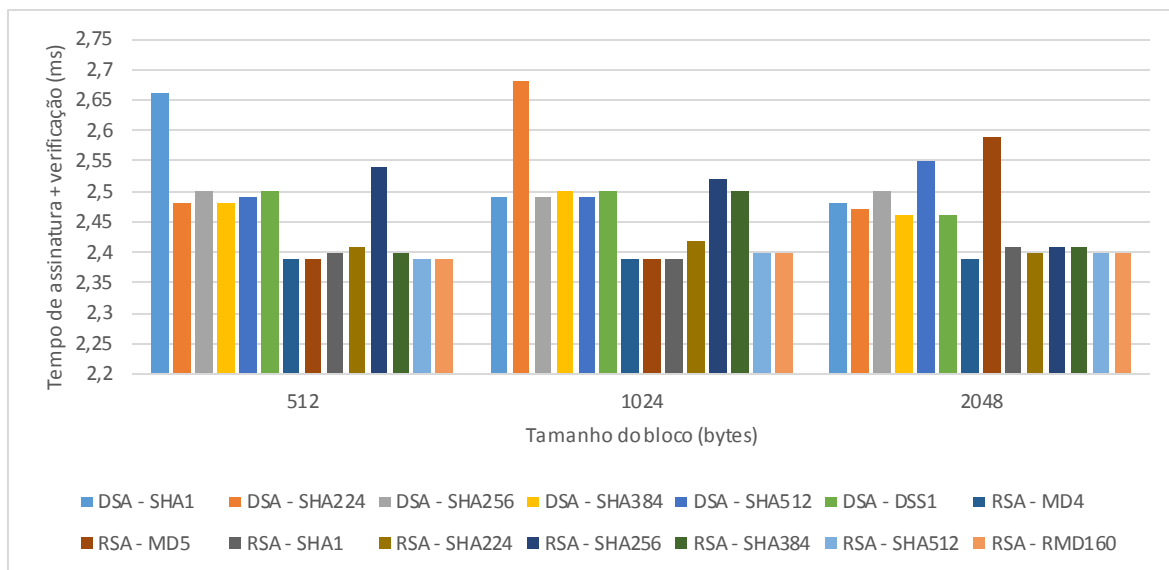


Figura 22 – Gráfico com tempos de assinatura + verificação para os diversos algoritmos de assinatura e síntese VS tamanho do bloco a assinar, com chaves de 2048 bits

Analisando o gráfico da Figura 22, podemos concluir que, se utilizarmos chaves de 2048 bits, o algoritmo RSA é mais rápido no processo completo. No entanto, com a utilização de chaves de 2048 bits, o processo completo chega a demorar cerca de 2,4ms. Prevemos que, com carga num servidor, utilizando chaves de 2048 bits, o processo completo seja bastante dispendioso, temporalmente. A diferença de performance dos dois algoritmos no processo completo é mínima (cerca de 0,05ms). Estas semelhanças de performance deve-se ao facto do algoritmo RSA ser bem mais rápido a verificar assinaturas do que o DSA (0,07ms e 1ms, respetivamente), mas de igual forma, o algoritmo DSA consegue ser bem mais rentável no processo de assinatura do que o RSA (1,5ms e 2,3ms, respetivamente).

Com base nos estudos prévios por nós efetuados e acima descritos, iremos utilizar o algoritmo RSA para geração e verificação de assinaturas, uma vez que este algoritmo é rapidíssimo a verificar assinaturas, tornando a navegação na loja mais rápida, nomeadamente nos processos de adição de um produto ao carrinho com desconto e checkout dessa mesma compra. Quanto ao algoritmo de síntese, uma vez que este pouco ou nada influencia, iremos adotar o SHA256, visto este mitigar o efeito de colisões na síntese de dados. Quanto ao tamanho da chave privada, iremos adotar o tamanho de 1024 bits, uma vez que o desempenho das chaves de 2048 bits é medíocre. Com um mecanismo de geração de novas chaves, chaves de 1024 bits tornam-se suficientes para garantir a autenticidade dos dados.

Concluído o estudo sobre criptografia iremos avançar para o próximo capítulo onde faremos uma análise da arquitetura do nosso projeto, os problemas que podem surgir em cada atividade e sugestões de solução para cada problema.

4. Casos de uso e interação

Nesta secção iremos primeiramente abordar o problema, referenciando os casos de utilização do *Optipricer*, e os respetivos objetivos de cada entidade. De seguida, avançaremos para a enunciação de cada atividade, quais as mensagens que circulam entre cada entidade e quais as falhas de segurança presentes em cada uma das comunicações. Por fim, enunciaremos soluções para cada uma das falhas de segurança detetadas anteriormente.

Como enunciado na introdução, o *Optipricer* é uma aplicação *Web* que disponibiliza descontos que dependem do perfil social do utilizador, através da sua API. Com efeito, sempre que o utilizador quiser obter um desconto extra sobre um produto numa loja, deve proceder à partilha do mesmo na rede social, a partir da aplicação do *Optipricer*, e de seguida o desconto é calculado de acordo com a sua influência na rede, o seu poder de compra, o seu interesse no produto entre outros critérios decisivos. Deste modo, é necessário obedecer a certos princípios de modo a que o desconto seja bem utilizado e aplicado. Devemos concentrar a aplicação em quatro fluxos principais: instalação do *plugin*, obtenção do desconto remotamente a partir do *Optipricer*, adição do produto ao carrinho com desconto e checkout do mesmo com desconto. Estes fluxos serão discutidos numa subsecção deste capítulo.

Visto que as comunicações com as redes sociais (como obtenção de um *token* de autenticação), são feitas do lado do *browser* do cliente, este é uma entidade importante que participa nos fluxos especificados.

4.1 Descrição dos casos de uso

Neste subcapítulo, iremos enunciar o funcionamento do *Optipricer*, descrevendo os seus casos de uso. Nos casos de utilização do *Optipricer* temos 3 entidades presentes: o administrador da loja que configura a extensão na loja com um intervalo de valores onde vai ser gerado o desconto, o cliente que obtém um desconto na loja a partir do *Optipricer*, e o próprio *Optipricer* que representa um servidor remoto de obtenção de descontos. Os casos de utilização estão descritos na Figura 23. Com base nesta, podemos descrever as atividades principais de cada entidade:

- **Admin Loja:** configura as definições do *plugin*, como o *token store* (disponibilizado pelo *Optipricer*) o valor de desconto mínimo e máximo (em percentagem), a chave partilhada, o tempo de longevidade do cupão, entre outros;
- **Utilizador:** pede um desconto para um produto na loja, partilha-o numa rede social através do *Optipricer*, e obtém um desconto dinâmico dependendo do seu perfil social. O utilizador (*browser*) também é responsável por enviar dados da loja e do produto aquando da obtenção do desconto, por adicionar o produto ao carrinho e efetuar a compra;
- ***Optipricer*:** é responsável pela criação de um cupão de desconto (envio do cupão) e por gerar a partilha no *Facebook* do produto sobre o qual o utilizador pretende um desconto. Quando a compra for efetuada pelo utilizador, esta entidade deve validar o cupão de desconto.

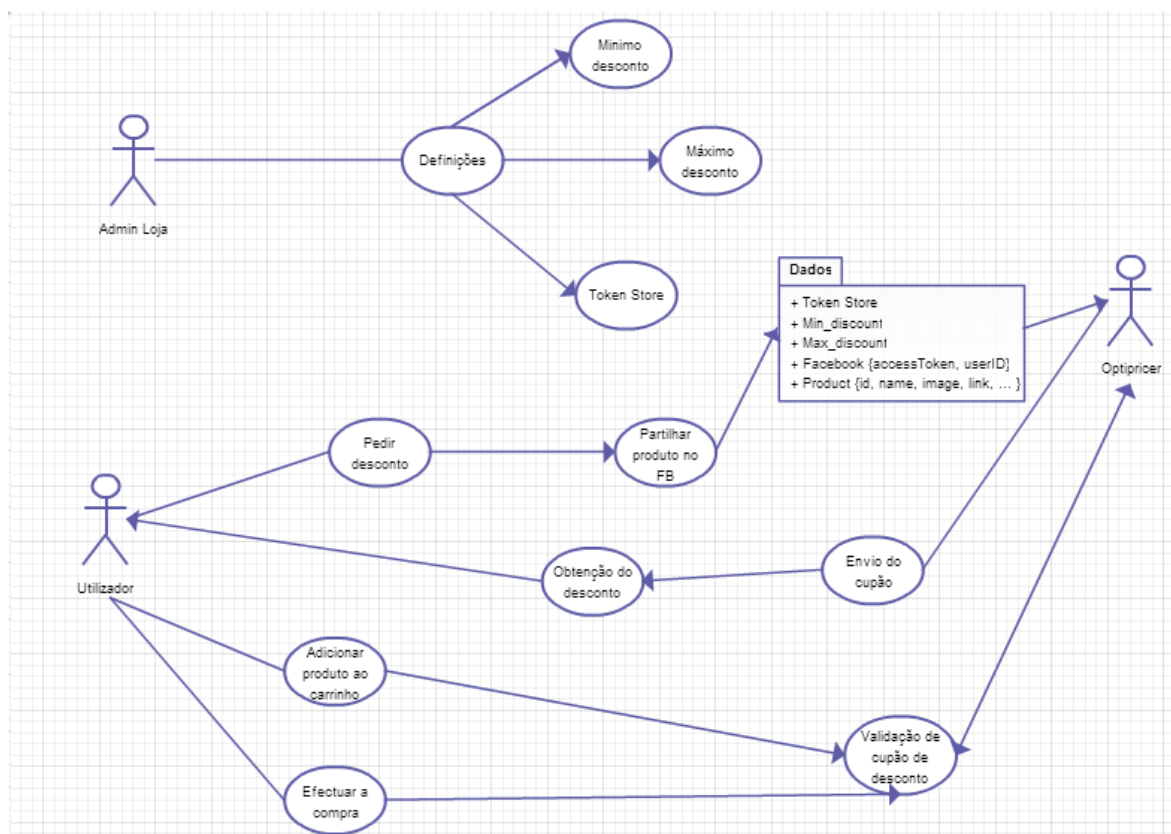


Figura 23- Diagrama de casos de utilização do Optipricer

4.2 Análise de Fluxos

Esta secção apresenta uma análise dos fluxos de informação para cada uma das atividades principais do *Optipricer*. Deste modo, a solução pode ser decomposta em 4 partes:

- Instalação e ativação do *plugin*: são geradas informações sobre a loja no servidor de descontos, e posteriormente é disponibilizado um ID à loja para que se possa identificar neste servidor;
- Obtenção do desconto de forma remota: o utilizador obtém um desconto para um certo produto numa loja a partir do *Optipricer*;
- Adição do produto com desconto ao carrinho: o produto deve ser adicionado ao carrinho com o desconto obtido;
- Checkout do produto com desconto: o desconto é validado internamente pela loja e é informado ao servidor que foi utilizado.

Seguidamente iremos referenciar cada fluxo para a resolução do problema, a informação sensível que intervém em cada comunicação, os problemas de segurança em cada comunicação e sugestões para a sua resolução. No final, iremos propor a solução a adotar para cada fluxo.

4.2.1 Fluxo de ativação

Neste processo, o cliente instala o *plugin* na loja e quer ativá-lo. Este fluxo pode ser feito manualmente ou automaticamente. O processo manual consiste no preenchimento dos campos de nome, *email* e observações, no *plugin* da loja, dados estes que serão enviados para o *Optipricer* através do consumo de um método da sua API. Seguidamente a loja será notificada pelo *Optipricer* com os dados a preencher na loja, através de um *email*.

A abordagem automática é um processo que conta com dois intervenientes: a loja e o servidor. Inicialmente o cliente instala o *plugin* na loja e depara-se com um ecrã de opções. Como o cliente não sabe quais os dados a preencher faz um pedido ao servidor. Neste pedido terá que preencher alguns dados como o seu nome, o seu *email*, o plano que quer adotar para a sua loja entre outros. Depois de preenchidos, o utilizador clica em registo. Neste passo há um pedido para o servidor de registo ou ativação do *plugin*. O servidor determina um id para a loja (*tokenStore*), entre outros dados, e envia para a loja. Estes dados são recolhidos pela loja e preenchidos automaticamente. Da mesma forma, o servidor envia um *email* ao cliente com dados para aceder ao *backoffice* do *Optipricer* para consultar estatísticas relativas aos seus clientes e aos descontos obtidos. Contudo, o *plugin* fica ativo e pronto a ser utilizado. A Figura 24 ilustra o diagrama de atividades do fluxo de ativação automática, em que existem duas comunicações entre a loja e o servidor: *activateStore()*, a loja envia os seus dados ao servidor e *response()*, o servidor (*Optipricer*) envia os dados a serem configurados na loja (de forma automática), que contém o *token* identificativo da loja, a chave partilhada entre as duas entidades, entre outros. Para o processo manual, o diagrama é idêntico, apenas difere nos dados enviados, a comunicação *response()* segue sem quaisquer dados, apenas para confirmar que o processo foi bem efetuado. Já a comunicação *activateStore()* representa um pedido de contacto em que a loja se mostra interessada no produto e envia os seus dados para mais tarde ser contactada pela administração do *Optipricer* com os dados a serem preenchidos na loja.

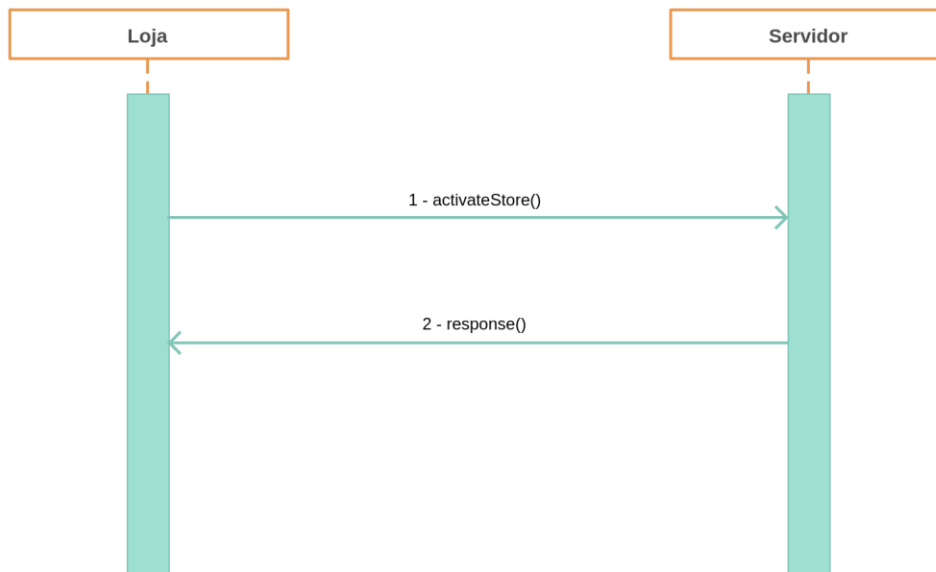


Figura 24 - Diagrama de Sequência do fluxo de ativação

Neste fluxo é possível ocorrer a manipulação dos dados tanto no pedido ao servidor como na resposta. Portanto, temos como problema o envio de dados falsos ao servidor (para criar lojas fictícias), replicação dos dados (envio de múltiplos pedidos replicados), e visibilidade dos dados (os dados do cliente podem ser sensíveis para o registo no *Optipricer*). Como é neste fluxo que existe a troca de informações entre a loja e o *Optipricer*, este último envia um id representativo da loja, entre outros dados que poderão vir a ser relevantes (como uma chave partilhada, um certificado do servidor, ...) é possível manipular os dados da resposta do *Optipricer*, de modo a enviar dados falsos para a loja e posteriormente conseguir captar outras comunicações ou utilizar o id da loja para proveito próprio (utilizar um serviço pago sem o pagar).

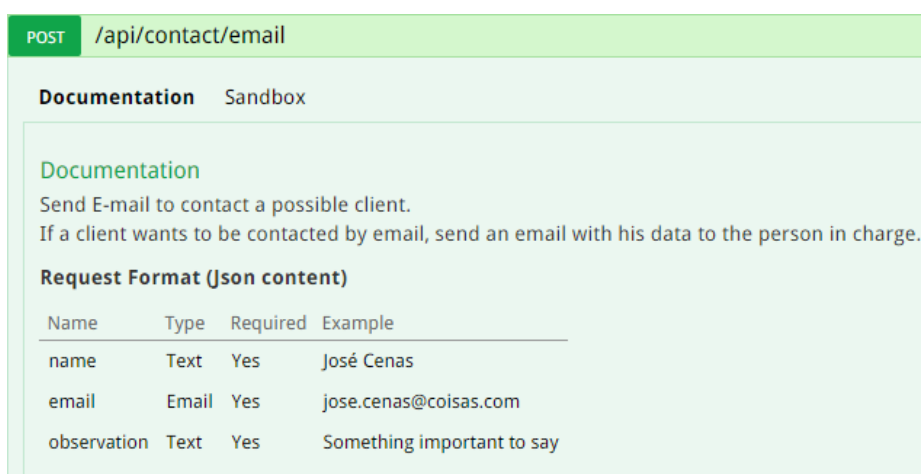
Resumidamente, os problemas de segurança que podem surgir em ambas as comunicações são: visibilidade dos dados, manipulação de dados (ataque MITM e replicação de pedidos / respostas).

Para estes problemas podemos ter as seguintes soluções: estabelecimento de uma ligação segura em HTTPS ou os dados irem cifrados (visibilidade dos dados e manipulação de dados), os dados irem assinados de forma a não serem manipulados (manipulação de dados) e colocação de um *timetamp* nas mensagens ou um campo gerado aleatoriamente, de forma a resolver a problemática da replicação de pedidos.

Para a utilização de uma solução centralizada na criação de cifras ou assinaturas, poderá ser necessário estabelecer inicialmente uma primeira comunicação, em que a loja envia um certificado de chave pública para o servidor. Na comunicação seguinte, o servidor poderá utilizar esse certificado para garantir a segurança dos dados enviados à loja. De notar que se esta comunicação for interceptada por um ataque MITM os dados enviados pelo *Optipricer* serão comprometidos, visto que o certificado poderá ser falso. Uma solução passa por incluir um certificado do *Optipricer* no *plugin* do *Woocommerce*, de modo a evitar que haja envio do certificado por parte do *Optipricer* e que este seja adulterado.

Neste fluxo optamos pela abordagem manual visto ser a mais facilitada e com mais interação entre o administrador da loja e o *Optipricer*. Na abordagem manual os problemas são os mesmos para a primeira comunicação (`activateStore()`), visto que os dados da loja que circulam são idênticos (nome, email e observações) e na resposta, visto esta ser vazia, podemos ter o problema de replicação de respostas. Para a primeira comunicação, as soluções já foram abordadas anteriormente, para a comunicação `response()`, de modo a evitar replicação de respostas, é necessário adicionar conteúdo aleatório à mensagem.

A API utilizada no processo de ativação manual é a descrita na Figura 25.



POST /api/contact/email

Documentation Sandbox

Documentation

Send E-mail to contact a possible client.
If a client wants to be contacted by email, send an email with his data to the person in charge.

Request Format (Json content)

Name	Type	Required	Example
name	Text	Yes	José Cenas
email	Email	Yes	jose.cenas@coisas.com
observation	Text	Yes	Something important to say

Figura 25 - Documentação da API de contato do *Optipricer*

Nesta API devem constar os campos “*name*”, “*email*” e “*observation*” em formato JSON. Além dos campos que devem constar no POST da mensagem, também é necessário adicionar uma “*Authorization Header*” que deve conter um *token* predefinido. Este *token* serve para o serviço não ser “bombardeado” com pedidos não autorizados. Assim, aquando da ativação do serviço, é necessário consumir esta API de forma a enviar os dados necessários para posteriormente a loja ser comunicada pelo *Optipricer*.

Por fim, propomos a seguinte arquitetura para resolução do problema neste fluxo:



Figura 26 - Proposta de arquitetura para o fluxo de ativação

Legenda:

1 – `activateStore()`, comunicação de ativação da loja no *Optipricer* (com os dados do utilizador e da loja)

2 – `response()`, normalmente sem conteúdo, apenas com código HTTP indicando que o processo foi efetuado com sucesso

As comunicações são protegidos por HTTPS

Neste fluxo, propomos a utilização do protocolo HTTPS para assegurar as comunicações entre a loja e o *Optipricer*.

A seguir, na próxima secção, iremos apresentar o fluxo mais importante e complexo do nosso projeto, o fluxo de obtenção de um desconto a partir do *Optipricer*. Nessa secção faremos uma análise ao fluxo, aos dados que circulam nas comunicações, às vulnerabilidades encontradas e iremos propor soluções sucintas para as mesmas.

4.2.2 Fluxo de obtenção de desconto

Neste fluxo o cliente quer obter um desconto extra sobre o produto. Portanto, terá que fazer login no *Facebook* via a aplicação do *Optipricer*, e contactar o servidor com esses dados para este lhe fornecer um desconto adequado para si. Nesta atividade existem 4 intervenientes: a loja, o *browser* do cliente, o *Facebook* e o *Optipricer*.

Analisando o diagrama da Figura 27, numa primeira comunicação (1- *pageView* e 2, 2a - *response*) existe uma troca de informação direta entre a loja e o *Optipricer*. Nesta é criada uma oferta para um determinado produto (caso ainda não exista). A oferta consiste na inclusão dos detalhes do produto, mapeando-o com a loja e os intervalos de valor em que o desconto deve ser gerado. Neste processo também se verifica o incremento do número de *pageview*'s (número de visualizações) para aquela oferta. O número de vezes que o cliente observou o produto terá influência no desconto do mesmo. A resposta por parte do *Optipricer* poderá não ter conteúdo ou ser um texto em formato HTML que serve para apresentar na página um *design* personalizado disponibilizado pelo servidor.

Num 2º passo existe um *render* da página (3 - *renderPage()*), onde a loja coloca os dados serem enviados para o *Optipricer*.

Posteriormente, existe o login no *Facebook* por parte do utilizador, e para este aceitar as permissões pedidas pela APP do *Optipricer* (4 - *login()* e 5 - *response*). Na resposta vem o *Facebook id* e o *Facebook token* que

serve para o *Optipricer* autenticar o utilizador, ter acesso ao seu perfil para o cálculo de descontos e ainda para proceder á partilha do produto no perfil social do utilizador.

De seguida, existe uma comunicação do cliente (*browser*) com o *Optipricer* (6 - *createCoupon()* e 7- *response()*). Nesta comunicação são enviados ao servidor os dados para a criação do cupão juntamente com as credenciais do *Facebook* e um campo com o comentário a ser incluído na partilha do produto. Como resposta é recebido um cupão de desconto que contempla o código do cupão, o preço final do produto, preço original do produto, o id do produto e a data de expiração do cupão.

Por fim, com os dados obtidos na comunicação anterior, o cliente (*browser*) cria uma *cookie*, que será enviada para a loja (8 - *createCookie()*). É a partir desta *cookie* que o servidor (loja) tem acesso aos dados de desconto e certifica que o desconto já está criado, evitando novas comunicações com o *Optipricer*.

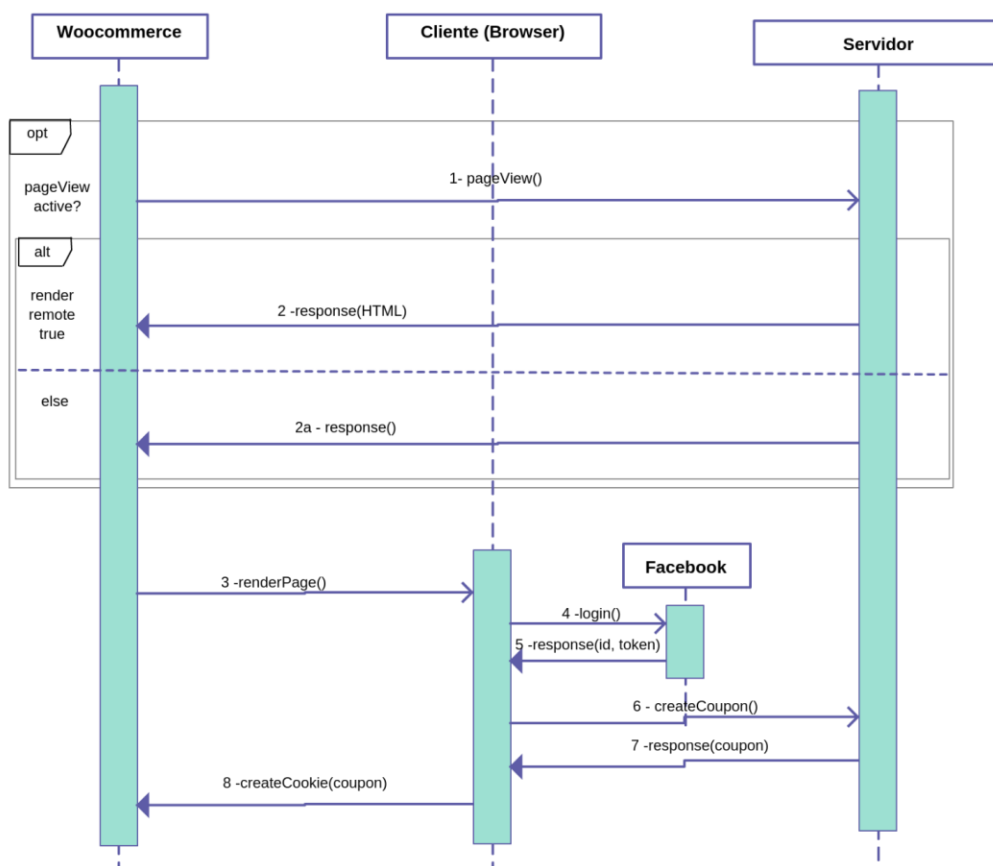


Figura 27 - Diagrama de Sequência de obtenção de um desconto

Os dados seguintes descrevem cada comunicação bem como os dados que circulam em cada uma:

- *pageView()*: Esta comunicação serve para determinar o número de vezes que um determinado produto é visto. Portanto, existe a criação da oferta (se já não existir) e deste modo é necessário enviar certos dados: os dados da loja, os dados do produto, um *sessionId* que corresponde a um identificador do

cliente, e as *social_credentials* que correspondem às credenciais do utilizador no *Facebook*. No *header* da mensagem, como *authorization* é obrigatório enviar o *token* identificativo da loja. Na Figura 28 encontram-se os dados a serem enviados nesta comunicação, em formato JSON:

```
1  "ssid*": "",
2  "data": {
3    "min*": "",
4    "max*": "",
5    "location*": "",
6    "text*": "",
7    "discount_render*": "",
8    "expiry_offset*": "",
9    "product_id*": "",
10   "name*": "",
11   "link*": "",
12   "product_brand*": "",
13   "product_barcode*": "",
14   "price*": "",
15   "categories*": ""
16  },
17  "social_credentials": {
18    "facebookID": "",
19    "facebookToken": ""
20  }
```

Figura 28 - Dados a serem enviados na comunicação *pageView()* e *createCoupon()*

Segundo a figura, os campos “min” e “max” apresentam o intervalo de valores em que deve ser gerado o desconto, “text”, uma descrição sobre o desconto (para usos futuros), a “location” que corresponde à localização do cliente (para usos futuros), o “discount_render” que representa uma *flag* que indica se o *design* da página de desconto é obtido via remoto ou localmente e, por fim, o “expiry_offset” que corresponde ao tempo em que o desconto se encontra válido.

Os campos com um asterisco (*) representam dados mais sensíveis a serem enviados. Certos dados são sensíveis ao nível da privacidade (como por exemplo informações da loja ou mesmo do produto que não podem ser partilhadas com o cliente), outros a nível de autenticidade (não podem ser manipulados, no entanto podem ser vistos). A maioria dos dados representam um problema de privacidade, ou seja, necessitam de ser cifrados para não serem visíveis ao cliente. A cifragem (se bem efetuada) também garante autenticidade.

- 2 e 2a - *response()* (*pageView()*): esta comunicação funciona como uma resposta ao *pageView()*. Esta resposta tanto pode ser “no conteúdo” (sem nenhum conteúdo) ou um código HTML que será *renderizado* na página (depende da *flag* “discount_render” enviada previamente).
- 3 - *renderPage()*: representa o render da página, bem como o espaço para obtenção de desconto e os dados a serem utilizados pelo *browser* do cliente para criação de cupão.
- 4 e 5 - *login()* e *response()*: representam comunicações com o *Facebook*. Nestas o utilizador deverá fazer login com as suas credenciais do *Facebook* e aceitar as permissões que a aplicação do *Optipricer* pede (se ainda não o tiver feito), a partir do *login()*. No *response()*, o *Facebook* responde com o *Facebook ID* e o *Facebook Token* que representa a autenticidade do utilizador para aquela aplicação.

- 6 - `createCoupon()` : esta comunicação pede ao servidor para ser criado um cupão de acordo com os dados enviados. Nesta comunicação são enviados os mesmos dados enviados em 1 e ainda o campo “comment” que corresponde ao comentário a ser inserido na partilha. O “comment” é um dado que não é sensível visto ser criado pelo utilizador. Apenas deveremos ter cuidado com os demasiados ataques já conhecidos como *Cross-site scripting* e *SQL Injection*.
- 7 - `response()` (`createCoupon()`): Em resposta à comunicação “`createCoupon()`” é enviado o respetivo cupão com o valor do desconto. No cupão de desconto são enviados os seguintes dados, no formato JSON, como enunciado na Figura 29:

```

1  "data": {
2      "ssid": "",
3      "expiryDate": "",
4      "value": "",
5      "token": "",
6      "product_id": ""
7  }

```

Figura 29 - Dados do cupão criado

Nesta comunicação todos os dados são sensíveis, ou seja, é necessário assegurar que não são manipulados. Por exemplo, o “value” (que corresponde ao preço final do produto) é altamente requerido que não seja manipulado, tal como o “token” do cupão para futuramente ser validado. No entanto, estes dados não são sensíveis para a loja, ou seja, podem ser visíveis ao utilizador. Neste caso, não é necessário garantir privacidade dos dados, apenas autenticidade dos mesmos. Deste modo, pode-se incluir nos dados uma assinatura sobre eles, de forma a que estes não sejam manipulados, e por fim sejam validados no servidor. No final, temos os seguintes dados:

```

1  "data": {
2      "ssid": "",
3      "expiryDate": "",
4      "value": "",
5      "token": "",
6      "product_id": "",
7      "sign": "signature(ssid: expiryDate: value: token: product_id)"
8  }

```

Figura 30 - Dados do cupão criado assegurados

A Figura 30 mostra que os dados do cupão criado são assegurados com base numa assinatura. Esta assinatura (“sign”) é criada a partir dos valores “ssid”, “expiryDate”, “value”, “token” e “product_id”, mitigando assim a sua manipulação.

- 8 - `createCookie()`: depois de o cliente ter recebido os dados do servidor remoto, o *browser* cria uma *cookie* com esses dados de modo a serem utilizados pela loja, e a identificar o cliente (e o respetivo desconto) na loja. O nome da *cookie* pode ser “discount_[token_store]_[product_id]”, no entanto é

importante que o desconto no produto não seja centralizado no nome da *cookie*, ou seja, que não se utilize o “product_id” do nome da *cookie* como referência para aplicação do desconto, porque o mesmo pode ser manipulado. Deve ser sempre feita uma validação entre o que se encontra dentro da *cookie* com a respetiva assinatura. Nesta abordagem, teríamos várias *cookies*, dependendo do número de produtos. Neste caso, o *JavaScript* deve ser “cego”, ou seja, deve copiar os dados do cupão para a *cookie*. Se os dados forem percebidos pelo *JavaScript* (por exemplo um objeto JSON), podem ser utilizados, não sendo preciso que o servidor envie uma réplica dos dados por baixo do conteúdo protegido.

Os problemas que podem surgir nos diversos pedidos são os seguintes: para as comunicações 1 - *pageView()* e 2 / 2a - *response()*, podemos ter ataques MITM, sendo que na resposta pode ser possível injetar código JS (XSS) prejudicial ao cliente, replicação de pedidos / respostas e visualização dos dados.

Na comunicação 3 - *renderPage()*, temos o problema de manipulação e visualização dos dados por parte do cliente, a serem enviados para o servidor, pois os dados são colocados no DOM, mesmo que sejam invisíveis na página, são de fácil visualização e manipulação. O mesmo se justifica via *JavaScript*, em que os dados a serem enviados para o servidor podem ser manipulados e visualizados no código.

No *login()* e *response()* apenas poderá existir o ataque MITM, um atacante poderá recolher as credenciais do utilizador para autenticação na aplicação do *Facebook* no *Optipricer*, de modo a conseguir descontos melhores.

Para a comunicação 6 - *createCoupon()* poderão surgir problemas que advêm da comunicação 3 e os mesmos ataques existentes na comunicação 1 e 2/2a. Por exemplo, um atacante a partir de um ataque “Man In The Middle” poderá criar cupões com base nos dados que intercetou, que não são para si.

Na 7 - *response()* ao *createCoupon()* é possível ataques MITM e replicação de respostas.

Por último, no *createCookie()*, é possível criar a *cookie* com dados fictícios, a partir do *JavaScript* e utilização dos mesmos dados de desconto para um produto diferente.

No geral, de modo a resolver ataques MITM, podemos utilizar o protocolo HTTPS com todas as suas vantagens, ou utilização de cifras / assinaturas para proteção de dados. Quando for necessário privacidade dos dados, a cifra deve ser o processo a utilizar. Para resolver a replicação de pacotes, pode ser incluído na mensagem um *timestamp* ou um campo gerado aleatoriamente, permitindo diferenciação de cada mensagem enviada por cada uma das entidades. No caso da resposta (2 e 2a), de modo a resolver ataques de injeção de código malicioso em *JavaScript*, podemos utilizar as mesmas soluções especificadas anteriormente para ataques MITM, de forma a impedir que um atacante manipule pacotes. De modo a não consentir a manipulação dos dados pelo utilizador (como é o caso nas comunicações 3. e 8.), os dados devem ser cifrados ou assinados. Como na comunicação 3 - *renderPage()* também é necessário que os dados da loja sejam privados (não visíveis) visto que não se deve expor informações da loja. Assim, estes dados devem ser cifrados. Na comunicação 8 - *createCoupon()*, os dados podem ser assinados ou cifrados. De modo a dar autenticidade ao servidor *Optipricer*, ou seja, de forma a garantir que o cupão foi realmente produzido ele, os dados devem ser

assinados com uma chave privada do *Optipricer*, e devem ser validados com a respetiva chave pública, por parte da loja.

Em suma, para garantir segurança neste fluxo, cabe às entidades principais (loja e *Optipricer*) cifrar ou assinar o conteúdo de cada comunicação.

Neste fluxo é consumido o seguinte serviço da API do *Optipricer*, presente na Figura 31.

POST /api/coupon/

Documentation Sandbox

Documentation
Coupon creation.
Create a new coupon if the offer does not exist, otherwise returns the currently active.

Request Header

Field name	Description	Required	Default	Example/Other options
Authorization	Authorization Token	Yes	-	Token nest99475e3e
Accept	Response format expected	No	application/json	text/html
Accept-Language	Response language	No	en_GB	pt_PT

Example:

```
Header Format:
"Authorization": "Token nest99475e3e"
"Accept: text/html"
"Accept-Language: pt_PT"
```

Request Format (Json content)

Field name	Type	Required	Example
data *	Encrypted Text	Yes	eyJwcm9kdWN0X2lkIjoINTEiL...==
social_credentials	Array	Yes	{ "facebookId": "123456789", "facebookToken": "JHG%/(25GYJ&%&GFVRD%}"
comment	Text	Yes	Hey, check this product...

Figura 31 - Método da API do *Optipricer* para obtenção de descontos

Nesta API é necessário incluir um *Authorization Header* que contém o *token* identificativo da loja. Esta *header* tem dois objetivos: evitar replicação de mensagens por terceiros (o servidor só aceita mensagens com a *header* bem especificada) e identificar a loja, explicitando que o cupão é pedido para a loja (neste caso *Token nest99475e3e*). A identificação da loja é um requisito importante, visto que também é através dela que é possível obter a chave partilhada entre ambas as entidades, a partir da base de dados no *Optipricer*. Portanto, se alguém conseguir interceptar uma mensagem e retirar o *header* de autorização e incluí-lo nas suas mensagens (utilização do serviço por uma loja não autorizada), não saberia como cifrar as mensagens, pois não tinha acesso à chave partilhada. Em suma, a decifra de texto com a chave errada ou interpretação errada de

mensagens (mensagens por cifrar), deve ser um requisito base para o *Optipricer* ignorar esses pedidos. O “*accept*” header tem por defeito “application/json” mas também aceita “text/html”, que poderá conter código em HTML para ser imprimido na página. Ainda é possível incluir outra header, *Accept-Language* que representa em que idioma deve vir a resposta (por defeito *en_GB*).

Como dados a serem enviados no corpo da mensagem, temos o campo “data” que é conteúdo cifrado, *social_credentials* que é um array em JSON que contém as credenciais de acesso à aplicação do *Optipricer* no *Facebook*, por exemplo, e o *comment* que representa o comentário que o utilizador quer deixar na partilha na rede social. Esta API é já uma proposta de resolução do problema, pois inclui o campo data cifrado, de modo a proteger os dados de terceiros. A Figura 32 especifica quais os dados incluídos no campo “data”, que posteriormente serão protegidos, através da documentação da API:

• **data** field structure
The following fields are from the eCommerce platform and security measures are used in order to preserve its integrity.

The type of this field is an Array with the following content:

Field name	Description	Type	Required	Example
product_id	Product Id from the eCommerce platform	Value	Yes	121
name	Product name	String	Yes	"Couch"
description	Product description (short)	String	No	"Comfort and style in a low maintenance package."
price	Product price	Number	Yes	599.99
image_url	Product image link	String	Yes	"http://optipricer.com/magento-demo/couch.jpg"
link	Product link	String	Yes	"http://optipricer.com/magento-demo/furniture/living-room/couch.html"
currency	Currency	String	No	"€"
min	Minimum discount of the offer	String	Yes *	"5"
max	Maximum discount of the offer	String	Yes *	"30"
text	Text of the offer	String	Yes *	-
expiry_offset	Expiry offset (in minutes)	Number	Yes	2880

* Min, Max and Text fields can not be null simultaneously.
If Text field is filled, Min and Max can be null and it means that it's a promotion.
If Max is defined, it is considered a fixed amount discount.
If Min and Max are defined, the discount value will be between the given range.

Example:

```
{
  "product_id" => "53",
  "name" => "Couch",
  "description" => "Inspired by the classic camelback sofa.",
  "price" => 599.99,
  "image_url" => "http://optipricer.com/magento-demo/couch.jpg",
  "link" => "http://optipricer.com/magento-demo/furniture/living-room/couch.html",
  "currency" => "€",
  "min" => "5",
  "max" => "30",
  "text" => "",
  "expiry_offset" => "2880"
}
```

Figura 32 - Documentação da API de obtenção de desconto (dados campo data)

Analisando a Figura 32, temos presentes os dados já referenciados nesta secção para a obtenção de um desconto e como devem ser incluídos, em formato JSON, na mensagem. Portanto, a aplicação sem proteção contempla o campo “data” diretamente com os dados presentes na figura (como é o caso do “Example”). Esta figura demonstra quais os dados que a mensagem deve incluir no campo *data* e quais são os extremamente requisitados.

secureContent que visa assegurar conteúdos e o *getContent* que tem como missão obter conteúdo anteriormente assegurado.

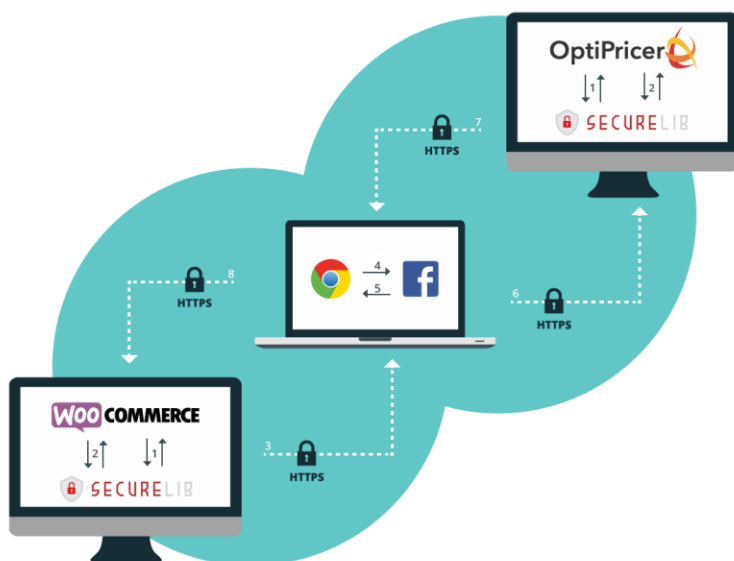


Figura 34 - Diagrama final do fluxo de obtenção de um desconto

Legenda:

- 1 – *secureContent()*, função para assegurar conteúdo
- 2 – *getContent()*, função para obter conteúdo seguro
- 3 – *renderPage()*, função de *render* na página da loja do conteúdo assegurado
- 4 – *login()*, login no *Facebook*
- 5 – *response()* a 4, que inclui *facebookId* e *facebookToken*
- 6 – *createCoupon()*, envio de dados do *Facebook* e dados segurados
- 7 – *response(coupon)*, envio do respetivo cupão
- 8 – *createCookie()*, criação de uma *cookie* com o cupão

De seguida, na próxima secção, iremos analisar o terceiro fluxo, de adição de um produto com desconto ao carrinho de compras, as vulnerabilidades que poderão ocorrer neste processo e apresentaremos soluções sucintas para as mesmas.

4.2.3 Fluxo de adição de um produto ao carrinho com desconto

Nesta atividade não existem comunicações com o *Optipricer*, apenas existe a interação entre o utilizador e a loja, o utilizador adiciona um produto ao carrinho e a loja tem que garantir que o mesmo é apresentado com o desconto obtido. Para aplicar o desconto ao produto existem três hipóteses:

- Utilização de cupões da loja
- Aplicação direta do desconto antes de adicionar o produto ao carrinho
- Recálculo do preço total do carrinho com base nos descontos obtidos

Neste caso, deveremos descartar a 2ª e a 3ª hipóteses por questões legais: uma loja estaria a vender um produto a um preço mais baixo que o preço real do produto e não haveria nenhuma prova de como essa venda tinha incluído um cupão de desconto do *Optipricer*. Utilizando a primeira abordagem existe feedback do desconto no carrinho e a prova de como o desconto foi providenciado pelo *Optipricer*: esta encontra-se no cupão gerado pela loja.

A Figura 35 apresenta o diagrama de atividades para a primeira abordagem, de adição de um produto ao carrinho e aplicação do desconto no mesmo. Note-se que neste fluxo não existem quaisquer comunicações

entre os servidores (loja e *Optiprizer*) e o cliente. Nesta interação, o produto é adicionado inicialmente ao carrinho, de seguida é testado se existe desconto para o carrinho, se existir, procura um cupão do *Woocommerce* para o desconto obtido (através do *token* do cupão). Se este se encontrar no sistema, então atualiza-o com novos valores (por exemplo adicionando um novo id de sessão). Se não, cria o cupão da loja. Por fim, adiciona o cupão de desconto ao carrinho. Veja-se que na fase 2 (Existe desconto para o produto?) deve-se proceder à validação se o desconto se encontra válido (isto é, se não foi manipulado), para proceder à sua remoção caso o conteúdo do cupão tenha sido adulterado.

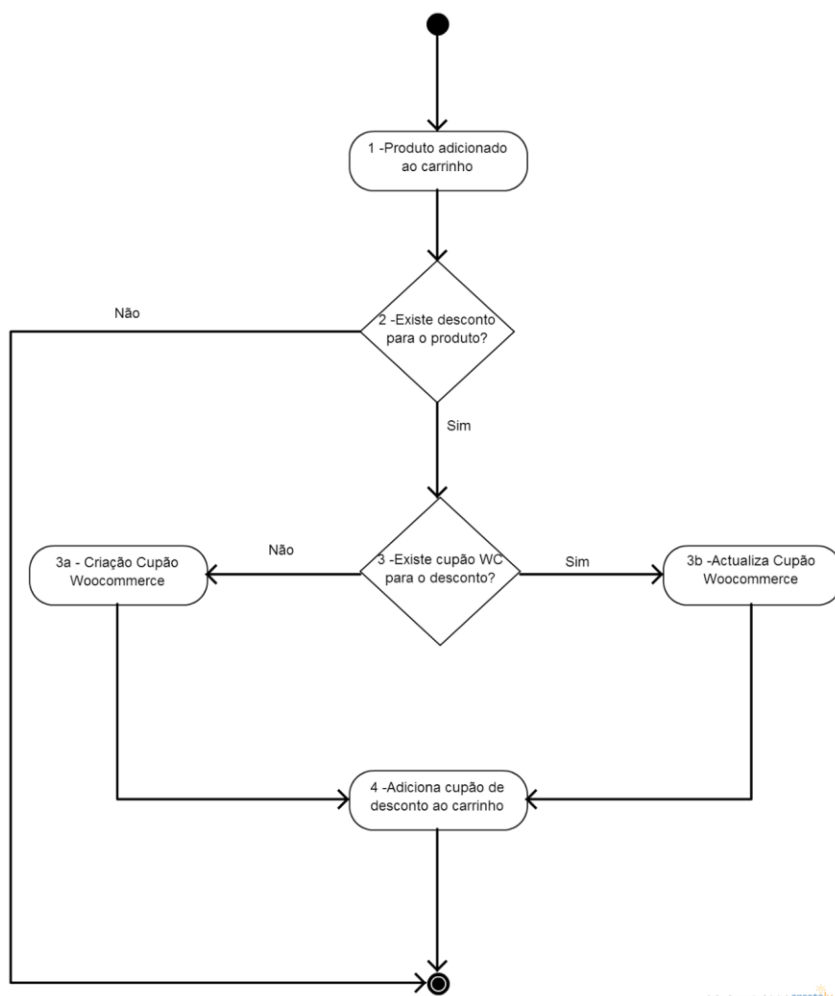


Figura 35 - Diagrama de Atividade de adição de um produto ao carrinho

Como podemos observar, depois da adição do produto ao carrinho e a respetiva adição do cupão de desconto ao carrinho, o fluxo fica independente do uso de *cookies*. Tudo se centralizará no cupão de desconto.

Os dados do cupão são representados por um *array* codificado em JSON, que contempla cada *cookie* associado àquele desconto: o conteúdo da *cookie* é codificado em base64. A utilização do *array* deve-se ao facto de um utilizador poder obter o mesmo desconto a partir de dispositivos diferentes (gerando identificadores de sessão

diferentes). O cupão da loja deve conter todos os cupões com identificadores diferentes, visto estes poderem vir a ser assinados pelo *Optipricer*.

Nesta abordagem podemos ter os seguintes problemas de segurança:

- Manipulação dos dados do cupão pelo empregado da loja: os dados podem vir a ser manipulados para beneficiar / prejudicar o cliente, e prejudicar a loja.
- Remoção de cupões: de um momento para o outro o empregado pode remover o cupão (ou o conteúdo) e o cupão deixa de ser válido / utilizável, obrigando o cliente a obter um novo desconto através do servidor remoto
- Geração de cupões falsos: o empregado da loja pode gerar cupões falsos, com os mesmos dados do cupão do *Optipricer*.

Este é o passo onde o conteúdo que veio do *Optipricer* é validado e utilizado. Portanto é necessário que os passos anteriores estejam bem implementados e seguros. A comunicação com o *browser* do cliente centra-se na *cookie* que contém os dados que o *Optipricer* disponibilizou relativamente ao desconto obtido. Assim, devemos ter a certeza que podemos confiar na *cookie*, que esta não foi manipulada / replicada. Uma das formas é a cifragem do conteúdo do lado do *Optipricer* e colocação daquele diretamente na *cookie*, o *browser* não necessita de validar nada. A *cookie* pode ser manipulada, mas a cifra passa a ser inválida e o desconto invalidado. Este método de segurança deve ser garantido no fluxo anterior (obtenção de um desconto remotamente). De modo a evitar a manipulação dos dados do cupão pelo empregado da loja, devemos criar uma assinatura do lado do *Optipricer*, com a sua chave privada, a ser validada pela loja: uma vez que o empregado da loja tem acesso ao código e aos parâmetros de configuração, saberia criar cupões fictícios com conteúdo cifrado. Mas, com a utilização de assinaturas, apenas o *Optipricer* saberia assinar mensagens, sendo que a loja apenas validaria as mesmas.

Por último, na secção seguinte apresentaremos uma análise sobre o fluxo de checkout de um produto com desconto, estudando as comunicações existentes, as vulnerabilidades possíveis e soluções para as mesmas.

4.2.4 Fluxo de checkout de um produto com desconto

Este último fluxo representa o checkout do carrinho de compras que contém produtos com desconto obtido a partir do *Optipricer*. Neste é necessário validar o desconto internamente e posteriormente validar o desconto com o servidor do *Optipricer*, indicando também que o desconto foi utilizado. Neste ponto não existem interações com o cliente.

No diagrama da Figura 36, apresentamos as interações que devem ser feitas para validação do desconto remotamente, bem como a indicação de que o desconto foi utilizado (*redeem()*). Nesta comunicação temos apenas interações entre a loja e o *Optipricer*, existindo duas comunicações: *redeemCoupon()* da loja para o servidor e a respetiva resposta (*response()*). Existe ainda uma validação que testa se o cupão pertence ao *Optipricer*, de modo a não efetuar validações nem comunicações desnecessárias.

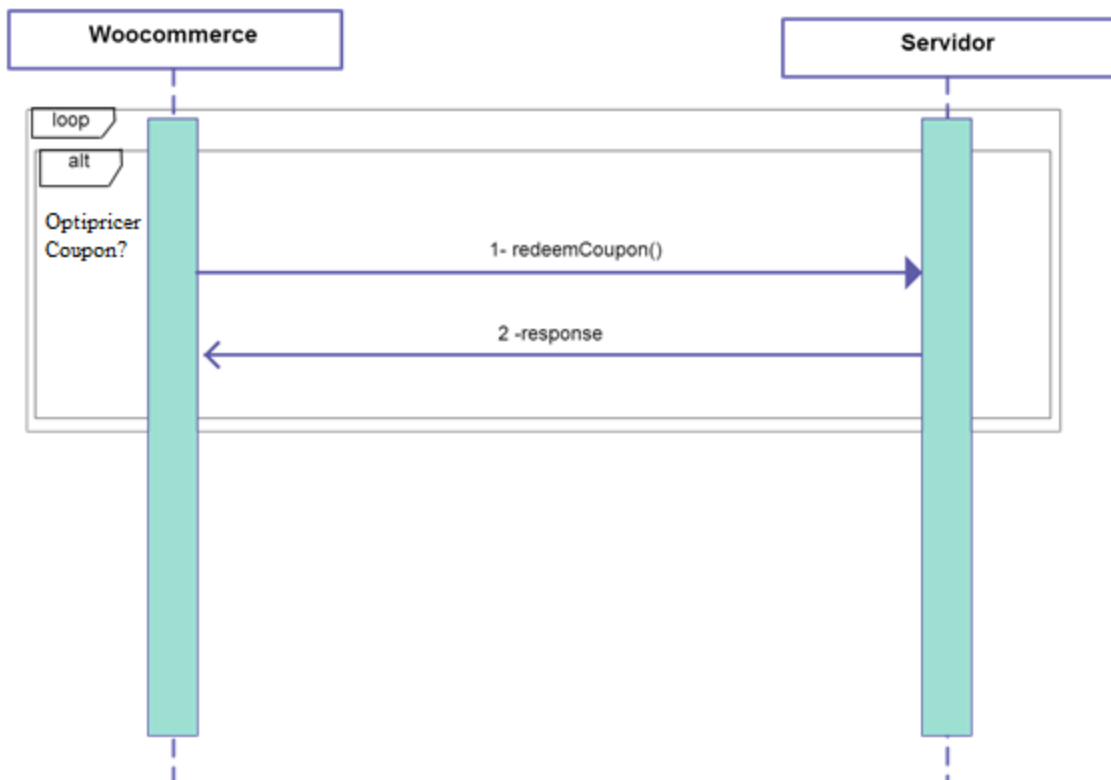


Figura 36 - Diagrama de Sequência do processo de checkout (Redeem de cupões)

Nesta implementação podemos ter dois tipos de *redeem*: o *redeem* atômico (apenas para um produto / desconto) ou o *redeem* múltiplo (para múltiplos produtos e descontos).

No *redeem*, de acordo com a Figura 37, temos os seguintes dados a serem enviados:

```

1  "data": {
2      "notes": "",
3      "location": "",
4      "tokens": array
5  }
  
```

Figura 37 - Dados a serem enviados no *redeem*

O campo *notes* pode incluir todos os produtos vendidos no carrinho e o id do pedido (*order*).

O *location* corresponde à localização do cliente que está a fazer checkout.

O campo *tokens* corresponde a um *array* de cupões. Ora, se for atômico o *array* só vai conter um cupão. Prevemos que possa incluir outros dados do cupão para serem validados no servidor remoto como por exemplo o valor do mesmo e o produto a que é aplicado.

Na resposta, deverá vir algo assinado pelo *Optipricer*, para validar que o cupão foi mesmo *redeem*, e para assegurar que não haja ninguém no meio da comunicação a replicar validações de *redeem*, para qualquer mensagem.

Com esta abordagem podemos ter os seguintes problemas de segurança:

- Utilização de cupões de desconto que não pertencem ao utilizador através do código: o utilizador pode partilhar o código de um cupão com um outro utilizador
- Outros problemas enunciados no fluxo de adição do produto ao carrinho
- Replicação de pedidos ao servidor: envio múltiplas vezes do *redeem* do cupão
- Manipulação de respostas a virem do *Optipricer* (validar descontos que já não são válidos ou que não estão corretos)

De modo a resolver os problemas referenciados anteriormente, é necessário fazer uma validação interna: validar se o desconto é válido com a data de validade do cupão, referenciada pelo *Optipricer*, validar se o utilizador do cupão é mesmo o dono do cupão através de um *sessionID* (variável de sessão única para cada utilizador) e validar se o valor do cupão é o mesmo que o *Optipricer* disponibilizou. Com efeito, é necessário que um campo do cupão da loja (por exemplo a descrição do cupão) tenha lá estes dados armazenados. Estes dados devem ser cifrados ou assinados de modo a impedir a sua manipulação e a providenciar a sua validação. A validação remotamente passa por a loja fazer uma comunicação com o *Optipricer* que envia os dados do cupão para o servidor, na comunicação *redeemCoupon()*.

PUT /api/coupon/{couponToken}/redeem

Documentation Sandbox

Documentation
Redeem an existing coupon.

Request Header

Field name	Description	Required	Default	Example/Other options
Authorization	Authorization Token	Yes	-	Token nest99475e3e

Example:

```
Request Header Format:
"Authorization": "Token nest99475e3e"
```

Request Format (Json content)

Field name	Type	Required	Example
data*	Encrypted Text	Yes	cm9kdWN0X2lkIjoI...==

* See **data** details below

Example:

```
Content (Raw Data):
{
  "data": "cm9kdWN0X2lkIjoI...=="
}
```

• data field structure

Field name	Description	Type	Required	Example
tokens	Coupon Token	Array	Yes	["aveiro12345abcd12265523dfba9dc1e"]
notes	Important notes	String	No	"Quantity purchased: 3"
location	Location	String	No	"Aveiro, Portugal"

Example:

```
Content (Json)
{
  "tokens" => ["aveiro12345abcd12265523dfba9dc1e"],
  "location" => "Aveiro, Portugal",
  "notes" => "Quantity purchased: 3"
}
```

Figura 38 - Documentação da API de redeem de cupões

Observando a Figura 38, podemos concluir que os dados que devem constar na mensagem de *redeem* são o “tokens”, que representa o *token* do cupão a ser *redeem*, o “location”, que corresponde à localização do utilizador (para usos futuros), e o “notes”, que visa incluir diferentes notas sobre a compra, por exemplo, outros produtos comprados ou a quantidade obtida do produto que contém o desconto (terceiro exemplo, “Example”). Neste caso, propomos a utilização de cifras, pelo segundo exemplo, “Example”, o conteúdo da mensagem é um campo “data” que apresenta um texto codificado em base 64, que representa a cifra resultante da mensagem do terceiro exemplo. Quanto às *headers* requisitadas, é comum encontrarmos a *header* de autorização presente noutros pedidos (“Authorization Header”). Esta *header* deve ser sempre incluída de modo a evitar replicação de mensagens para o servidor, como já foi discutido anteriormente a propósito de outros fluxos. Ao contrário de outras APIs aqui mencionadas (fluxo de ativação e fluxo de obtenção de desconto) que utilizam o método POST, esta API é do tipo PUT, ou seja, o *token* do cupão deve ser incluído no URL aquando da utilização deste serviço.

Por fim, no próximo diagrama demonstramos como podemos proteger as comunicações deste fluxo, através da implementação de uma camada de segurança, já mencionada no ponto 4.2.2.



Legenda:

- 1 – `secureContent()`, função para assegurar conteúdo
- 2 – `getContent()`, função para obter conteúdo seguro
- 3 – `redeemCoupon()`, leitura do cupão
- 4 – `response()`, à leitura do cupão, normalmente sem conteúdo

Figura 39 - Diagrama de arquitetura final do fluxo de checkout de cupão (redeem)

Neste diagrama, propomos a resolução do problema de checkout de um produto com desconto, quando a loja valida com o *Optipricer* a utilização do cupão. Neste fluxo, além das comunicações seguras entre a loja e o *Optipricer*, através do protocolo HTTPS acrescentamos também uma camada de segurança (*SecureLib*) que visa proteger os dados.

Terminado o estudo da arquitetura do projeto, procederemos, no próximo capítulo, à apresentação da implementação do fluxo e da camada de segurança, com vista a promover uma extensão segura para uma loja online do *Optipricer*.

5. Comunicação segura no *Optipricer*

Neste capítulo iremos abordar a implementação da solução proposta no capítulo anterior. Como analisámos numa secção anterior deste trabalho, o projeto contém quatro fluxos principais, entre a loja e o *Optipricer*, sendo o primeiro o fluxo de ativação; o segundo o fluxo de obtenção de desconto; o terceiro o fluxo de adição do produto ao carrinho com desconto; e por último, o quarto fluxo que representa o checkout do produto com desconto. A propósito de cada fluxo será especificada a solução adotada por nós, ilustrando o comportamento da loja e do *Optipricer*, através de um *plugin* que desenvolvemos para o *Woocommerce*.

O *Woocommerce* funciona por eventos (*actions & filters*): para qualquer ação que se faça na loja é disparado um evento. Por exemplo, ao adicionar um produto ao carrinho é disparado o evento (*Woocommerce_add_cart_item*). Na construção de uma extensão para o *Woocommerce*, podem-se adicionar ações (ou filtros) aos eventos, de modo a executarem o nosso código. Desta forma, todo o código de cálculo do preço do produto antes de ser adicionado ao carrinho e a validação do checkout para *redeem* do cupão são feitos do lado do servidor da loja, garantindo assim segurança no carrinho de compras e no processo de checkout.

Os fluxos do *Optipricer* no *plugin* que implementámos funcionam previsivelmente em três fases. Numa fase inicial, é adicionado um bloco de código HTML à página, que constitui o espaço para obtenção de um desconto, através de um evento (*woocommerce_after_add_to_cart_form*); nesta fase, também é adicionado ao DOM conteúdo invisível ao utilizador necessário para a obtenção de um desconto. Numa segunda fase, é adicionado código que pretende interagir com a loja quando um produto é adicionado ao carrinho: se existir um desconto, deve aplica-lo ao carrinho de compras, criando um cupão na loja; este código é executado através do disparo de um evento (*woocommerce_add_to_cart*). Por último, na terceira fase, é acionado um evento (*woocommerce_after_checkout_validation*) que visa interagir com a loja no processo de checkout: se o carrinho de compras conter cupões de desconto do *Optipricer*, estes devem ser validados e deve ser informado o *Optipricer* que o cupão foi utilizado. Além destas fases implementadas, foi ainda desenvolvido código para configurar as opções do *plugin* e código correspondente ao primeiro fluxo: fluxo de ativação.

Por fim, com os fluxos implementados no *plugin*, devemos garantir que estes são seguros, ou seja, os dados que circulam nestes não podem ser adulterados, e não podem ser visualizados, em alguns casos. Para tal, implementámos uma biblioteca de segurança como objetivo de proteger os dados de cada fluxo. Esta biblioteca disponibiliza diversos métodos criptográficos, e pode ser aplicada a outros *plugins*, providenciando segurança à loja, como vemos mais à frente, no ponto 5.5 deste trabalho.

Em suma, o trabalho realizado que discutimos nesta secção, inclui a implementação de fluxos seguros no *Optipricer* que foram descritos na secção anterior, um *plugin* para o *Woocommerce* de modo a testar e validar todo o processo e por fim uma biblioteca de segurança que visa proteger conteúdo e informação impedindo que esta seja adulterada.

5.1 Fluxo de ativação

O fluxo de ativação é manual, sendo feito através das definições do *plugin*. Neste fluxo, o cliente deve preencher um formulário com o seu nome, *email* e algumas observações. Com estes dados, é consumido um serviço na API do *Optipricer* que envia um *email* com um novo contacto feito. Cabe ao administrador do *Optipricer* criar um *token* de autenticação da loja e a chave que será utilizada nos processos de cifra e decifra de dados.

A Figura 40 apresenta o menu para as definições do *plugin* do *Optipricer* no *Woocommerce*. É a partir deste menu que podemos aceder às definições do *Optipricer* no *Woocommerce*. Este menu está presente no painel de administração do *Wordpress*. A Figura 41 apresenta o primeiro ecrã das definições do *plugin*, quando estas já estão preenchidas e o contacto feito. Como podemos observar, neste ecrã é possível ter acesso a determinadas informações como um guia de instalação e o *website* oficial do *Optipricer*. Também é possível verificar o estado do sistema e o estado da conta.

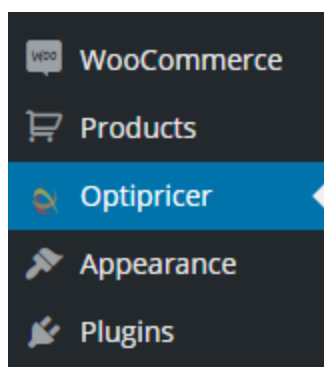


Figura 40 - Menu no backoffice do Wordpress para configurações do Optipricer

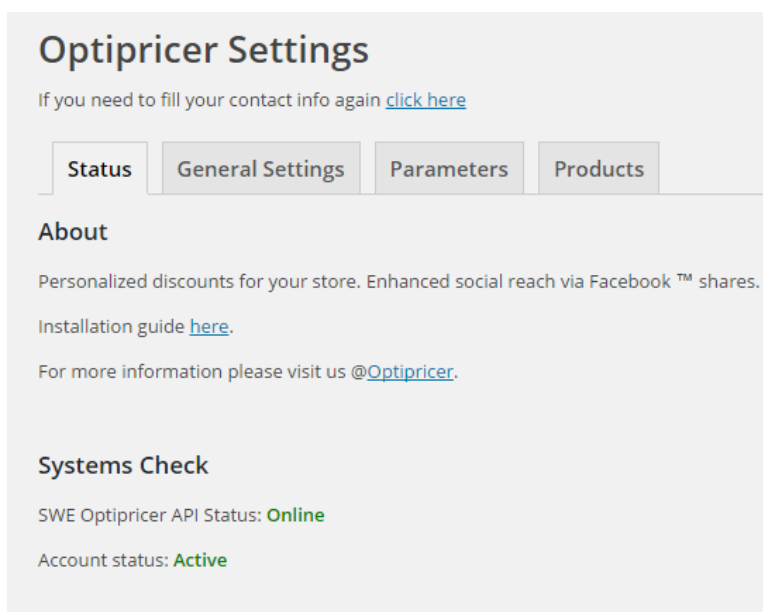


Figura 41 - Menu inicial de opções do Optipricer

De acordo com a Figura 41, o ecrã encontra-se no separador *Status*. Os outros separadores serão estudados mais adiante.

A Figura 42 apresenta o ecrã de submissão de contacto para aderir ao *Optipricer*. É com este ecrã que o utilizador se depara pela primeira vez (nas definições do *plugin*), após instalar o *Optipricer* no *Woocommerce*.

The screenshot shows a form titled "Optipricer Settings" with the heading "Thanks for downloading the Optipricer Plugin!". Below the heading, it says "Now we will contact you. Just fill your data:" and "If already fill your contact info, and if you have the values to set up [click here](#)". The form contains three input fields: "Your name:" (labeled 1), "Your email:" (labeled 2), and "Some obs:" (labeled 3). A "Send info!" button is located at the bottom left of the form area.

Figura 42 - Ecrã de definições para submissão de informações de contato

Analisando a Figura 42, o utilizador pode preencher as suas informações de contato para serem submetidas para o *Optipricer*, após a ativação do *plugin*, através do ecrã de definições do *Optipricer* no *backoffice* do *Wordpress*. O cliente deve preencher o formulário com o seu nome (1), *email* (2) e algumas observações (3). Ao clicar no botão “Send info!”, as informações são enviadas para o *Optipricer*, a partir do consumo da API estudada no capítulo 4 (Arquitetura e objetivos do projeto), no fluxo de ativação. Se o administrador da loja já detiver os parâmetros de configuração pode sempre ignorar este passo, clicando no *link* presente em “click here” em sublinhado.

Depois de as informações de contacto terem sido enviadas, devem ser preenchidas as definições do *plugin*. Nestas definições constam o *token* identificativo da loja e a chave partilhada entre ambas as entidades (disponibilizados pelo *Optipricer*), o URL do serviço (neste caso do *Optipricer*), e os parâmetros a serem configurados pela loja (intervalos de desconto, o tempo de validade do cupão, render remoto ou local, e informação sobre se o “pageviews” está ativo).

The screenshot shows the "Optipricer Settings" page with tabs for "Status", "General Settings", "Parameters", and "Products". A "Save options" button is highlighted with a red box. The "General Settings" tab is active, showing four configuration items: "Store Token" (labeled 1) with value "nest54e1c061823aa", "Store Encryption Key" (labeled 2) with value "c4fdfe7fd8430329ca83c214bc6a93d5", "Swe URL Endpoint" (labeled 3) with value "http://www.optipricer.com/api/", and "Active plugin" (labeled 4) with a checked checkbox.

Figura 43 - Ecrã de definições para configuração dos parâmetros do Optipricer na loja

Na Figura 43 constam parâmetros de configuração do *Optipricer* na loja, o *token* identificativo da loja (1), a chave de cifra partilhada com o *Optipricer* (2), o URL da API do *Optipricer* (3) e a definição que visa ativar o *plugin* (4), com *check* caso ativo. O botão com realce (vermelho) serve para guardar as opções após editadas. Como podemos verificar, as definições do *plugin* encontram-se no separador *General Settings*, que constitui as definições gerais de configuração do *plugin*. Estes são os dados disponibilizados pelo *Optipricer* (partilhados com o mesmo), que diferem de cliente para cliente: cada cliente (loja) deve conter um *token* identificativo (1) e uma chave partilhada com o *Optipricer* (2). Apenas o URL da API do *Optipricer* (3) é que é comum a todos os clientes.

Avançando para o separador seguinte, temos os parâmetros de configuração (*Parameters*). A figura que se segue apresenta isso mesmo, para o *plugin* do *Optipricer*:

Optipricer Settings

If you need to fill your contact info again [click here](#)

Status General Settings Parameters Products Save options

Expiry Offset (in minutes) 1 120

Minimum discount 2 5

Maximum discount 3 30

Enable Page Views 4

Render template 5 Local Remote

Figura 44 - Definições da loja para os descontos provenientes do *Optipricer*

Com base na Figura 44, é possível perceber por quanto tempo o desconto é válido (1), os intervalos de desconto em percentagem (2 - mínimo e 3 - máximo), se as “pageviews” (número de visualizações de uma página de produto) estão ativas (4) e se o render do *template* deve ser local ou remoto (5). Estes parâmetros dependem sobretudo das opções de escolha do cliente (loja), que pode personalizar a seu gosto o intervalo de valores de desconto, o tempo de validade do mesmo, entre outros. Como se pode ver na Figura 43, realçámos também o botão para guardar as definições, com um retângulo vermelho.

Desta forma, com estes parâmetros bem configurados, é possível obter descontos através do *Optipricer*. Relembramos que os parâmetros mais importantes que devem estar bem configurados são os presentes no separador *General Settings* (Figura 43). Se estes estiverem bem configurados (incluindo o URL do *Optipricer* (3)), o resultado aparecerá no separador *Status* (Figura 41), com o “Swe Optipricer API Status” com o valor de *Online* e o “Account Status” como *Active*.

5.2 Fluxo de obtenção de desconto

Nesta secção será descrita a implementação do fluxo de obtenção do desconto, dentro da loja, através do *Optipricer*. Como analisámos na secção referente à arquitetura, este fluxo conta com a participação de três entidades: a loja, o *browser* do cliente e o *Optipricer*.

Inicialmente, o *plugin* é responsável por fazer a produção de uma parte da página de produto da loja, despoletando um evento para adição de conteúdo em HTML.

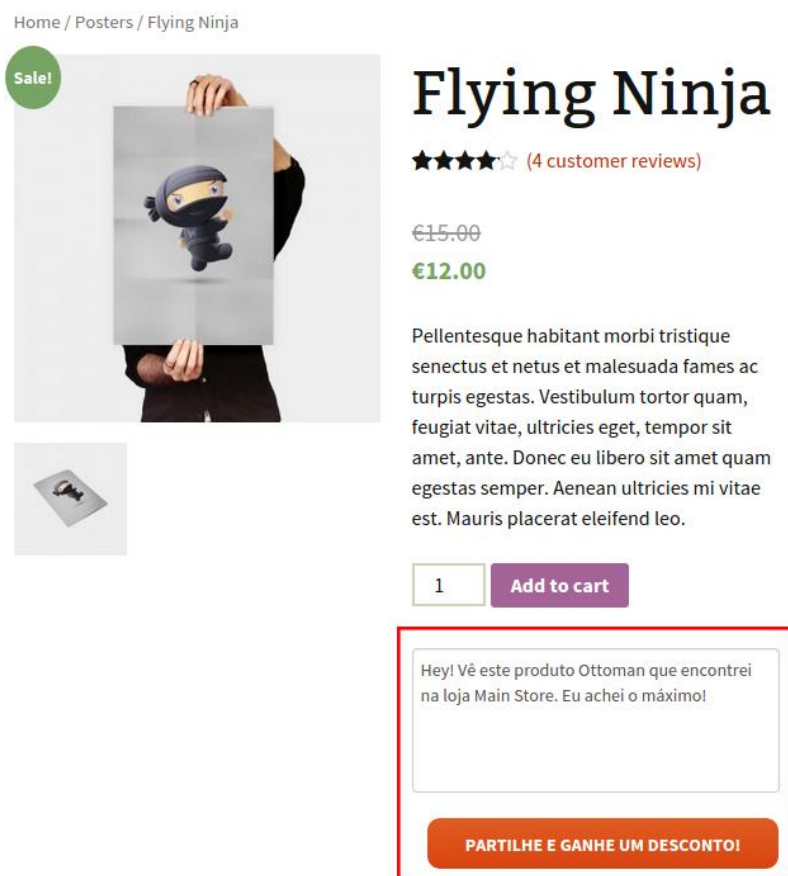


Figura 45 - Página de produto para obtenção de desconto, com o *plugin* do *Optipricer*, no *WooCommerce*

Na Figura 45, destacamos um bloco adicionado pelo *plugin* que permite interagir com o utilizador e com o *Optipricer* para a obtenção de um desconto (com um retângulo vermelho). Este bloco é colocado na página através da ativação de um evento do *WooCommerce*, que permite adicionar conteúdo após o botão de adição de um produto ao carrinho (botão *Add to cart*). O bloco inserido apresenta uma caixa de texto que serve para o cliente da loja escrever um comentário aquando da compra do produto, que será o comentário que aparece na partilha do produto nas redes sociais. O botão de “Partilhe e ganhe um desconto!” tem como intuito a comunicação com o *Optipricer* (e também com o *Facebook*, para recolha de credenciais de acesso à aplicação), para proceder à partilha do produto no meio social e também para o cálculo do desconto adequado a cada tipo de cliente.

Juntamente com este bloco são ainda colocados no DOM dados da loja e do produto a serem enviados para o *Optipricer*, mas invisíveis ao utilizador:

```

▼ <div id="dataCipher" style="display:none;">
  "
  {"content": "fr7MbzDppYUoYk3xIWgF8N1LoSqho38ZwGooRRhblD8J3N7r8Eu8HRZRu0lP55jNwyFQD\YJyGj78MBu5x9C5iK1soGjJUAaskzEIPxo9\NVXvqe6vzon
amZm5VdBR4pL5955+DnbnrbWlVIAeqnrFoQOu0p6Xi01CrFrkyVEHmgTomaR+uER05ARJvLcY7j2J1DEjAVcygx91P3mQKdmh8fEBxGzsyPDWJsPvebvODVRSODkN8YIRvwPVb
ffWd0xIrmV2FN6ebovjtyrdvzqupa7R2zMbtPuTsUjWLF\qhnt8\8b2p1dQjvFUNiK8AROLJJ8xZKuJGRRaREF6981K\6LTVzztEMdy\6EBss7qAj5eQ6rf+yNKuMj1
UxDzZn07+4jDwc\h\PdrcTk63mVdutx1oVuTi0WJYzxun0Fz\KdZuU0\W5skB87HAGoYkSnkDT6pm1AcifvXyFOeG9wsBg14EmI+FTfvTMOoyktbK6e0c00jtHvOtCma
ke0PCrvLnLsgwv9McSL41bgHMxi30M0kGGzuvmAppq8rdMDW141RplWC5PdmPpEx17b0I3qR6tsENM\fxbyxcI5vJ1ue8r\oX1ooAs6KQdeEv66sLiJgbdEdkP23Yoyo
8XMOFqoa\CCumC9EyKI66A\NOMCCZ3EXzX3S5KphaP11cKPBvLJgIzomZ\5z6GEFvF7PFzgeEaLAIIX1CwA2fB1TMk\1uNZ5fcPgZGr6PNP3xW43YEIRvGpZQ2P41
4xyFbxe8b109apDq8jN0MssFaEDzYAQB8ixue3iqIfs1aOjgj+ftwupjB8yy0oW11BRQ4hk6QX2eYDHD7yGV2AZ8dk8g9sErIG3aazExThRa7W2gdK1LfuCXnSeKCPMa35a
HiwN1LA0w5Ib8gyz+ft8vN3LYIQ05adtqLaX2ur8Lx\55uAEjfcOc78bRhojV+xdHeaOozHpb543y80F02sJ\K8+0iESDYnwLqE8o+cEGzNvkIGz2kzA0A0q6+Ugyfqr
Psb0SEYR3n+A==","iv":"xpFkfxCKUa8KkkR7F5kCZw==","hmac":"ea61566faFc52af8bd28b685959f2ad15dc15b786c3f0caec82fb7ae23a9eb9","alg":"AES-
128-CBC","smode":1}"
</div>
<input type="hidden" id="prodID" value="70">
<input type="hidden" id="prodName" value="Flying Ninja">
<input type="hidden" id="tokenStore" value="nest54e1c061823aa">
<input type="hidden" id="prodPrice" value="12">
<input type="hidden" id="prodFPrice" value="€12.00">
<input type="hidden" id="currency" value="€">

```

Figura 46 - Dados do produto e da loja “impressos” na página, a serem enviados para o *Optipricer*

Pela Figura 46, verificamos que os dados apresentam-se em forma de *input*'s e *div*'s invisíveis ao utilizador. Estes dados são necessários para o *browser* do cliente (JavaScript) interpretar e proceder à comunicação com o *Optipricer*: a *div* com id “dataCipher” representa dados cifrados (da loja e do produto), de acordo com a estrutura que apresentámos no ponto 4.2.2, Fluxo de obtenção de desconto. Em relação a este fluxo, do lado da loja, optámos por cifrar o conteúdo, uma vez que é uma operação rápida e que os dados podem ser sensíveis para a loja. Estes dados foram cifrados a partir da biblioteca *MCrypt* do PHP. Os *input*'s representam apenas dados a serem utilizados pelo código *JavaScript* para riscar o antigo preço (Figura 51), e outras informações. Se estes dados forem manipulados, não influenciará o comportamento do *Optipricer* no cálculo do desconto, uma vez que estes dados não seguem para o *Optipricer*. Deste modo, a Figura 45 e a Figura 46 apresentam a comunicação 3 – *renderPage()*, constante no diagrama do Fluxo de obtenção de desconto (Figura 27), na secção 4.2.2.

Após o clique no botão de obtenção de desconto, “Partilhe e ganhe um desconto!”, poderá ser necessário o utilizador fazer *login* no *Facebook* e aceitar as permissões da aplicação do *Optipricer* na rede social. Este processo representa os passos 4 – *login()* e 5 – *response()* no diagrama da secção 4.2.2. O *response()* inclui o id do perfil do utilizador no *Facebook* e o *token* de autorização do utilizador na aplicação do *Facebook* do *Optipricer*. A Figura 47 apresenta a janela de login no *Facebook* com as credenciais do utilizador. Se o utilizador não estiver ligado no *Facebook* este passo é necessário. A Figura 48 representa a janela de aceitação das primeiras permissões da aplicação do *Optipricer* no *Facebook*: são informações necessárias para o cálculo do desconto, como o perfil do utilizador, a sua cronologia, os seus interesses, o seu grupo de amigos entre outros. Por último, a Figura 49 contempla a aceitação de permissões para publicação na rede social por parte do *Optipricer*, e com quem devem ser partilhadas essas publicações.



f Facebook

Inicia sessão para utilizares a tua conta do Facebook com a aplicação Software With Emotion

E-mail ou telefone:

Palavra-passe:

Manter sessão iniciada

[Esqueceste-te da tua palavra-passe?](#)

Iniciar sessão **Cancelar**

Figura 47 - Login no Facebook a partir da aplicação do Optipricer (SWE)



f Iniciar sessão com o Facebook



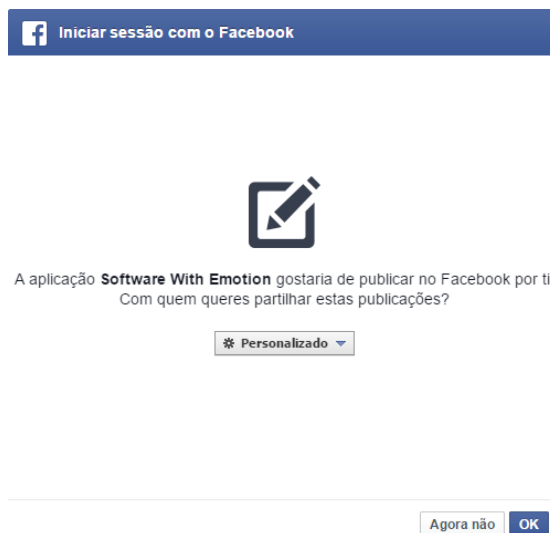
Software With Emotion vai receber as seguintes informações:
as tuas perfil público, endereço de e-mail, Publicações na cronologia, relações, data de nascimento, historial de trabalho, historial de formação, interesses e gostos.

[Editar a informação que indicas](#)


Isto não deixa a aplicação publicar no Facebook.

[Política de Privacidade](#) **Cancelar** **OK**

Figura 48 - Pedido de permissões do Facebook para acesso a dados do perfil, por parte da aplicação do Optipricer



f Iniciar sessão com o Facebook



A aplicação **Software With Emotion** gostaria de publicar no Facebook por ti.
Com quem queres partilhar estas publicações?

* Personalizado ▾

Agora não **OK**

Figura 49 - Pedido de permissões para partilha do produto na rede social

As permissões devem ser todas aceites para o processo de obtenção de desconto ter sucesso.

De seguida, o script em *JavaScript* é responsável pela junção dos dados (dados de autenticação do utilizador no *Facebook* e dados da loja e do produto) para serem enviados para o *Optipricer*, enquanto o processo se completa (através de uma chamada *AJAX*), como está representado na Figura 50.

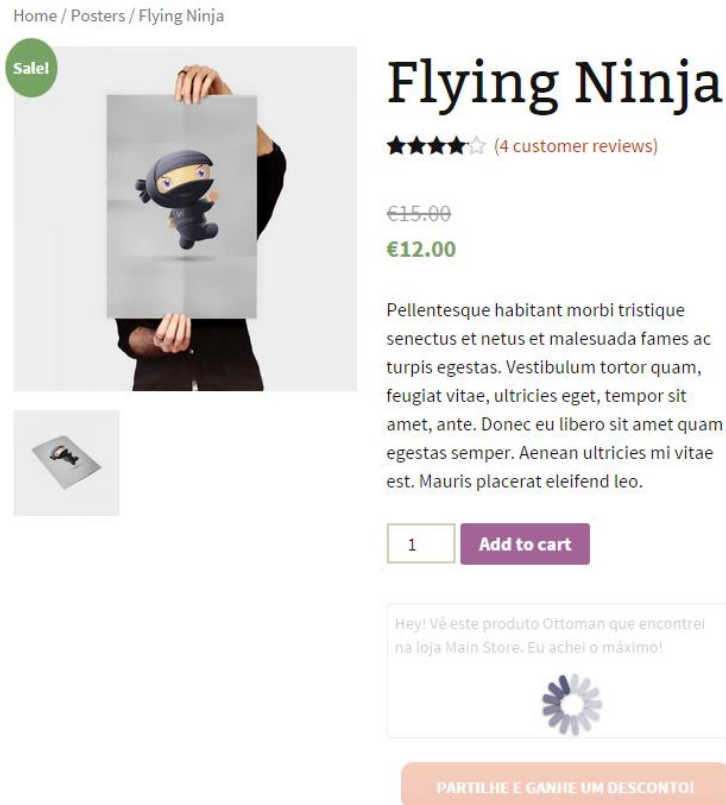
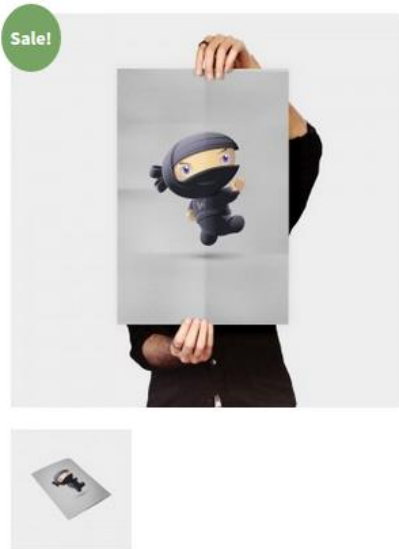


Figura 50 - Página de produto durante a obtenção de desconto

No processo supracitado, é invocada a chamada 6 – *createCoupon()* do diagrama do Fluxo de obtenção de desconto, tal como explicitado na secção 4.2.2. Assim, é consumida a API do *Optipricer* dessa comunicação de forma a obter um desconto de modo seguro remotamente. De seguida, após alguns segundos, a chamada 7 – *response()* é invocada pelo *Optipricer*, como resposta à chamada 6 – *createCoupon()* por parte da loja. O *response()* inclui o cupão de desconto a ser utilizado pelo cliente na loja. Este cupão é assinado pelo *Optipricer* utilizando criptografia assimétrica, ou seja, assina o cupão com a sua chave privada, para posteriormente ser validada pela loja, com a respetiva chave pública. Esta chave é incluída na extensão do *Optipricer*, neste caso, para a plataforma *Woocommerce*. Depois deste processo concluído, o preço é riscado apresentando um novo preço e a percentagem de desconto é mostrada ao utilizador no local onde estaria o espaço para o comentário e o botão de partilha. Este processo é feito a partir de uma biblioteca do *Optipricer*, em *JavaScript*. A Figura 51 mostra como se apresentaria o desconto obtido pelo *Optipricer*, na página do produto (destacados com retângulo vermelho). Na figura destacam-se o preço final (10,08€) e a percentagem de desconto obtido (16%). Nesta fase é consumido o fluxo 8 – *createCookie()* do diagrama de Obtenção de um desconto. Este fluxo visa criar uma *cookie* com os dados do desconto e a biblioteca de *JavaScript* é responsável por testar a sua existência, a cada *refresh* da página: se a *cookie* existir e for válida, o bloco da Figura 45 desaparece e passa a apresentar a percentagem de desconto obtido (“You got 16% discount”).

Home / Posters / Flying Ninja



Flying Ninja

★★★★☆ (4 customer reviews)

~~€15.00~~
€12.00
€ 10.08

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

1

Add to cart

You got a 16% discount

Category: [Posters](#).

Figura 51 - Desconto obtido pelo Optipricer

Do lado do *Optipricer*, optámos por criar uma assinatura com uma chave privada do conteúdo do cupão, visto que é uma forma de evitar rejeição por parte da gestão da loja (não é possível criar cupões na loja), garantindo que o cupão foi mesmo emitido pelo *Optipricer*. A razão para a não cifra dos dados é a possibilidade de utilização dos dados pela biblioteca *JavaScript*, por exemplo para apresentar o novo preço e ainda a não necessidade de privacidade de dados, visto que os mesmos não são sensíveis. Com o desconto obtido, as atenções centram-se na adição do produto ao carrinho com o respetivo desconto, o que será discutido na secção seguinte.

5.3 Fluxo de adição do produto ao carrinho com desconto


Nesta secção será discutida a implementação do fluxo de adição do produto ao carrinho com desconto. Assim, iremos criar um cupão na loja (se ainda não existir) e mapeá-lo com o cupão gerado pelo *Optipricer*, sendo que a descrição do cupão terá o conteúdo do cupão criado pelo *Optipricer*, devidamente assinado. Neste contexto, os dados da *cookie* são validados, com a respetiva assinatura, e se válidos o cupão é criado. Se o cupão da loja já existir, pode ser atualizado, acrescentando este conteúdo. Por exemplo, se um desconto foi obtido via outro dispositivo, por exemplo um *tablet* ou um *smartphone*, pelo mesmo utilizador, então na descrição do cupão da loja devem constar dois cupões do *Optipricer*, em que apenas muda o id de sessão (*ssid*). A Figura 52 ilustra o processo de adição de um produto ao carrinho:

Home / Posters / Flying Ninja

✓ Coupon code applied successfully.

✓ "Flying Ninja" was successfully added to your cart. [View Cart](#)

Sale!



Flying Ninja

★★★★☆ (4 customer reviews)

€15.00
~~€12.00~~
€ 10.08



Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

1 [Add to cart](#)

Figura 52 - Adição de um produto ao carrinho com desconto

Sublinhado, na Figura 52, destaca-se a ação de adicionar um cupão de desconto ao carrinho. Aquando do clique no botão “Add to cart”, o produto é adicionado ao carrinho, e após esta ação é despoletado um evento, que visa criar (ou ir buscar) um cupão na loja, aplicando-o de seguida no carrinho de compras. Esta ação é demonstrada pelo diagrama do Fluxo de adição de um produto ao carrinho com desconto, exposto no capítulo 4. (Figura 35).

Cart

	PRODUCT	PRICE	QUANTITY	TOTAL
	 Flying Ninja	€12.00	1	€12.00
Coupon code	<input type="text"/>	<input type="button" value="Apply Coupon"/>		<input type="button" value="Update Cart"/>

Cart Totals

SUBTOTAL	€12.00
COUPON:	-€1.92
OPTIPRICER_AVEIRO12345ABCD70112555CD07AB2925	[Remove]
SHIPPING	Free
	Shipping
	Calculate
	Shipping
TOTAL	€10.08
<input type="button" value="Proceed to Checkout"/>	

Figura 53 - Carrinho de compras que contém produto com desconto

Analisando a Figura 53, reparamos que o código do cupão se encontra no carrinho de compras e que o total corresponde ao preço do produto com desconto. O código do cupão corresponde à junção de “Optipricer_” com o código do cupão produzido pelo Optipricer (Coupon Token). O cupão tem utilização única e apenas é válido para o produto para o qual foi pedido o desconto. Este cupão não pode ser utilizado por outro indivíduo, ou seja, é possível adicionar o código do cupão ao carrinho mas no checkout as validações (ssid) falham. O valor do cupão (1,92€) é baseado no preço do produto, subtraindo-se ao preço original do produto o preço final, com desconto do mesmo (12,00€ - 10,08€). Seguidamente, na próxima secção, será analisado como se processa o fluxo de checkout do produto com desconto.

5.4 Fluxo de checkout de um produto com desconto

Neste subcapítulo iremos descrever como se comporta a aplicação aquando do checkout de um carrinho de compras que contém um produto com desconto, obtido através do *Optipricer*. Como referido no capítulo 4, neste fluxo irão ser feitas todas as validações a nível do cupão do *Optipricer*: verificação da assinatura com os dados do cupão (desconto e produto associado), análise da validade do cupão, validação do *ssid* do cupão, comparando-o com o *ssid* do utilizador que pretende usufruir do desconto. Se todas as validações estiverem corretas, a aplicação procede ao envio dos dados do cupão para o *Optipricer*, confirmando que o cupão foi utilizado.


Your order

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL	€12.00
COUPON: OPTIPRICER_AVEIRO12345ABCD70112555CD07AB2925	-€1.92 [Remove]
SHIPPING	Free Shipping
TOTAL	€10.08

Direct Bank Transfer

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Cheque Payment

PayPal  [What is PayPal?](#)

Place order

Figura 54 - Checkout de um produto com desconto

Na Figura 54, sublinhado com um tom forte, encontra-se o preço do produto, o valor do cupão (desconto) e por fim o preço final. No checkout do produto, podemos comprovar que o desconto se encontra corretamente calculado e apresentado, a partir de um cupão de desconto da loja. Ao clicar no botão “place order”, a aplicação fará a validação dos dados, a partir da biblioteca de segurança que implementámos. Se os dados se encontram corretamente validados, a aplicação procede ao envio do cupão ao *Optipricer* para confirmar que o cupão foi utilizado. Depois de todo o processo concluído, se tudo se processou de forma correta, deparamo-nos com a compra efetuada (Figura 55).

Nesta figura temos a confirmação como o desconto foi bem utilizado, e o preço do produto foi bem calculado. No *backoffice* do *Optipricer*, podemos ter a confirmação de que o cupão foi mesmo utilizado, a partir da Figura 56.

Thank you. Your order has been received.

ORDER NUMBER: 110	DATE: May 20, 2015	TOTAL: €10.08	PAYMENT METHOD: Direct Bank Transfer
-----------------------------	------------------------------	-------------------------	--

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Our Bank Details

Order Details

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL:	€12.00
DISCOUNT:	-€1.92
SHIPPING:	Free Shipping
PAYMENT METHOD:	Direct Bank Transfer
TOTAL:	€10.08

Figura 55 - Checkout concluído do produto com desconto

The screenshot shows the 'Coupons Activity' section of the Optipricer backoffice. The interface includes a navigation bar with 'Optipricer' logo and menu items: ACTIVITY, STORES, PRODUCTS, USERS, PROFILES, ANALYTICS, and COORDENADOR DE. Below the navigation bar is a search bar with 'Store' selected. The main content area displays three coupon activity entries for the 'Only-good-Stuff Store':

- Entry 1:** João Pires has a discount on product Flying Ninja with 16% discount. The original price is 12.00 €, and the discounted price is 10.08 €. The coupon is 'Activated'.
- Entry 2:** Pedro Amaral has a discount on product Shirt with 17% discount. The original price is 100.00 €, and the discounted price is 83 €. The coupon is 'Deactivated'.
- Entry 3:** Francisco Almeida has a discount on product Shirt with 13% discount. The original price is 100.00 €, and the discounted price is 87 €. The coupon is 'Activated'.

Figura 56 - Backoffice do Optipricer

Pela análise da Figura 55, verificamos que o checkout foi concluído com sucesso, e o desconto foi bem aplicado ao produto, verificando o preço final (*total*) e o desconto aplicado (*discount*). De acordo com a Figura 56, no *backoffice* do *Optipricer*, é possível visualizar todos os cupões obtidos, os que foram utilizados (*activated*, a verde), bem como os donos do cupão e a respetiva loja. Daqui concluímos que o cupão foi devidamente bem utilizado, e a comunicação com o servidor verificou-se, visto que o cupão se encontra ativo (*activated*).

5.5 Biblioteca de Segurança

Nesta secção, descrevemos a biblioteca de segurança que implementámos, que visa proteger os dados de cada entidade, com intuito de ser incluída num *plugin* para uma plataforma de *e-commerce*. Esta biblioteca é compatível com o *Optipricer*, sendo que inclui um método que obtém a chave pública do *Optipricer* a partir de um ficheiro. Esta mesma biblioteca foi aplicada ao *Magento* e ao *Woocommerce*. Em futuros *plugins* para outras plataformas, é possível integrar esta biblioteca de segurança.

Esta biblioteca contempla dois ficheiros: um para colocar numa entidade secundária, que apenas pode verificar assinaturas, cifrar e decifrar mensagens (a colocar do lado da loja), e um outro ficheiro a colocar num outro servidor, uma entidade principal, que terá mais regalias, além das mencionadas anteriormente, como assinar mensagens com a sua chave privada (a colocar no *Optipricer*).

Aplicado ao nosso projeto, do lado da loja, adicionámos o ficheiro que contempla as operações mais básicas: cifra e decifra de dados e verificação de assinaturas com a chave pública do *Optipricer*. Deste lado, também é possível um método conjunto, ou seja, verificar assinaturas e decifrar conteúdo assinado. Como adição ainda implementámos métodos para gerar e verificar um id de sessão, representativo do utilizador, que poderá depender do *user agent* do utilizador e do endereço *IP*. Além disto, contém também um id de sessão do PHP, que pode ser obtido através da função PHP *session_id()*.

Do lado do *Optipricer*, incluímos o ficheiro principal mais completo, visto este ser a entidade principal que disponibiliza o método para assinar dados com a sua chave privada e também um método composto, que contempla cifra com chave partilhada e assinatura com a sua chave privada.

No início do código são definidas várias constantes, que servem de parâmetros de configuração:

- `PRIVATE_KEY_FILENAME`: localização do ficheiro que contém a chave privada da entidade principal (neste caso o *Optipricer*, apenas presente na entidade principal);
- `PUBLIC_KEY_FILENAME`: localização do ficheiro que contém a chave pública da entidade principal;
- `CIPHER_ALG`: algoritmo de cifra, representado pelas constantes do *MCrypt*;
- `CIPHER_MODE`: modo de cifra, representado pelas constantes do *MCrypt*;
- `CIPHER_ALG_MODE`: algoritmo e modo de cifra conjugados; *string* que representa a chave para um *array* que mapeia os algoritmos e modos de cifra;

- `SIGN_ALG`: algoritmo de síntese a utilizar nas assinaturas. Deve ser confirmado com as chaves geradas.

São ainda definidas três constantes que ditam como devem ser protegidos ou analisados os dados pela biblioteca de segurança:

- `SECURE_CIPHER`: modo de proteção de dados por cifra
- `SECURE_SIGN`: modo de proteção de dados por assinatura
- `SECURE_CIPHER_SIGN`: modo de proteção de dados por assinatura e cifra

Esta biblioteca produz um objeto padrão que contempla as informações necessárias para recolher os dados enviados que foram previamente protegidos. Neste objeto podem estar presentes valores, como o algoritmo de cifra / assinatura utilizado, o IV, o *hmac*, entre outros.

Definimos dois métodos comuns genéricos: o primeiro protege o conteúdo de acordo com o pedido (cifrar, assinar ou ambos). O segundo método trata as mensagens e analisa qual o método que se adequa para obter o texto desprotegido, de acordo com o criado no primeiro método.

O primeiro método é definido pela função *secureContent*:

```

/**
 * Method to secure content (can use different security methods)
 */
public static function secureContent($task, $content, $key = false, $privKey
= false)

```

Figura 57 - Função *secureContent* da biblioteca de segurança

Os argumentos de entrada desta função são os seguintes:

- A *task* define qual a tarefa de segurança dos dados: se cifra, se assinatura ou se ambos, definindo a partir das constantes: `SECURE_CIPHER`, `SECURE_SIGN` e `SECURE_CIPHER_SIGN`. Do lado da loja apenas é possível a operação de cifra dos dados.
- O *content* define o conteúdo a ser assegurado.
- A *key* define a chave partilhada entre as duas entidades (neste caso, loja e *Optipricer*).
- A *privKey* define a chave privada do *Optipricer* para proceder à assinatura dos dados. Este campo não é utilizado do lado da entidade secundária (loja).

O conteúdo retornado será uma *string* em formato JSON ou em formato codificado com base 64. O objeto retornado por um conteúdo cifrado pode ser da seguinte forma, em formato JSON:

```

1  "data": {
2      "content": "Conteúdo cifrado(codificado em base64)",
3      "iv": "InitializationVector(codificado em base64)",
4      "hmac": "HMAC",
5      "alg": "Algoritmo utilizado na cifra"
6      "smode": "Modo de segurança (neste caso SECURE_CIPHER)"
7  }

```

Figura 58 - Dados retornados pela biblioteca de segurança no processo de cifra

No caso de conteúdo assinado, este deve conter o “sign”, retirando o “iv” e o “hmac” do objeto. Por fim, no caso das duas operações, estará presente o “sign” e o “iv”, mas o “hmac” não será necessário visto que a mensagem estará assinada. O “smode” deve estar de acordo com o modo de segurança utilizado.

Nestas funções, devem entrar como argumentos as chaves (partilhadas e/ou chave privada do *Optipricer*) e o conteúdo em *string*, em formato JSON se o conteúdo a proteger for um *array* ou objeto.

O método de obtenção de dados é definido pela função *getContent*:

```

/**
 * Get content of a secured data object
 */
public static function getContent($content, $key = false, $publicKey = false)

```

Figura 59 – Função *getContent* da biblioteca de segurança

Neste caso não existe a operação (*task*), a mesma vai contemplada no *content*, onde se encontra especificado como é que a mensagem foi protegida.

Os argumentos de entrada desta função são os seguintes:

- *content*, que representa o conteúdo protegido. Pode estar em objeto JSON ou ter ainda uma codificação de base 64 por cima. Este argumento de entrada deve ser o mesmo obtido pela função *secureContent*.
- *key*, que representa a chave partilhada entre as duas entidades, a loja e o *Optipricer*
- *publicKey*, que representa a chave pública do *Optipricer*. Este argumento, no *Optipricer*, apenas é utilizado para testes unitários.

Dependendo do definido pelo campo “smode”, presente no *content*, a função pode decifrar dados (SECURE_CIPHER), verificar assinatura (SECURE_SIGN) ou realizar ambas as operações (SECURE_CIPHER_SIGN). Se este campo não estiver conforme a mensagem irá ser retornado um erro.

Findo este capítulo sobre a implementação de cada fluxo e da biblioteca de segurança, iremos proceder, no capítulo seguinte, à apresentação dos resultados, bem como à sua análise e discussão, promovendo diferentes ambientes sobre a plataforma de e-commerce.

6. Avaliação e resultados

Neste capítulo iremos apresentar os resultados obtidos com a execução do protótipo enunciado na secção anterior. Como o fluxo mais importante e completo é o de obtenção de desconto, os testes funcionais e de carga focar-se-ão neste fluxo. Numa fase inicial, iremos apresentar os resultados alcançados através da nossa aplicação para a obtenção de um cupão, com e sem segurança, enunciando os dados enviados pela loja e recebidos pelo *Optipricer*. De seguida, introduziremos carga no processo de obtenção de cupão, bombardeando o servidor do *Optipricer* com pedidos. Faremos este processo come sema biblioteca de segurança, comparando os resultados e calculando o atraso introduzido pela mesma. Depois, validaremos o trabalho realizado, introduzindo erros no conteúdo seguro, testando assim o comportamento da aplicação. Por fim, apresentaremos também testes unitários que permitem validar o comportamento da biblioteca de segurança, para diferentes conteúdos.

6.1 Testes funcionais

Como referido anteriormente, esta secção apresentará testes da aplicação, exibindo resultados em termos de pacotes, que circulam durante o processo de obtenção de desconto, bem como os dados incluídos com e sem segurança.

As *headers* enviadas no pedido para criação de cupão estão presentes na Figura 60:

```
Request Method: POST
Request URI: /api/coupon/
Request Version: HTTP/1.1
Host: 10.0.3.141\r\n
Connection: keep-alive\r\n
▶Content-Length: 1227\r\n
Accept: application/json\r\n
Origin: http://lh.optipricer.com:4669\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36\r\n
Authorization: Token aveiro12345abcd\r\n
Content-Type: application/json\r\n
Referer: http://lh.optipricer.com:4669/?product=flying-ninja\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: pt-PT,pt;q=0.8,en-US;q=0.6,en;q=0.4\r\n
```

Figura 60 - Headers do método POST request api/coupon

Com base na figura, verificamos que devem ser incluídos no pedido os *headers*, *Authorization* (*token* da loja especificado), *Content-type* (*application/json*), e *accept* (*application/json* ou *application/html*), como analisámos na secção 4.2.2.

```

1  {
2    "product_id": 70,
3    "name": "Flying Ninja",
4    "link": "http://lh.optipricer.com:4669/?product=flying-ninja",
5    "description": "Pellentesque habitant morbi tristique senectus et netus et males(...)",
6    "image_url": "http://lh.optipricer.com:4669/wp-content/uploads/2013/06/poster_2_up-300x300.jpg",
7    "price": "12",
8    "product_brand": "",
9    "product_barcode": "",
10   "categories": [
11     "Posters"
12   ],
13   "min": "5",
14   "max": "30",
15   "location": "Portugal",
16   "text": "",
17   "swe_render": "0",
18   "expiry_offset": "1200",
19   "currency": "€",
20   "ssid": "52b13113ebb707000de3ba95773dedb742dfc089"
21 }

```

Figura 61 - Dados não protegidos pelo pedido createCoupon

```

1  {
2    "content": "SrTx5PORFBeziwPmPN1vvQu9payLHvUgiZJcXfrxda8xlrD590HDxeMo4/GbE4ha9V5BwdJ(...)"
3    "iv": "sKcsrFQuCLPDqoJLfbZf0Q==",
4    "hmac": "c3420b6bbfc4a99a47c4ec18d822dbf6abdafa691df87d1022da2a70ec3265208",
5    "alg": "AES-128-CBC",
6    "smode": 1
7  }

```

Figura 62 - Dados protegidos (cifrados) pelo pedido createCoupon

Na Figura 61, o *content* apresenta-se visível e manipulável para o utilizador, pois os dados não estão cifrados. Já na Figura 62, o conteúdo do campo *content* é inteligível para o utilizador, o que também o torna não manipulável por terceiros. Analisando as figuras, concluímos que o conteúdo que circula nas comunicações foi protegido com êxito. A mensagem cifrada apresenta-se em formato JSON conseguida a partir da biblioteca de segurança (5.5 - Biblioteca de Segurança), constituída por *contente* (conteúdo cifrado), *iv*, *hmac*, *alg* (algoritmo e modo de cifra por bloco) e *smode* (cifra). Como apresentámos na secção anterior, a mensagem apresenta-se protegida através da biblioteca de segurança, tendo um espaço para o conteúdo. Através de uma observação mais atenta das figuras, concluímos também que uma mensagem protegida (neste caso cifrada) ocupa muito mais espaço que uma mensagem em claro, cerca de mais de 700 bytes, o que se pode traduzir em atrasos na rede.

Na resposta, o *Optipricer* envia os dados do cupão visíveis pelo utilizador, mas com uma assinatura por baixo para validação dos mesmos. As figuras seguintes ilustram a comparação entre os dados enviados com e sem segurança:

```

1  {
2    "ssid": "52b13113ebb707000de3ba95773dedb742dfc089",
3    "expiryDate": "2015-07-20T14:03:38+0000",
4    "product": {
5      "id": "70",
6      "name": "Flying Ninja",
7      "price": "12.00"
8    },
9    "value": "9.84",
10   "token": "aveiro12345abcd7055abe67acbbdc"
11  }

```

Figura 63 - Dados não protegidos enviados na resposta do createCoupon pelo Optipricer

```

1  {
2    "content": '{
3      "ssid": "52b13113ebb707000de3ba95773dedb742dfc089",
4      "expiryDate": "2015-07-20T14: 03: 38+0000",
5      "product": '{
6        "id": "70",
7        "name": "FlyingNinja",
8        "price": "12.00"
9      }',
10     "value": 9.84,
11     "token": "aveiro12345abcd7055abe67acbbdc"
12   }',
13   "alg": 7,
14   "smode": 2,
15   "sign": "MCwCFGVMB391LCV2o4G53KVOOM27bSgLAhReeDg4aA07UqIISPooRZNYiWNeIA=="
16 }
17

```

Figura 64 - Dados protegidos (assinados) enviados na resposta do createCoupon pelo Optipricer

No caso da Figura 63, os dados circulam visíveis e não protegidos na rede, já na Figura 64, os dados embora circulem visíveis na rede, incluem uma assinatura que permite validar os mesmos. Estes dados traduzem-se em não privados, tanto para o *Optipricer* como para a loja, sendo apenas necessário assiná-los dando-lhes autenticidade. A assinatura deve ser emitida via chave privada do *Optipricer* e validada posteriormente pela loja, com a respetiva chave pública. Através de uma análise mais fina das figuras, concluímos que a inserção de uma assinatura sobre os dados influencia muito pouco o tamanho do pacote, cerca de 180 bytes, sendo uma mais-valia a utilização de assinaturas. Uma das grandes vantagens da sua utilização é ocuparem pouco espaço, o que se traduz em *cookies* mais pequenas na nossa aplicação.

6.2 Testes de carga

Nesta secção serão apresentados os resultados dos testes de carga efetuados sobre a aplicação do *Optipricer*. Para tal, as comunicações a partir do *Optipricer* com o *Facebook* foram desprezadas, visto que os testes apenas se centram na obtenção dos dados, criação de cupão e retorno do mesmo (com ou sem segurança). Assim considerámos quatro cenários primários de segurança, que serão conjugados com dois cenários de criação de cupão:

- 1 – Com segurança, o processo inclui decifra de pedidos e assinatura de cupões
- 2 – Com segurança parcial (só loja), o processo inclui apenas decifra de pedidos

- 3 – Com segurança parcial (só *Optiprizer*), o processo inclui apenas assinatura de cupões
- 4 – Sem segurança, o processo é realizado num todo sem segurança

- A – Utilização de um mesmo cupão para todas as comunicações
- B – Criação de um cupão diferente para cada comunicação

Deste modo, temos presentes 8 cenários: A1, A2, A3, A4; B1, B2, B3, B4. Estes testes foram corridos sobre o programa AB do apache, com 1000 pedidos constantes, alternando o número de pedidos concorrentes (de 20 em 20, até 260). Os testes de carga foram efetuados sobre o *Optiprizer*, que corria numa máquina virtual lançada pelo LXC, com 1700 MB de RAM como parâmetro de inicialização, num Notebook, com um CPU Intel Core i5-3317U CPU 1.70GHz, com 6GB de RAM, no sistema operativo *Ubuntu* 14.10. Para determinar a velocidade de serviço para cada cenário, só são considerados até 160 pedidos concorrentes (inclusive), visto que a partir destes, a aplicação começa a retornar erros do tipo HTTP 500, o que acelera o comportamento da mesma, não correspondendo à realidade. As tabelas dos resultados podem ser consultadas no apêndice C.

O gráfico seguinte demonstra o comportamento da aplicação para cada um dos cenários mencionados, apresentando o número de pedidos por segundo suportados pela aplicação.

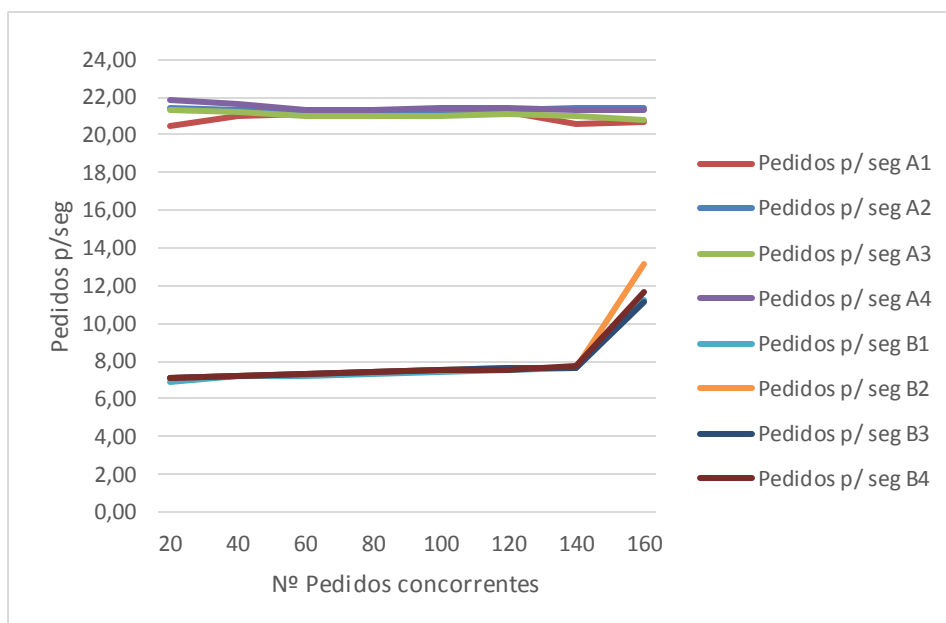


Figura 65 - Comportamento da aplicação para os diferentes cenários

Analisando o gráfico obtido através dos testes de carga efetuados sobre os diferentes cenários (Figura 65), concluímos que, no geral, o cenário A é bem mais rápido do que o cenário B, o que já era expectável. O processo de criação de cupão inflaciona drasticamente a capacidade do servidor, uma vez que além da pesquisa na base de dados por um cupão existente, é necessário o recálculo do desconto, que é um processo exigente, e ainda a inserção do respetivo cupão na base de dados. As linhas do gráfico mais acima representam os testes

efetuados sobre o cenário A, onde o servidor consegue suportar mais pedidos por segundo, do que no cenário B, representado pelas linhas mais abaixo. Para diversos testes ao cenário A, o número de pedidos que o servidor serve por segundo varia entre 20 e 22, no entanto, para o cenário B, o servidor apenas é capaz de responder, em média, a uma taxa de 7 pedidos por segundo.

Os gráficos seguintes fazem um “zoom” sobre a performance do servidor no cenário A (Figura 66) e cenário B (Figura 67). A partir destes gráficos, é possível analisar cada cenário secundário e as diferenças de performance do utilizador, quando utilizada segurança parcial, segurança completa sobre os pedidos, ou quando estes não são assegurados. É de notar que estes gráficos utilizam escalas diferentes das habituais, de modo a podermos analisar cuidadosamente o comportamento do servidor em diferentes cenários. A escala utilizada no gráfico da Figura 66 situa-se entre 20,40 e 22,00 pedidos por segundo, já no gráfico da Figura 67, a escala utilizada encontra-se entre os 6,80 e os 7,80 pedidos por segundo.

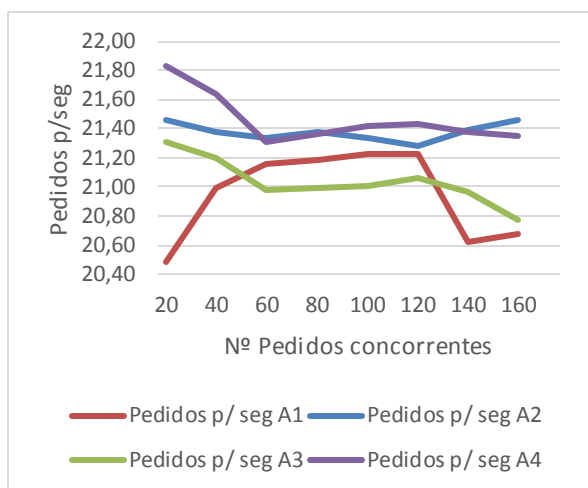


Figura 66 - Gráfico de testes de carga (pedidos p/seg) para os cenários A1, A2, A3 e A4,

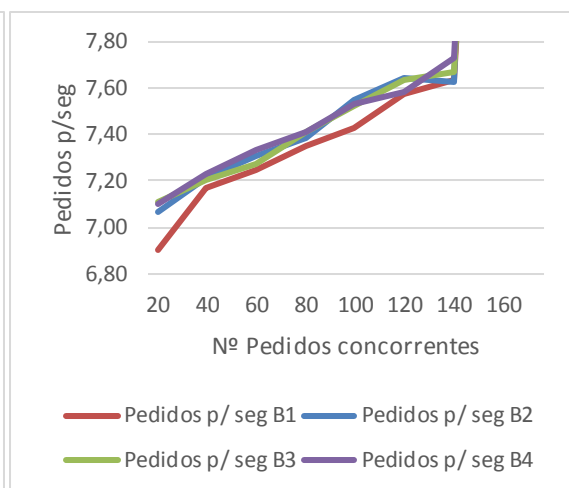


Figura 67 - Gráfico de testes de carga (pedidos p/seg) para os cenários B1, B2, B3 e B4

O primeiro cenário (cenário A) é menos realista, sendo sempre utilizado um mesmo cupão para todos os pedidos, mas serve para analisar de uma forma facilitada o comportamento da nossa biblioteca de segurança, quando é introduzida carga sobre o servidor. Deste modo, analisando o gráfico da Figura 66, a aplicação tem maior performance a servir pedidos cujos dados se encontram sem segurança (cenário A4) do que pedidos que contêm mensagens seguras a partir da nossa classe, o que já seria de esperar (cenário A1). No entanto, com a nossa camada de segurança completamente implementada, os valores de pedidos por segundo suportados pelo servidor diferem em apenas 1 pedido, no máximo e no mínimo cerca de 0,20. Pelo gráfico, concluímos que o auge da aplicação ocorre nos 120 pedidos concorrentes, sendo que com a nossa camada de segurança, o servidor consegue servir cerca de 21,20 pedidos por segundo. Com a análise do gráfico, ainda conseguimos concluir que o processo que mais inflaciona os tempos de serviço do servidor é o de geração de assinaturas digitais

(A3), sendo que o cenário de decifra de mensagens e envio de cupões desprotegidos tem uma boa performance, conseguindo no seu auge atingir os 21,40 pedidos por segundo, servidos pelo *Optiprizer*.

O segundo cenário (cenário B) é mais realista, pois é mais provável que a aplicação esteja a ser bombardeada com utilizadores diferentes, criando um cupão para cada utilizador, do que um mesmo utilizador requerer múltiplas vezes o mesmo cupão. Assim, analisando o gráfico correspondente a este cenário (Figura 67), verificamos que a inclusão da camada de segurança pouco influencia os tempos de serviço do servidor, sendo que, no máximo, o número de pedidos por segundo que o servidor suporta, nos cenários B1 e B4, apenas difere em cerca de 0,2. Como no caso anterior, com o aumento dos números de pedidos concorrentes, até 140, o número de pedidos que o *Optiprizer* consegue atender por segundo também aumenta. Com a nossa camada de segurança completamente implementada, o servidor consegue suportar, no máximo, cerca de 7,60 pedidos por segundo, para 140 pedidos concorrentes.

Concluído o estudo sobre o suporte do servidor para cada cenário, iremos apresentar os pacotes retornados pelo servidor com um erro 500, em que os pedidos não conseguem ser atendidos pelo servidor. Os gráficos seguintes demonstram o comportamento da aplicação para cada um dos cenários mencionados, apresentando o número de pedidos que não foram atendidos pela aplicação. O gráfico do lado esquerdo corresponde aos cenários de utilização de um mesmo cupão para cada pedido, e o do lado direito corresponde ao cenário de criação de um novo cupão para cada pedido, variando os cenários de segurança:

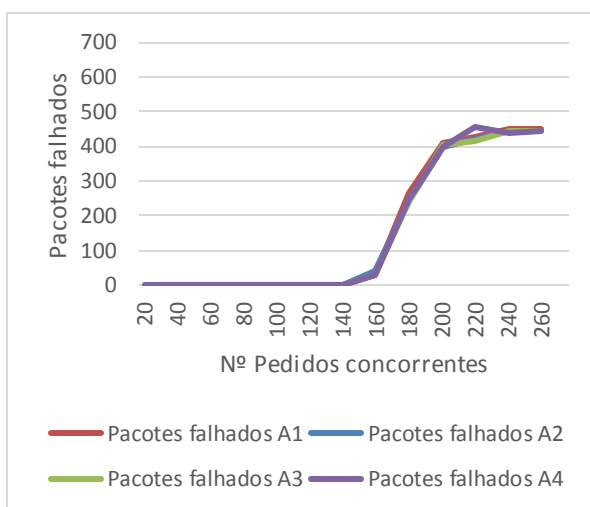


Figura 68 - Pedidos falhados (erro 500) para testes de carga com os cenários A1, A2, A3 e A4

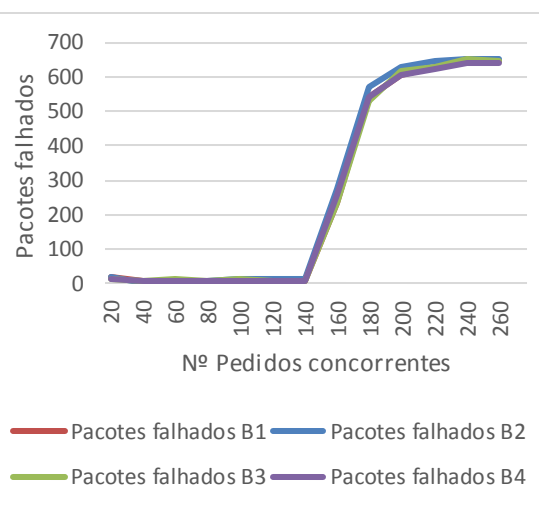


Figura 69 - Pedidos falhados (erro 500) para testes de carga com os cenários B1, B2, B3 e B4

Numa análise geral aos dois gráficos (Figura 68 e Figura 69), a inserção da camada de segurança pouco ou nada influencia o comportamento da aplicação, no suporte ao erro, já que para os quatro cenários de segurança o número de pedidos que não conseguem ser servidos pelo *Optiprizer* pouco difere. Comparando os dois cenários, o cenário A apresenta uma melhor qualidade de serviço, com uma percentagem de pedidos não atendidos tendendo para os 45%. Já no cenário B, visto que o servidor demora mais tempo a processar cada

pedido, a sua qualidade de serviço é bastante menor, sendo que a percentagem de pedidos não atendidos tende para os 65%. De uma forma geral, o número de pedidos falhados cresce exponencialmente a partir dos 160 pedidos concorrentes para o cenário A e 140 pedidos concorrentes para o cenário B.

Concluído o estudo sobre o comportamento da aplicação quando introduzida carga, com e sem a nossa camada de segurança, procederemos à análise da extensão quando são introduzidos erros ou ataques sobre os dados que circulam na mesma.

6.3 Análises de ataques / erros

Nesta secção será apresentado o comportamento da aplicação aquando da introdução de erros ou ataques. Assim, os dados para a criação do cupão irão ser manipulados de forma a analisar o comportamento da aplicação no tratamento de erros ou ataques. Da mesma forma, irão ser manipulados também os dados do cupão criado pelo *Optipricer*, analisando o comportamento da aplicação com a introdução destes erros. Por fim, iremos remover a *cookie* de sessão de modo a simular a utilização de um cupão por um outro utilizador que não o dono do mesmo.

O primeiro ataque a ser realizado centra-se na manipulação do conteúdo a ser enviado para criação do cupão via *JavaScript*. O conteúdo será manipulado na consola de *JavaScript* do *browser*:

```
cp.d
"
{"content":"SCD583NdUt\AXfkubw8MLbL14ycczaVB5phmcQwDhCKJz2foSjWrl0Pnng4PotF++Ikb3KsJlzewgmjJ9HrxaIXBciaeJaFWSQkkBoZ\oXEL5oeqg9Ythf19raxRHBP6\CPZajzRkvj9PpQEjwP0ForZZEBMN
BpL+07d09U25cTqZ\U239vJyyAbuo1IGtpeqz7Lm0S\cLpm8MRT4e3t3gF00mJ77q0aY0us\0oapKRJ20MJkwWtFRAXstCaAZH96FWCLDXEc30T16NC78KVDmPgGP73u1HLCJVG2xvyfffljJ6jW4Pb6ne8tMJBn8mY0KkWS5UpI
iixAEuT5g0RH15jpp0mFpeIHfP5mmMpCwGLDbQH8CRdEzP\oYPR7v5crRYDuE4k7xtjB5ShIv2y8pIXdH00GCoicTRZ8TgXH9TLtS4uT01Z1sUfXmMqWkLYDvuvbR94F5e2AUvjBSKkMgnHM9d0qWkGL9bVA0ft\QrT\nsGuTYB8
bRePulmT0eUgWtmdk+gBkKBYj0vemK3rtvR+V\ENxqItx488WaiwJ1BL0F+fhX1UA2ZtC81Rt6n7EndTUVuYt6sXHCroh2R5ngn7F\65y+dbkjQ025rg+61yb+gb4Ra4Bko1nJ3KH6o5gu5Aa3048pP9ZwMknIOXgIecubV1jR45kFC
zIRpV/+8LXaoT6Gix0We40WgLU0709foYtfa1M661q881hyXtn\ALQX3\GtH5vLZ0ky+FvhcqtWk321YpUBLa6Rykyd7rBlIS8\Xnt1F0tdCyR7y+7KZ7qwPX64w2+0bt+0Ay67Dv0815NRoYgotvzX\ZXt70hXf+hbHLASct664nd
EKzvs61Qt4mV\5durefncj+3d1aB4L2ZjZDmnbdkVLC805skHFwJZw1nb0mZzULAs5XaapVomUXYEkawE0d+ffFL7BPvKc3a5x23+s6aKwUu74M0wh1Sub050tqwS\amy1N4-", "iv":"2c8W7F1gM0EmIbly0UsJ5Q==",
" hmac":"a381986a33b6257ca4ce663c65f320fdcd31242c94479d7c215984dbb4070dc", "alg":"AES-128-CBC", "smode":1}
cp.d = '{"content":"Manipulado", "iv":"2c8W7F1gM0EmIbly0UsJ5Q==", "hmac":"a381986a33b6257ca4ce663c65f320fdcd31242c94479d7c215984dbb4070dc", "alg":"AES-128-CBC", "smode":1}'
{"content":"Manipulado", "iv":"2c8W7F1gM0EmIbly0UsJ5Q==", "hmac":"a381986a33b6257ca4ce663c65f320fdcd31242c94479d7c215984dbb4070dc", "alg":"AES-128-CBC", "smode":1}
```

Figura 70 - Conteúdo para criar cupão manipulado via JavaScript

Pela Figura 70, verificamos que o conteúdo foi adulterado, tendo sido alterados os dados de “content” para “Manipulado”. Desta forma, aquando da criação do cupão, o resultado obtido é o seguinte:

Home / Posters / Flying Ninja

Sale!



Flying Ninja

★★★★☆ (4 customer reviews)

€15.00
€12.00

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

1 [Add to cart](#)

Houve um erro interno, por favor tente outra vez mais tarde.

Figura 71 - Resultado retornado pelo Optipricer quando o conteúdo para criação de cupão é manipulado

Na Figura 71, ao ter clicado no botão “Partilha e ganha um desconto”, que remete os dados de criação de desconto para o *Optipricer*, este responde com um erro, visto que não conseguiu decifrar o conteúdo da mensagem. Desta forma, garantimos a autenticidade e privacidade dos dados, visto que ao serem manipulados, a biblioteca de segurança comporta-se como devia, ou seja, procede ao retorno de erro. Se considerarmos uma aplicação insegura, ainda é possível adulterar o cupão depois de este ser retornado pelo *Optipricer*, via *JavaScript* ou alterando o valor da *cookie*.



Name	Value	Domain	Path	Expires / Max-Age
PHPSESSID	qt460keb8fq45dfj6850g321p0	lh.optipric...	/	Session
__utma	144571627.60647640.1430911187.1430988458.1431077459.3	.optipricer...	/	2017-05-07T09:30:58.000Z
__utmz	144571627.1430911187.1.1.utmcscr=(direct) utmccn=(direct) utmcmd=(none)	.optipricer...	/	2015-11-06T21:30:58.000Z
swe_aveiro12345abcd_70	eyJb250ZW50joie1wic3NpZw0lwiMDkzZDUxZjE4MzU0NWUwMDQ3NTp1ZmVknzRkOTEz...	lh.optipric...	/	2015-05-20T22:50:39.000Z

Figura 72 - Adição de um produto ao carrinho com a cookie manipulada (1)

A Figura 72 mostra que a *cookie* foi manipulada pelo utilizador e a Figura 73 ilustra o que acontece quando o cliente clica no botão “Add to cart” para proceder à adição do produto com desconto ao carrinho de compras:




Figura 73 - Adição de um produto ao carrinho com a cookie manipulada (2)

Embora a aplicação não mostre qualquer *feedback* (Figura 73), esta comporta-se como devia. Ao adicionar um produto ao carrinho, o produto apenas é adicionado e o cupão de desconto não é aplicado no carrinho, como verificámos na secção 5.3 (Figura 52). Visto que a *cookie* foi manipulada, a aplicação procede à sua remoção

e, portanto, o cupão não chega a ser criado, nem adicionado ao carrinho. A *cookie* pode ser removida por outros motivos, aquando da adição do produto correspondente ao carrinho: por o utilizador não ser o detentor do desconto, por o cupão de desconto ter expirado entretanto ou por o desconto não ter sido obtido para aquele produto (através da manipulação do nome da *cookie*). Embora a *cookie* contenha no nome o ID do produto com o desconto associado, este apenas é utilizado para a procura desse mesmo produto, sendo que o ID do produto presente no corpo da *cookie* é que é realmente aproveitado para validações, testando se corresponde realmente ao ID do produto que o utilizador está a adicionar ao carrinho. Portanto, no processo de adição de um produto ao carrinho, existe uma validação por parte da biblioteca de segurança, que assegura que os dados não foram manipulados. Concluimos que, apesar da falta de feedback da aplicação, esta comporta-se como devido, procedendo à remoção da *cookie* correspondente a um desconto caso algo irregular se verifique.

Finda esta análise, concluimos que durante o fluxo de obtenção de desconto não é possível adulterá-lo, nem antes deste ser obtido nem depois, aquando da adição do produto correspondente ao carrinho de compras. Ainda assim, é necessário estudar o que ocorre no fluxo de checkout do produto, que ataques é possível efetuar numa aplicação insegura. O primeiro que nos ocorre, e o de mais fácil aplicação, surge quando um utilizador partilha o código de desconto obtido com outro utilizador. Visto que a extensão do *Optipricer* do *Woocommerce* utiliza cupões da loja, é possível partilhá-los com outros utilizadores. No entanto, seria expectável que o *plugin* não o permitisse. A Figura 74 e a Figura 75 mostram como a aplicação se comporta quando um utilizador tenta aplicar um cupão do *Optipricer* que não lhe pertence, ao seu carrinho de compras:

Checkout

 **Cupão de desconto** Redeem failed: Este cupão não lhe pertence. Cupão removido do carrinho

Have a coupon? [Click here to enter your code](#)

Billing Details

Country *

Ship to a different address?

Order Notes

Figura 74 - Checkout de um produto com um cupão de desconto que não pertence ao utilizador (1)

Your order

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL	€12.00
SHIPPING	Free Shipping
TOTAL	€12.00

Direct Bank Transfer

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Cheque Payment


PayPal  [What is PayPal?](#)

Figura 75 - Checkout de um produto com um cupão de desconto que não pertence ao utilizador (2)

Quando um utilizador tenta fazer checkout de um produto com um cupão de desconto que não lhe pertence é deparado com o erro presente na Figura 74, “Este cupão não lhe pertence (...)”. Posto isto, a aplicação procede à remoção do respetivo cupão (Figura 75), e se existirem *cookies* que referenciem o cupão de desconto, estas também são removidas. Para garantir que a aplicação se comporta como expetável, é criado um id de sessão para o utilizador e que circula na comunicação com o *Optipricer*, aquando da obtenção de desconto: a loja cifra o identificador (id) de sessão e envia-o para o *Optipricer*, este retira-o da mensagem e coloca-o na mensagem a enviar, com o cupão devidamente assinado. Assim, o cupão de desconto da loja contém um id de sessão do utilizador dono do cupão. Este id é calculado através de uma *cookie* de sessão (PHPSSID) e através do *user-agent* do utilizador. Mesmo assim, é possível ocorrerem outros erros aquando do checkout do produto com desconto por parte do utilizador. Mesmo que o utilizador tenha efetuado todos os passos de forma correta, é possível ocorrerem diversos erros no checkout do produto. Por exemplo, quando um administrador ou empregado da loja manipula o cupão desta para prejudicar a loja e/ou o *Optipricer*, é expectável que a aplicação retorne um erro e não deixe o utilizador concluir a compra. A figura seguinte representa o cupão manipulado pelo administrador da loja, presente no carrinho de compras do utilizador:

Your order

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL	€12.00
COUPON: OPTIPRICER_AVEIRO12345ABCD70114555CE58F73951	-€5.00 [Remove]
SHIPPING	Free Shipping
TOTAL	€7.00

Direct Bank Transfer

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Cheque Payment



PayPal  [What is PayPal?](#)

Figura 76 - Cupão manipulado pela loja (processo de checkout)

A Figura 76 mostra que o cupão foi manipulado pela loja. O valor original era de 1,92€, como apresentámos na secção 5.4. Com a manipulação do valor do cupão da loja, este passou a ser 5,00€ (valor do desconto), sendo que o preço final resultou em 7,00€ (em vez de 10,08€). Assim, se o processo continuar e se o cliente conseguir fazer checkout com sucesso, este é beneficiado por um desconto maior do que aquele que adquiriu (a que tinha

direito) mas a loja é prejudicada, podendo culpabilizar o *Optipricer*. Ora, se o *Optipricer* providenciar uma prova como o desconto foi mesmo emitido por ele, como por exemplo uma assinatura sobre os dados, e esta estiver correta, é legítimo culpabilizar o *Optipricer* caso haja manipulação de preços. Seguindo esta abordagem, apenas o *Optipricer* poderia emitir provas de cupões emitidos, com os respetivos valores, cabendo à loja validá-los com a respetiva prova. As figuras seguintes representam o comportamento da aplicação quando o cupão é manipulado por um administrador da loja:

Checkout

 **Cupão de desconto** Redeem failed: Cupão Manipulado! Cupão removido do carrinho

Have a coupon? [Click here to enter your code](#)

Billing Details

Country *
Portugal

Ship to a different address?

Order Notes

Figura 77 - Checkout de um cupão manipulado pelo administrador da loja (1)


Your order

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL	€12.00
SHIPPING	Free Shipping
TOTAL	€12.00

Direct Bank Transfer

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Cheque Payment

PayPal  [What is PayPal?](#)

Place order

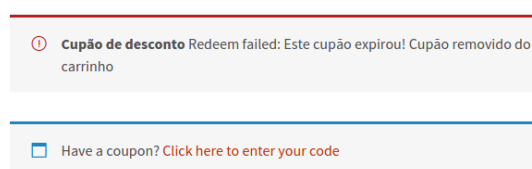
Figura 78 - Checkout de um cupão manipulado pelo administrador da loja (2)

Após o utilizador proceder à conclusão da compra (clicando no botão “Place order”), o processo não continua, ficando preso no ecrã de checkout novamente. A Figura 77 demonstra que ocorreu um erro aquando da tentativa de conclusão da compra: “Cupão manipulado! (...)”. A aplicação comporta-se como deve e procede à remoção do cupão em causa, como se pode verificar pela Figura 78, onde o cupão já não está presente na compra do utilizador. Embora o cliente não tenha culpa do sucedido, a aplicação deve-se proteger contra qualquer fraude, ou seja, deve impedir que qualquer processo irregular progrida. Se a aplicação deixasse prosseguir com a compra, iria prejudicar a loja, dando um desconto muito maior ao utilizador do que o realmente obtido, e esta poderia culpabilizar o *Optipricer* pelo sucedido. O método utilizado, que visa impedir que este tipo de irregularidades aconteça, são as conhecidas assinaturas digitais, em que cada cupão é assinado e, por fim, validado pela loja. A loja não consegue emitir assinaturas sobre os cupões, visto que apenas o *Optipricer* pode proceder à emissão de cupões e é responsável por eles. O processo de validação neste fluxo é o seguinte: inicialmente é validada a assinatura do cupão com os dados presentes na descrição do cupão. Se a assinatura estiver correta, avança-se para a próxima fase que consiste na validação do valor do desconto do cupão com o valor identificado na descrição do cupão. Estes dois passos são essenciais, pois se houve tentativa de

manipulação de cupões por um indivíduo responsável pela administração da loja *online*, esta deve ser travada. Se a assinatura for forjada ou os valores não corresponderem ao devido, então a aplicação procede à remoção do cupão do carrinho de compras, alertando o utilizador.

Outro erro que poderá ocorrer nesta fase é o cupão de desconto expirar entretanto. As figuras seguintes ilustram o comportamento da aplicação quando o prazo de validade do cupão expira durante o processo de checkout:

Checkout



❌ **Cupão de desconto** Redeem failed: Este cupão expirou! Cupão removido do carrinho

Have a coupon? [Click here to enter your code](#)

Billing Details

Country *

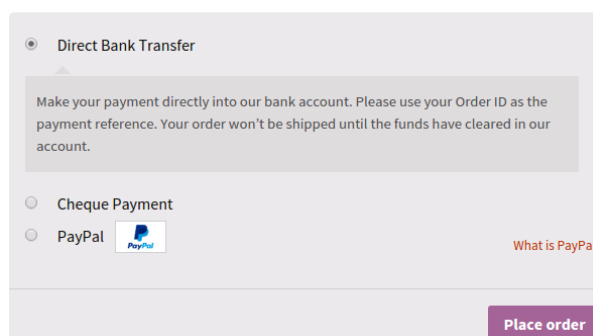
Ship to a different address?

Order Notes

Figura 79 - Checkout com cupão expirado (1)

Your order

PRODUCT	TOTAL
Flying Ninja × 1	€12.00
SUBTOTAL	€12.00
SHIPPING	Free Shipping
TOTAL	€12.00



Direct Bank Transfer

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order won't be shipped until the funds have cleared in our account.

Cheque Payment


PayPal  [What is PayPal?](#)

Figura 80 - Checkout com cupão expirado (2)

Após o cliente tentar concluir o processo de checkout, depara-se com um erro que lhe indica que o cupão expirou (Figura 79). Aqui, é demonstrado que a aplicação se comporta como é expectável, apresentando um erro ao cliente, “Este cupão expirou! (...)” e procedendo à remoção do cupão de desconto do carrinho (Figura 80). Além das validações anteriormente mencionadas, a extensão garante que o cupão ainda é utilizável (não expirou) validando o tempo atual com o tempo de expiração do cupão de desconto, após a validação da assinatura dos dados. Caso exista alguma *cookie* que esteja mapeada com o desconto, esta também é removida, de modo a permitir ao utilizador obter um novo desconto. De igual modo, se o cupão expirou ou foi manipulado, este é colocado como inutilizável, se o utilizador tentar usufruir desse mesmo cupão (aplicação direta do código do mesmo no carrinho de compras, por exemplo), a própria plataforma retornará um erro avisando o utilizador que o cupão já não é utilizável.

Em suma, a aplicação superou todos os requisitos funcionais, comportando-se à altura dos problemas. Verificámos que os dados que circulam na rede se encontram protegidos, e que no caso dos dados para a criação de cupão, estes mostram-se ilegíveis para o utilizador, garantindo a privacidade dos mesmos. Quanto à

introdução de erros e ataques, verificámos que a nossa extensão supera as expetativas, detetando, resolvendo e reportando cada erro que ocorre.

6.4 Testes unitários

Nesta secção iremos descrever os testes unitários realizados para testar a fiabilidade da nossa biblioteca de segurança, de modo a validá-la corretamente. Estes testes são efetuados sobre cada função da biblioteca, testando cada condição, de forma a analisar o comportamento da mesma, e no fim são comparados os valores retornados por cada função com os valores esperados.

Os testes unitários executados sobre a biblioteca vieram comprovar que esta se comporta como previsto, ou seja, os resultados retornados por ela, para cada teste, são os esperados. De seguida, iremos apresentar exemplos dos testes executados na nossa biblioteca, o resultado esperado e o resultado obtido.

Teste	Objetivo do teste	Resultado esperado	Resultado obtido
Secure Content With Invalid Task	Analisar o comportamento da função “secureContent()” com uma <i>task</i> inválida	<i>false</i>	<i>false</i>
Secure Content For Cipher Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de cifra, comparando o valor retornado no “smode”	<i>SECURE_CIPHER</i>	<i>SECURE_CIPHER</i>
Secure Content For CipherSign Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de cifra e assinatura (conjunta), comparando o valor retornado no “smode”	SECURE_CIPHER_SIGN	SECURE_CIPHER_SIGN
Secure Content For Sign Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de assinatura, comparando o valor retornado no “smode”	SECURE_SIGN	SECURE_SIGN

Figura 81 - Testes unitários efetuados sobre a nossa plataforma

Pela Figura 81, podemos observar que os resultados dos exemplos de testes foram os esperados, e a nossa camada de segurança comportou-se de forma adequada. A figura comprova que os testes são executados ao ínfimo pormenor, ou seja, neste caso, testam um argumento de entrada da função, a *task* de segurança, que poderá ser de cifra, de assinatura ou conjunta. Os restantes testes unitários efetuados sobre a nossa plataforma podem ser encontrados no apêndice D.

7. Síntese conclusiva

Este projeto focou-se na segurança de uma aplicação para o e-commerce, o *Optipricer*. Esta aplicação disponibiliza descontos de acordo com o perfil social de um sujeito, para um determinado produto, em troca da partilha do mesmo nas redes sociais e disponibilização de informações do seu perfil social (interesses, amigos, relações, entre outros). Com o objetivo de tornar seguras todas as comunicações entre o sujeito, a loja *online* e o *Optipricer*, surgiu este projeto. Este trabalho foca-se na garantia de segurança do utilizador, da loja e do *Optipricer* aquando da obtenção de um desconto e da aplicação do mesmo. Assim, é necessário proteger as comunicações da loja com o *Optipricer* que, entretanto, se cruzam com o cliente, a partir do seu *browser*, e poderão cruzar-se com terceiros.

Neste trabalho analisámos como se comporta a implementação de uma camada de segurança sobre o *Optipricer* e sobre a loja *online*, tendo mostrado resultados do comportamento de cada entidade com e sem a camada de segurança adicional, introduzindo carga sobre as comunicações. Do mesmo modo, demonstrámos resultados funcionais da biblioteca de segurança nas comunicações entre a loja e o cliente, fazendo a comparação dos dados que circulam nas comunicações com e sem segurança. Assim, com os estudos realizados e os testes cumpridos, concluímos que a biblioteca implementada cumpre os requisitos propostos, protegendo os dados e evitando a manipulação dos mesmos.

Com o objetivo de testar corretamente a aplicação que desenvolvemos, introduzimos carga sobre o servidor, em diferentes cenários, de modo a determinar o atraso temporal introduzido pela nossa biblioteca de segurança. Desta forma, testámos o *Optipricer* com segurança implementada e sem segurança e averiguámos que o comportamento da nossa aplicação é bastante positivo: a implementação da mesma sobre o *Optipricer* pouco influencia a performance do mesmo, servindo uma taxa de pedidos por segundo praticamente igual com e sem segurança implementada, sendo que com a nossa camada de segurança implementada, a taxa de resposta do servidor decresce, no máximo, apenas um pacote por segundo.

Por fim, como sugestões para trabalhos futuros versando este mesmo tema, dada a sua relevância, pensamos que seria de eleger como objeto de estudo a análise da implementação de uma API para trocas de chaves entre o *Optipricer* e a loja *online*, de forma segura. Com efeito, trata-se de um tema que viria na linha do projeto que desenvolvemos, completando-o e enriquecendo-o. Deixamos, assim esta sugestão para trabalhos futuros neste âmbito. A partir do trabalho por nós já desenvolvido, ter-se-ia que implementar um mecanismo de geração de novos pares de chaves para o *Optipricer*. Para armazenamento de chaves, iria ser necessário recorrer a uma base de dados que mapeasse um id para o par de chaves e o respetivo par de chaves. Do lado da loja, a extensão teria que contemplar o id da chave pública do *Optipricer* e a respetiva chave.

O projeto que desenvolvemos, como atrás explicitado, veio possibilitar a segurança sobre os serviços do *Optipricer*. Uma API de troca de chaves viria trazer ao serviço um melhor desempenho na segurança do produto, não comprometendo as chaves de segurança. Trata-se de uma ideia a explorar em futuros trabalhos!

8. Bibliografia

- [1] R. F. Munir, N. Ahmed, A. Razzaq, A. Hur, and F. Ahmad, "Detect HTTP specification attacks using ontology," *Proc. - 2011 9th Int. Conf. Front. Inf. Technol. FIT 2011*, pp. 75–78, 2011.
- [2] S. J. Minutillo, "Cookieless HTTP Session Persistence Using the BASE Tag," *20th Comput. Sci. Semin.*, pp. 1–6, 2004.
- [3] A. X. Liu, J. M. Kovacs, C. T. Huang, and M. G. Gouda, "A secure cookie protocol," *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, vol. 2005, pp. 333–338, 2005.
- [4] B. Nicolae and F. Academy, "Considerations on e-commerce platforms," vol. 1, no. March, pp. 27–29, 2014.
- [5] R. C. Marchany, V. Tech, J. G. Tront, and V. Tech, "E-Commerce Security Issues," vol. 00, no. c, pp. 1–9, 2002.
- [6] C. Holcombe, "Advanced Guide to eCommerce," *LitLangs Publishing*, 2007.
- [7] L. Wen, Y. Ni, and B. Huang, "Analysis of the Application of Social E-commerce Marketing," no. Iccia, pp. 1106–1108, 2012.
- [8] M. Anderson, J. Sims, J. Price, and J. Brusa, "Turning ' Like ' to ' Buy ' Social Media Emerges as a Commerce Channel," 2011.
- [9] eMarketer, "More Time Spent on Social Media than Email Worldwide." [Online]. Available: <http://www.emarketer.com/Article.aspx?R=1008025>. [Accessed: 31-May-2015].
- [10] N. B. Ellison, C. Steinfield, and C. Lampe, "The benefits of facebook 'friends': Social capital and college students' use of online social network sites," *J. Comput. Commun.*, vol. 12, no. 4, pp. 1143–1168, 2007.
- [11] "Social Media Site Usage 2014 | Pew Research Center." [Online]. Available: <http://www.pewinternet.org/2015/01/09/social-media-update-2014/>. [Accessed: 01-Jun-2015].
- [12] P. Messmer, "Conversion Rate With Pre-Purchase Sharing." [Online]. Available: <http://www.addshoppers.com/blog/how-to-increase-your-conversion-rate-with-pre-purchase-sharing>. [Accessed: 03-Jun-2015].
- [13] WooThemes, "Smart Coupons | WooThemes Documentation." [Online]. Available: <http://docs.woothemes.com/document/smart-coupons/>. [Accessed: 04-Jun-2015].
- [14] M. Doernhoefer, "Surfing the net for software engineering notes," *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 3, p. 15, 2004.
- [15] D. Flanagan, *JavaScript: The Definitive Guide: Activate Your Web Pages (Definitive Guides)*, 6rd Editio. O'Reilly Media, Inc, 2011.
- [16] J. Garrett, "Ajax: A new approach to web applications," 2005. [Online]. Available: <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>.
- [17] D. M. Kristol, "HTTP Cookies: Standards, privacy, and politics," *ACM Trans. Internet Technol.*, vol. 1, no. 2, pp. 151–198, Nov. 2001.

- [18] G. Pujolle, A. Serhrouchni, and I. Ayadi, "Secure session management with cookies," *ICICS 2009 - Conf. Proc. 7th Int. Conf. Information, Commun. Signal Process.*, pp. 1–6, 2009.
- [19] K. Fu, E. Sit, K. Smith, and N. Feamster, "Dos and Don'ts of Client Authentication on the Web," *USENIX Secur.*, pp. 1–16, 2001.
- [20] D. X. D. Xu, C. L. C. Lu, and a. Dos Santos, "Protecting Web usage of credit cards using One-Time Pad cookie encryption," *18th Annu. Comput. Secur. Appl. Conf. 2002. Proceedings.*, 2002.
- [21] G. S. Vernam, "Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications," *Trans. Am. Inst. Electr. Eng.*, vol. XLV, pp. 109–115, 1926.
- [22] P. Eckersley, "How unique is your web browser?," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6205 LNCS, pp. 1–18, 2010.
- [23] T. Choi and M. G. Gouda, "HTTPI: An HTTP with integrity," *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, 2011.
- [24] W. Alcorn, C. Fritchot, and M. Orrù, *The Browser Hacker's Handbook*. John Wiley & Sons, Inc., 2014.
- [25] S. Whalen, "An introduction to arp spoofing," *Node99 Online Doc. April*, 2001.
- [26] I. Green, "DNS spoofing by the man in the middle," <http://www.sans.org/rr/whitepapers/dns/1567.php>, 2005.
- [27] A. Ornaghi, "Man in the middle attacks Demos The scenario," *Blackhat Conf. - USA*, 2003.
- [28] T. Dierks, "The TLS Protocol Version 1.0," 1999. [Online]. Available: <http://tools.ietf.org/html/rfc2246>. [Accessed: 03-Jul-2015].
- [29] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the HTTPS protocol," *IEEE Secur. Priv.*, vol. 7, no. 1, pp. 78–81, 2009.
- [30] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," *Proc. 2013 Conf. Internet Meas. Conf. - IMC '13*, pp. 291–304, 2013.
- [31] A. Zúquete, *Segurança em Redes Informáticas*, 3ª edição. FCA - Editora de Informática, Lda, 2010.
- [32] Expression Software Blog, "HTTPS Sequence Diagram" [Online]. Available: <http://blog.expressionsoftware.com/2011/02/https-sequence-diagram.html>. [Accessed: 03-Jul-2015].
- [33] D. Orchard, M. Champion, F. McCabe, H. Haas, D. Booth, C. Ferris, and E. Newcomer, "Web Services Architecture," 2004. [Online]. Available: <http://www.w3.org/TR/ws-arch/>. [Accessed: 02-Jul-2015].
- [34] D. P. D. Peng, C. L. C. Li, and H. H. H. Huo, "An extended UsernameToken-based approach for REST-style Web Service Security Authentication," *2009 2nd IEEE Int. Conf. Comput. Sci. Inf. Technol.*, 2009.
- [35] Y. V Natis, "Service-Oriented Architecture Scenario," 2003.
- [36] F. Curbera, M. Duftler, R. Khalaf, and W. Nagy, "Unraveling the Communication : SOAP," *IEEE Internet Comput.*, no. April, pp. 86–93, 2002.
- [37] L. Richardson and S. Ruby, *RESTful web services*. O'Reilly Media, Inc, 2008.

- [38] G. Serme, A. S. De Oliveira, J. Massiera, and Y. Roudier, "Enabling message security for RESTful services," in *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 2012, pp. 114–121.
- [39] A. DuVander, "New Job Requirement: Experience Building RESTful APIs," *Programmable Web*, 2010. [Online]. Available: <http://blog.programmableweb.com/2010/06/09/new-job-requirement-experience-building-restful-apis/>. [Accessed: 03-Jul-2015].
- [40] Google Trends, "Web Search interest: soap api, rest api - Worldwide, 2004 - present." [Online]. Available: <http://www.google.com/trends/explore?hl=en-US#q=soap+api,+rest+api&cmpt=q>. [Accessed: 03-Jul-2015].
- [41] R. Elfving, U. Paulsson, and L. Lundberg, "Performance of SOAP in Web Service environment compared to CORBA," *Ninth Asia-Pacific Softw. Eng. Conf. 2002.*, no. August, 2002.
- [42] M. Jones, J. Bradley, P. Identity, and N. Sakimura, "JSON Web Token (JWT)," 2012. [Online]. Available: <https://tools.ietf.org/html/rfc7519>. [Accessed: 10-Jul-2015].
- [43] J. Bradley, N. Sakimura, and M. Jones, "JSON Web Signature (JWS)," 2012. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-06>. [Accessed: 10-Jul-2015].
- [44] M. Jones, E. Rescorla, and J. Hildebrand, "JSON Web Encryption (JWE)," 2012. [Online]. Available: <http://self-issued.info/docs/draft-ietf-jose-json-web-encryption.html>. [Accessed: 10-Jul-2015].
- [45] T. McLean, "Critical vulnerabilities in JSON Web Token libraries." [Online]. Available: <https://auth0.com/blog/2015/03/31/critical-vulnerabilities-in-json-web-token-libraries/>. [Accessed: 10-Jul-2015].
- [46] S. Dhande and S. R. Lodha, "Web Database Security Techniques," 2014.
- [47] J. Grossman, "Cross-Site Scripting Worms & Viruses - The Impending Threat & the Best Defense," 2006.
- [48] H. Saiedian and D. Broyle, "Security vulnerabilities in the same-origin policy: Implications and alternatives," *Computer (Long Beach, Calif.)*, vol. 44, no. 9, pp. 29–36, 2011.
- [49] C. Jackson and H. J. Wang, "Subspace: secure cross-domain communication for web mashups," *Proc. 16th Int. Conf. World Wide Web*, pp. 611–620, 2007.
- [50] P. De Ryck, M. Decat, L. Desmet, F. Piessens, and W. Joosen, "Security of web mashups: A survey," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7127 LNCS, pp. 223–238, 2012.
- [51] M. Dworkin, "Recommendation for Block Cipher Modes of Operation. Methods and Techniques," *Nist Spec. Publ.*, 2001.
- [52] H. O. Alanazi, B. B. Zaidan, a. a. Zaidan, H. a. Jalab, M. Shabbir, and Y. Al-Nabhani, "New Comparative Study Between DES, 3DES and AES within Nine Factors," vol. 2, no. 3, pp. 152–157, 2010.
- [53] S. Lucks, "Attacking Triple Encryption," *Fast Softw. Encryption, 5th Int. Work. FSE '98, Paris, Fr. March 23-25, 1998, Proc.*, vol. 1372, pp. 239–253, 1998.
- [54] C. Snyder, M. Southwell, and T. Myer, *Pro PHP security*. Apress, 2010.

- [55] K. H. Boey, Y. Lu, M. O'Neill, and R. Woods, "Differential Power Analysis of CAST-128," *VLSI (ISVLSI), 2010 IEEE Comput. Soc. Annu. Symp.*, pp. 143–148, 2010.
- [56] C. Adams, "The CAST-128 Encryption Algorithm." [Online]. Available: <https://tools.ietf.org/html/rfc2144>. [Accessed: 05-Jul-2015].
- [57] J. Gilchrist and C. Adams, "The CAST-256 Encryption Algorithm." [Online]. Available: <https://tools.ietf.org/html/rfc2612>. [Accessed: 06-Jul-2015].
- [58] X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, C. Ma, and D. Gu, "Algebraic Fault Analysis on GOST for Key Recovery and Reverse Engineering," *2014 Work. Fault Diagnosis Toler. Cryptogr.*, pp. 29–39, 2014.
- [59] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in *Selected Areas in Cryptography*, vol. 2259, 2001, pp. 1–24.
- [60] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4," *MD5, HAVAL-128 RIPEMD*, vol. 5, pp. 5–8, 2004.

Apêndices

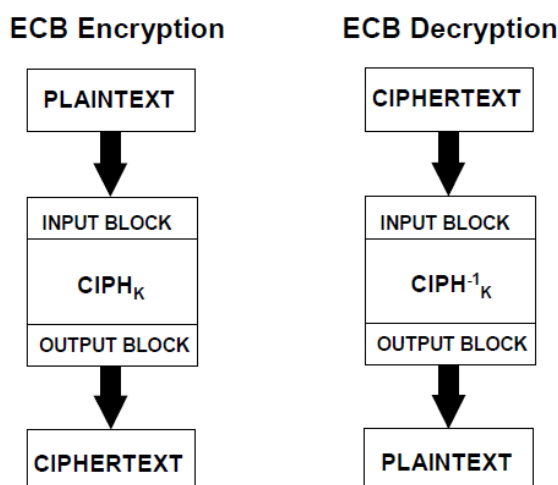
Apêndice A: Mecanismos criptográficos

Este apêndice contém a continuação do estudo teórico acerca das diversas técnicas criptográficas existentes.

De seguida apresentaremos os diferentes modos de cifra por bloco conhecidos, além do CBC.

Electronic Codebook (ECB)

Este é o modo mais simples e intuitivo de cifra por bloco. Consiste em dividir o texto em blocos independentes e contíguos de igual dimensão que são cifrados independentemente. Deste modo, cada bloco de texto é transformado diretamente em bloco cifrado, independentemente dos blocos anteriores. Na de cifra o processo é o mesmo, ou seja, cada bloco cifrado é transformado em blocos de texto independentes [31]. A seguinte figura demonstra como se procede a cifra e a decifra utilizando este método de cifra por bloco



Legenda: Método de cifra por bloco ECB [51]

A grande vantagem deste método é que os blocos podem ser todos cifrados simultaneamente visto que são cifrados independentemente. Deste modo, os blocos podem ser cifrados utilizando por exemplo um GPU onde cada microprocessador pode cifrar cada bloco, tornando este método extremamente rápido.

A grande vulnerabilidade deste método é a introdução de padrões, ou seja, um bloco de texto igual produz um bloco cifrado igual, se for cifrado com a mesma chave. Deste modo, este método de cifra por bloco não garante a confidencialidade da mensagem nem a proteção dos dados [54].

Um grande exemplo da ineficácia deste modo de cifra é quando tentamos cifrar uma imagem bitmap que utilize grandes áreas de cor uniforme. Neste caso, enquanto cada *pixel* é cifrado, a imagem cifrada poder-se-á aproximar da original. Analisemos o seguinte exemplo da imagem:



Imagem Original

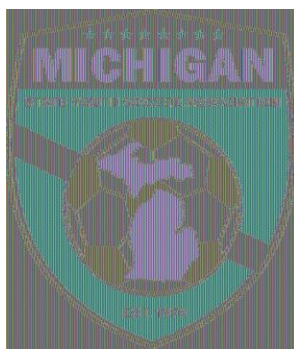


Imagem cifrada utilizando ECB



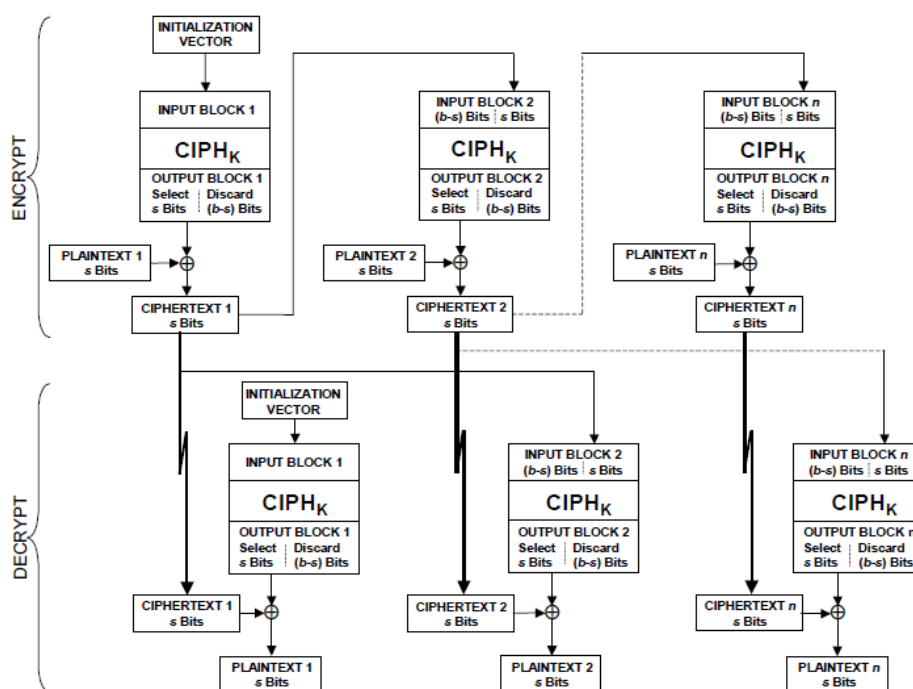
Imagem cifrada utilizando outros modos de cifra (CBC, CFB, ...)

Legenda: Cifra de uma imagem BMP com o modo ECB ou com outros modos (por exemplo CBC, CFB ou OFB)

Como podemos observar, cifrando a imagem utilizando o modo de cifra por bloco ECB, a imagem resultante é bem perceptível e aproximada, não adicionando confidencialidade sobre a imagem inicial. No entanto, se quisermos garantir confidencialidade, e privacidade através deste método de cifra (baixa privacidade) é possível e extremamente eficiente através deste método, sendo necessário adicionar uma assinatura (HMAC) com vista a validar os dados.

Cipher Feedback Mode (CFB)

Este modo de cifra transforma uma cifra por bloco numa cifra contínua [31]. Na cifra por CFB, o primeiro bloco a ser cifrado é o IV. Depois de cifrado, ao resultado são selecionados os k bits mais significativos, sendo a estes aplicado um XOR com o primeiro bloco de texto a cifrar, produzindo assim o primeiro bloco cifrado. Os restantes bits do IV cifrado serão utilizados no segundo bloco, onde são concatenados com os k bits mais significativos do primeiro bloco cifrado. Este novo bloco é cifrado, de onde os k bits mais significativos são selecionados, onde é aplicado um XOR com o segundo bloco a cifrar. Uma alternativa á utilização dos restantes bits é aplicar ao bloco cifrado um *shift* k vezes de modo a obter o tamanho de bloco pretendido para ser cifrado. Este processo de cifra é repetido sucessivamente até ao último bloco. A seguinte figura ilustra como funciona este método de cifra por bloco, para as operações de cifra e decifra:



Legenda: Método de cifra por bloco CFB

No processo de decifra, o IV é o primeiro bloco de entrada, e os blocos seguintes são formados como no processo de cifra, a partir da concatenação dos $N-k$ bits menos significativos do bloco de entrada anterior, com os k bits mais significativos do bloco previamente cifrado, sendo N o número total de bits do bloco de entrada. O processo de decifra é aplicado a cada bloco de entrada produzindo um bloco de saída. Aos k bits mais significativos desse bloco é aplicado um XOR com o bloco cifrado produzindo assim o bloco de texto decifrado.

Como no modo de cifra descrito no capítulo 2. (CBC) o bloco de entrada (exceto o primeiro) depende do resultado do bloco cifrado anteriormente, não sendo possível o processo de cifra ser realizado em paralelo. No entanto, é possível realizar o processo de decifra em paralelo se os blocos de entrada forem previamente construídos (em série) a partir do IV e dos blocos cifrados [51].

Output Feedback Mode (OFB)

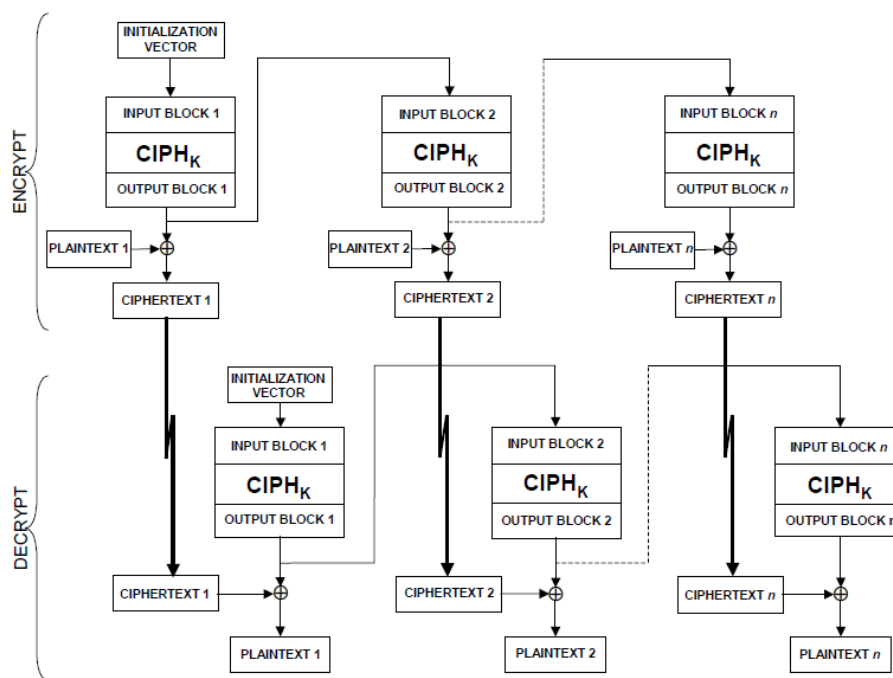
Há semelhança do modo de cifra especificado anteriormente (CFB), este modo pretende transformar uma cifra por bloco numa cifra contínua [31]. Neste modo de cifra por bloco há a necessidade de um vetor de inicialização (IV) para o primeiro bloco, como se verifica nos modos de cifra por bloco anteriormente enunciados (exceto no modo ECB).

A diferença deste modo de cifra para o CFB está na função de realimentação, onde a cifra do bloco seguinte depende da saída do gerador em pleno em vez do texto cifrado (CFB) [31].

Neste modo de cifra por bloco o IV é inicialmente transformado pela função de cifra produzindo o primeiro bloco de saída. A este bloco é aplicado um XOR com o primeiro texto a cifrar produzindo o primeiro bloco de texto cifrado. O seguinte bloco vai depender do bloco de saída do gerador, que será o bloco de entrada. Este bloco será cifrado, onde posteriormente é aplicado um XOR com o segundo bloco de texto a cifrar, gerando deste modo o segundo bloco cifrado. O processo repete-se para os restantes blocos. Para o último bloco, que pode ser incompleto, apenas os k bits mais significativos do penúltimo bloco são utilizados, de onde os N-k bits são descartados, sendo N o tamanho do penúltimo bloco e k o tamanho do último bloco [51].

No processo de decifra, o IV é igualmente transformado pela função de cifra produzindo o primeiro bloco de saída. A este bloco é aplicado um XOR com o primeiro bloco cifrado produzindo o primeiro bloco de texto. Para o segundo bloco, o primeiro bloco sofre uma cifra, de acordo com a função de cifra. De seguida, ao resultado será aplicado um XOR com o segundo bloco cifrado, resultando assim o segundo bloco de texto. O processo repete-se para os restantes blocos. Para o último bloco, o processo é idêntico ao de cifra, utilizando os k bits mais significativos do bloco anterior, sendo k o tamanho do bloco a decifrar [51].

Na figura seguinte, apresentamos como se processa o método de cifra e decifra desta técnica de cifra por bloco:



Legenda: Método de cifra por bloco OFB

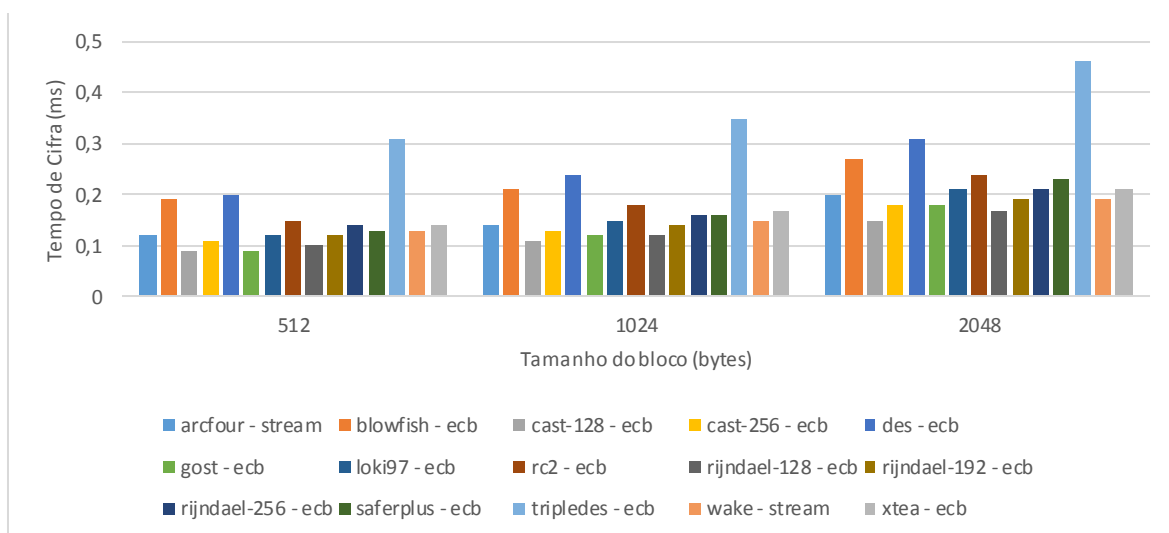
Neste caso, tanto na cifra como na decifra, cada bloco depende do anterior (exceto o primeiro). Deste modo, estes processos não podem ser feitos em paralelo. No entanto, se o IV for conhecido, os blocos de saída podem ser gerados primeiramente para avaliação do texto ou da cifra. No modo OFB é requerido um IV único para cada mensagem a ser cifrada, com a mesma chave. Se pelo contrário o mesmo IV for utilizado em duas ou mais mensagens distintas, então a confidencialidade dessas mensagens pode ser comprometida. Particularmente, se um bloco de texto dessas mensagens for conhecido, por exemplo o i-ésimo bloco, o resultado da saída do gerador da cifra desse bloco pode ser determinado facilmente, a partir desse bloco cifrado.

Isto permite que o i-gésimo bloco de texto de outra mensagem que utilize o mesmo IV seja facilmente decifrável [51].

Apêndice B: Benchmarking de Cifras

Neste apêndice iremos apresentar um estudo prático sobre os diferentes algoritmos de cifra por bloco, conjugando os diversos modos (ECB, CFB, OFB e OFB com blocos de n bits). O modo de cifra por bloco CBC foi apresentado no capítulo 2. Em todos os gráficos são apresentados além dos algoritmos de cifra por bloco, são apresentados os algoritmos de cifra contínua, de modo a se poderem comparar com os diversos modos de cifra por bloco. Nesses gráficos também é variado o tamanho do bloco a cifrar (512, 1024 e 2048 bytes). Relembramos que para medições de tempo foi utilizada a função *microtime* do PHP e que a máquina utilizada para estes testes opera com um CPU Intel Core i5-3317U CPU 1.70GHz, com 6GB de RAM, no sistema operativo *Ubuntu* 14.10. Este programa corre em PHP, sobre consola, utilizando a biblioteca *MCrypt*.

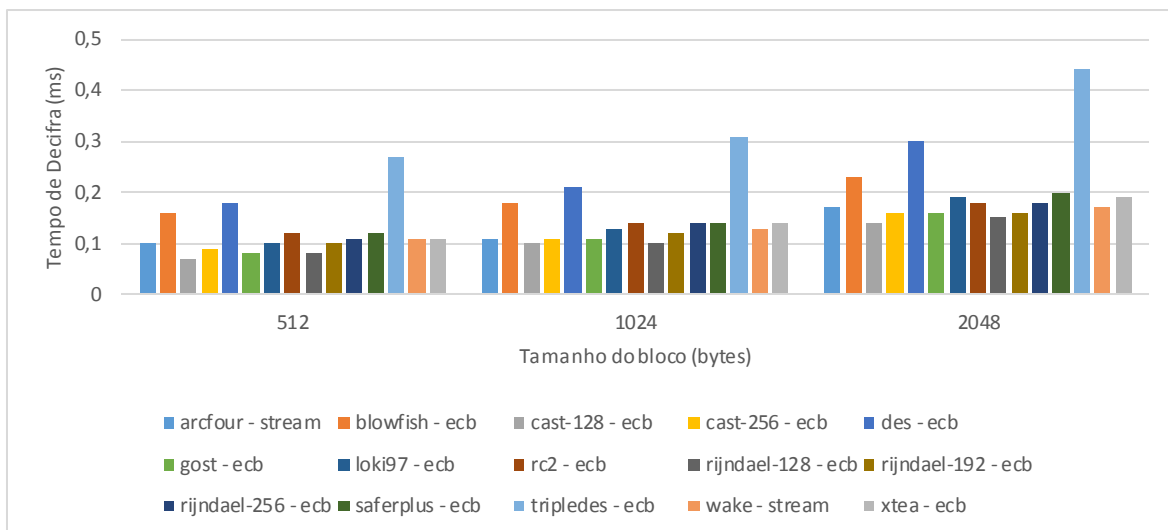
O seguinte gráfico representa os tempos obtidos no processo e cifra utilizando o modo Electronic Codebook (ECB), nos diversos algoritmos conhecidos:



Legenda: Gráfico com os tempo de cifra de diversos algoritmos com o modo de cifra por bloco ECB e cifras contínuas VS tamanho do bloco a cifrar

Como podemos verificar, neste modo de cifra por bloco, os algoritmos com maior performance no processo de cifra são: (1) CAST-128, (2) Rijndael-128 e (3) Gost.

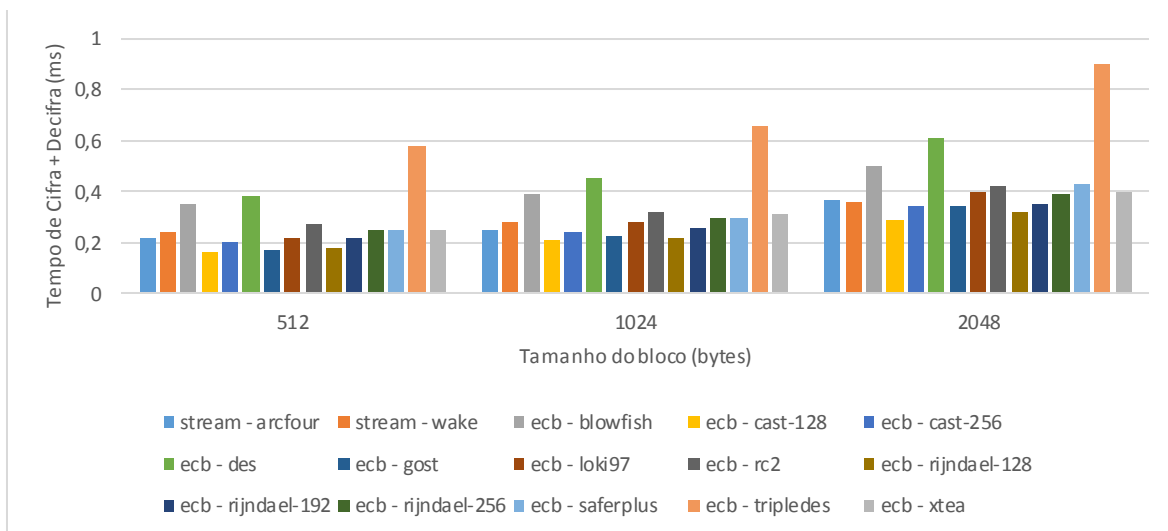
De seguida, para o mesmo modo de cifra por bloco, iremos mostrar o gráfico do processo de decifra.



Legenda: Gráfico com os tempos de decifra de diversos algoritmos com o modo de cifra por bloco ECB e cifras contínuas VS tamanho do bloco que foi cifrado

No processo de decifra, os tempos de resposta são idênticos, sendo que os algoritmos mais rápidos a cifrar são também os mais rápidos a decifrar: (1) CAST-128, (2) Rijndael-128 e (3) Gost.

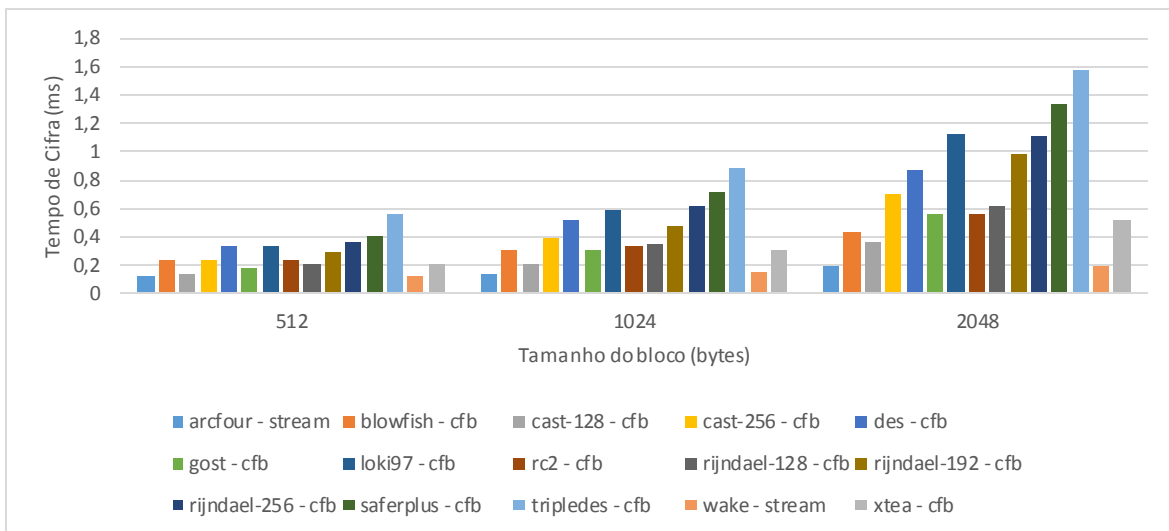
O seguinte gráfico representa a soma dos tempos de cifra e decifra para o modo de cifra por bloco ECB.



Legenda: Gráfico com os tempos de cifra + decifra de diversos algoritmos com o modo de cifra por bloco ECB e cifras contínuas VS tamanho do bloco a cifrar

Com a análise deste gráfico, podemos concluir que o modo ECB é um modo de cifra por bloco extremamente rápido, onde cifrar e decifrar um bloco de 2048 bytes com um dos algoritmos mais rápidos demoraria cerca de 0,3 ms.

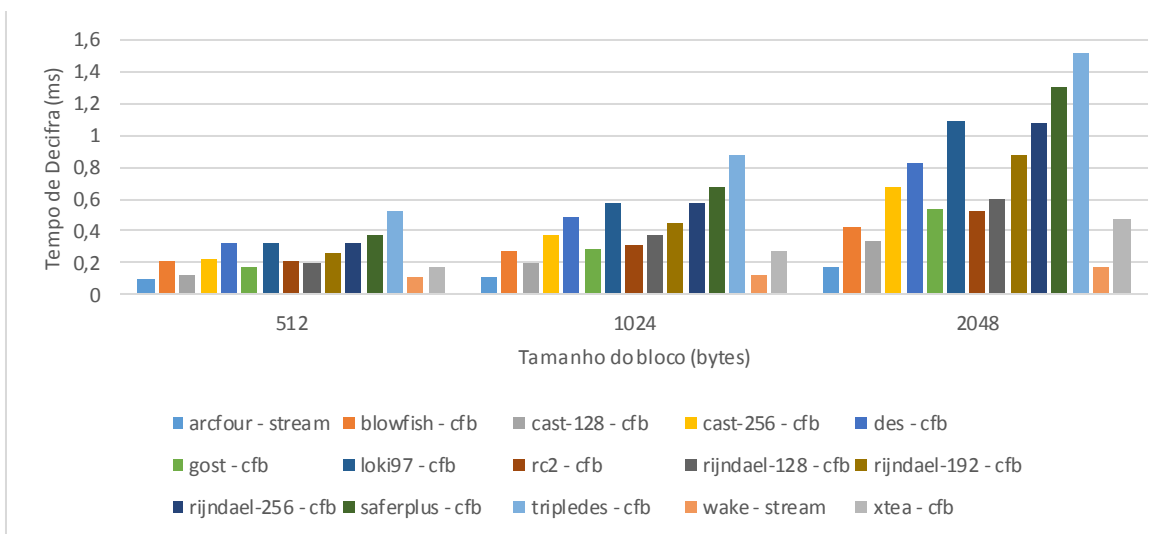
O gráfico seguinte mostra os tempos do processo de cifra no modo de cifra por bloco CFB (Cipher FeedBack) e cifras contínuas, variando o tamanho do bloco:



Legenda: Gráfico com os tempos de cifra de diversos algoritmos com o modo de cifra por bloco CFB e cifras contínuas VS tamanho do bloco a cifrar

Sendo o modo de cifra por bloco CFB um pouco lento, neste, os algoritmos de cifra contínua acabam por ser os mais rápidos a cifrar. Assim, os algoritmos que têm uma maior performance no processo de cifra são os seguintes: (1) wake (cifra contínua), (2) arcfour (cifra contínua) e (3) CAST-128 (cifra por bloco).

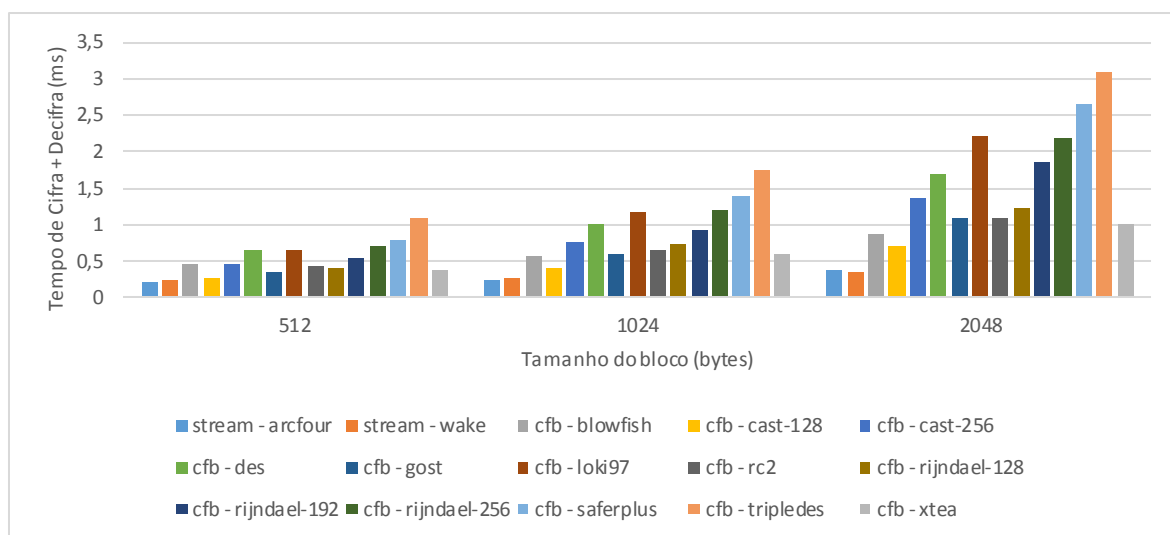
No seguinte caso, avaliaremos os tempos de decifra de diversos algoritmos com o modo de cifra por bloco CFB, bem como os algoritmos de cifra contínua, variando o tamanho do bloco, em forma de gráfico:



Legenda: Gráfico com os tempos de decifra de diversos algoritmos com o modo de cifra por bloco CFB e cifras contínuas VS tamanho do bloco que foi cifrado

Como se pode verificar, os algoritmos de cifra contínua são os que têm uma melhor performance no processo de decifra, sendo que os mais rápidos são: (1) Wake (cifra contínua), (2) Arcfour (cifra contínua) e (3) CAST-128 (cifra por bloco).

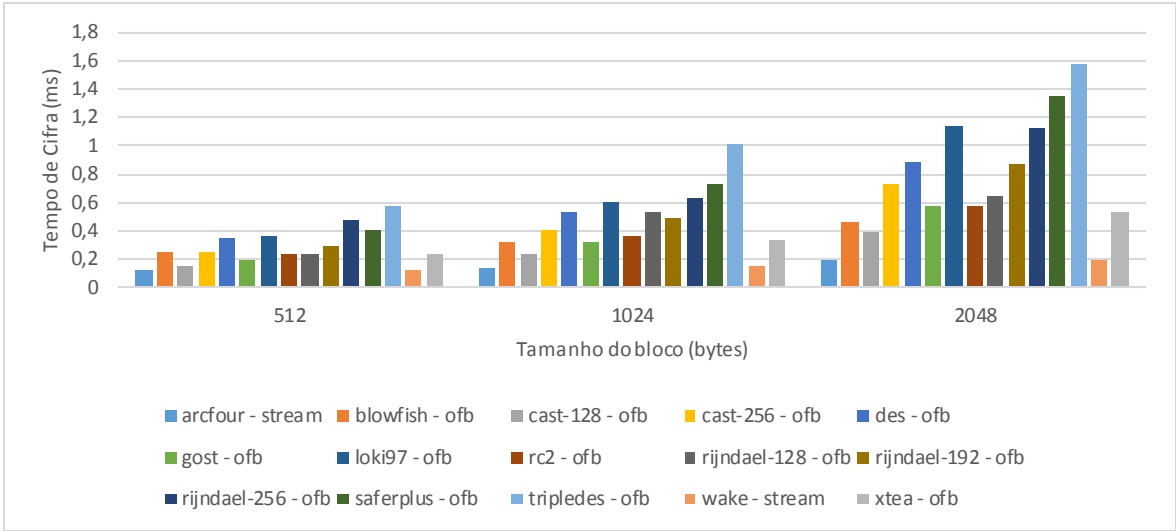
Por último, apresentaremos o gráfico para o processo completo neste modo de cifra, com os diferentes tamanhos de bloco.



Legenda: Gráfico com os tempos de cifra + decifra de diversos algoritmos com o modo de cifra por bloco CFB e cifras contínuas VS tamanho do bloco a cifrar

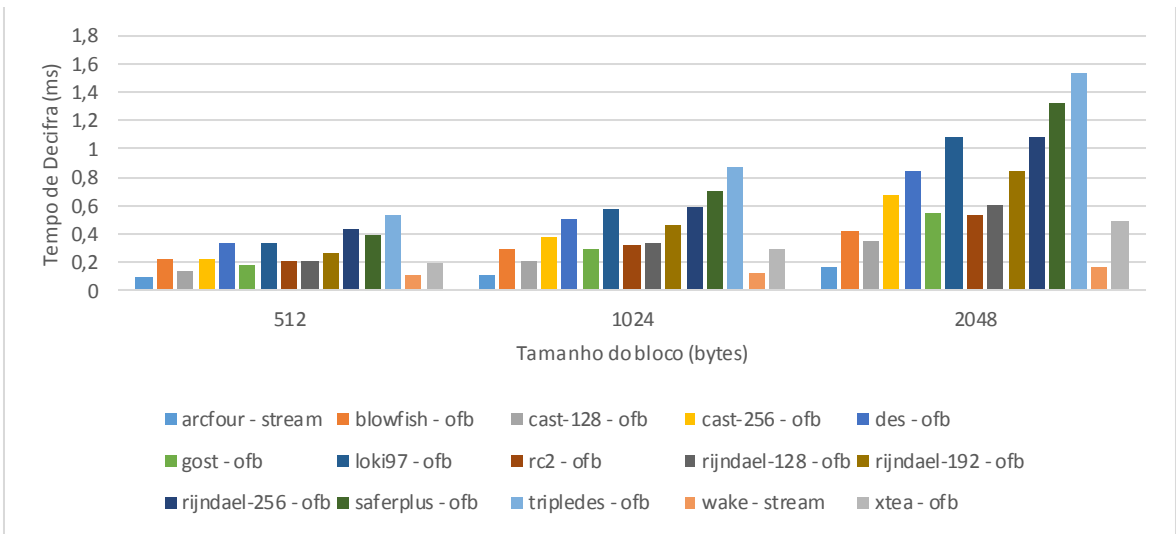
Este gráfico serve para reforçar que o processo de cifra e decifra com o modo CFB é extremamente lento, sendo que os algoritmos de cifra contínua são mais rápidos, pois não operam num modo de cifra por bloco.

O seguinte gráfico apresenta a performance dos diferentes algoritmos no modo de cifra por bloco OFB, bem como os algoritmos de cifra contínua, para a técnica de cifra:



Legenda: Gráfico com os tempos de cifra de diversos algoritmos com o modo de cifra por bloco OFB e cifras contínuas VS tamanho do bloco a cifrar

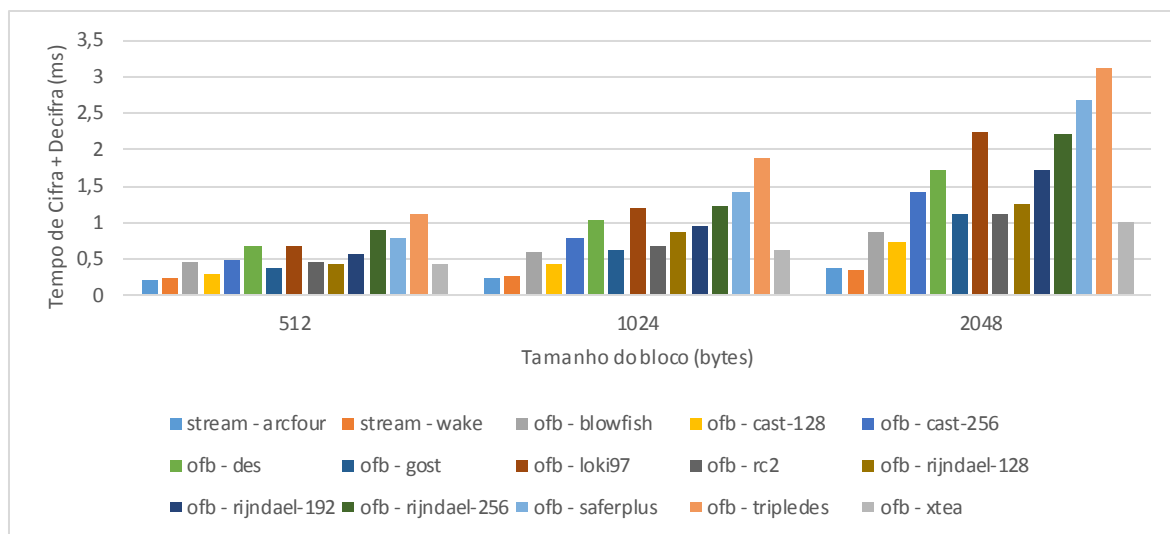
Como podemos verificar, operando num modo de cifra por bloco OFB, é mais vantajoso em termos temporais cifrar com uma cifra contínua, sendo que os algoritmos mais rápidos são: (1) Wake (cifra contínua), (2) Arcfour (cifra contínua) (3) CAST-128 (cifra por bloco).



Legenda: Gráfico com os tempos de decifra de diversos algoritmos com o modo de cifra por bloco OFB e cifras contínuas VS tamanho do bloco que foi cifrado

Os tempos de decifra operando no modo de cifra por bloco OFB são bastante lentos, sendo mais rentável temporalmente proceder a uma cifra contínua. Assim, os algoritmos de cifra mais rápidos no processo de decifra são os enunciados anteriormente, (1) Wake (cifra contínua), (2) Arcfour (cifra contínua) (3) CAST-128 (cifra por bloco).

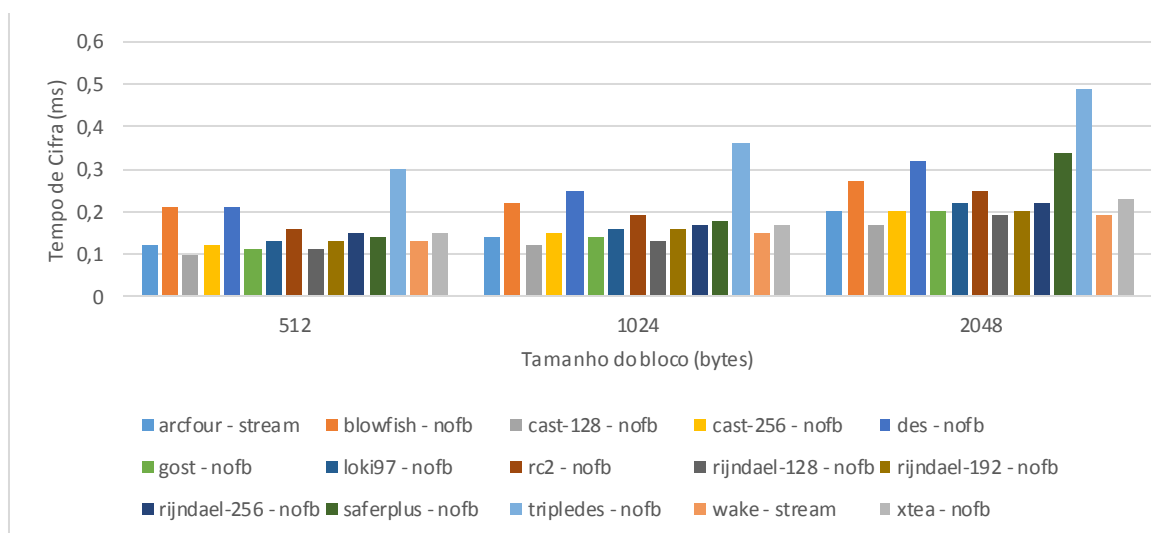
De seguida apresentaremos o gráfico do processo de cifra e decifra operando no modo OFB, de modo a retirarmos os tempos totais de cifra e decifra dos algoritmos mais rápidos:



Legenda: Gráfico com os tempo de cifra + decifra de diversos algoritmos com o modo de cifra por bloco OFB e cifras contínuas VS tamanho do bloco a cifrar

Neste gráfico podemos verificar que os algoritmos mais rápidos noutros modos de cifra por bloco (ECB e CBC), neste modo demoram cerca de 1ms, a exceção do CAST-128, que demora cerca de 0,7ms, para blocos de 2048 bytes. Por isso compensa utilizar algoritmos de cifra contínua, que neste caso, para o processo completo, apresentam tempos menores que 0,5ms.

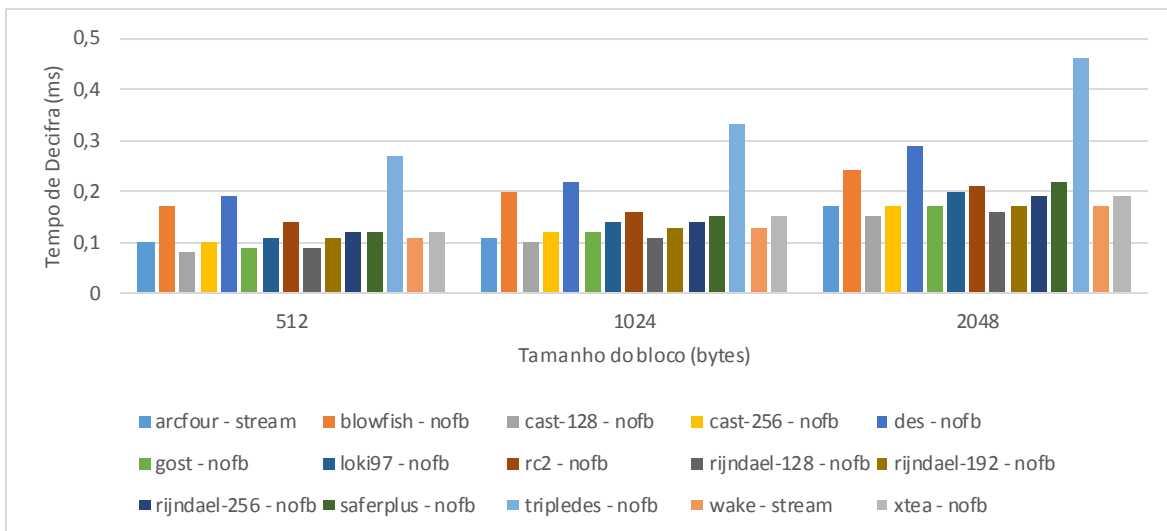
O seguinte gráfico apresenta a performance do modo de cifra por bloco N-OFB (OFB com blocos de n bits, onde n é o tamanho do bloco utilizado pelo algoritmo), no método de cifra:



Legenda: Gráfico com os tempos de cifra de diversos algoritmos com o modo de cifra por bloco OFB com n bits (n é o tamanho do bloco do algoritmo de cifra) e cifras contínuas VS tamanho do bloco a cifrar

Ao contrário do que verificámos nos métodos OFB e CFB, operando no modo N-OFB, os algoritmos com maior performance a cifrar mensagens são os algoritmos que operam no modo de cifra por bloco, sendo os três mais rápidos: (1) CAST-128, (2) Rijndael-128 e (3) Gost.

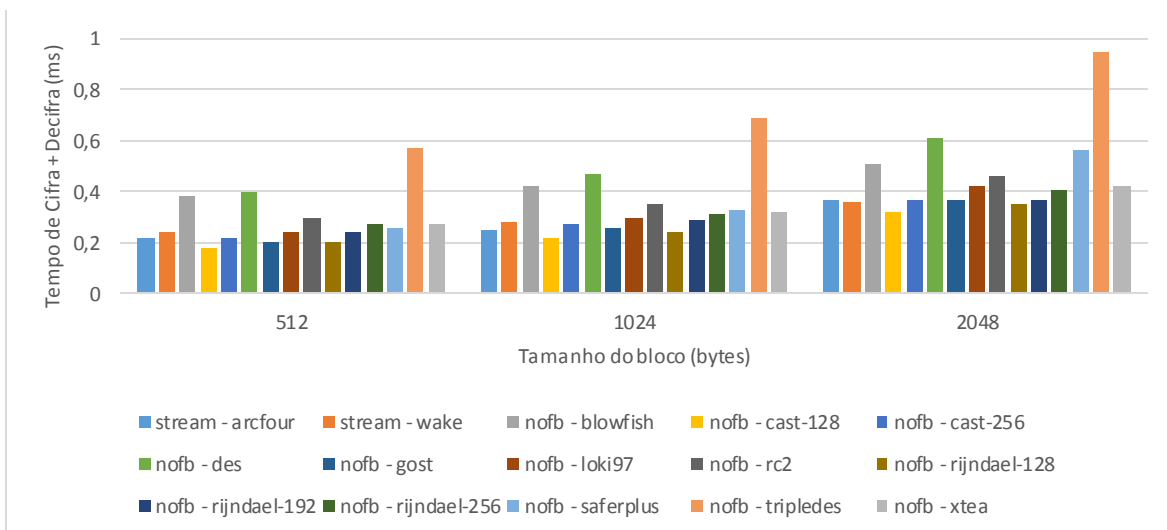
De seguida, apresentaremos o gráfico que contém os resultados de performance deste modo de cifra por bloco para a técnica de decifra:



Legenda: Gráfico com os tempos de decifra de diversos algoritmos com o modo de cifra por bloco OFB com n bits (n é o tamanho do bloco do algoritmo de cifra) e cifras contínuas VS tamanho do bloco que foi cifrado

Os algoritmos mais rentáveis no processo de decifra no modo de cifra por bloco N-OFB são os mesmos algoritmos determinados pelo processo de cifra: (1) CAST-128, (2) Rijndael-128 e (3) Gost.

Por fim, o gráfico seguinte apresenta os tempos do processo completo (cifra e decifra), dos diversos algoritmos operando no modo de cifra por bloco N-OFB:



Legenda: Gráfico com os tempos de cifra + decifra de diversos algoritmos com o modo de cifra por bloco OFB com n bits (n é o tamanho do bloco do algoritmo de cifra) e cifras contínuas VS tamanho do bloco a cifrar

Deste modo, podemos concluir que este método de cifra por bloco é bastante rápido, ficando assim as cifras contínuas derrotadas em termos de performance, ao contrário do que verificamos nos métodos OFB e CFB. Com efeito, os algoritmos de cifra mais rápido neste modo de cifra por bloco são os seguintes: (1) CAST-128, (2) Rijndael-128 e (3) Gost.

Apêndice C: Resultados dos testes de carga

Este apêndice contém os resultados para os testes de carga corridos, nos diferentes cenários, analisados na secção 6.1.

Cenário A1 (Cupão idêntico com segurança completa)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	48,980	48,825	48,980	48,825	20,42	20,48	0	0
2			48,610		48,610		20,57		0	
3			48,884		48,884		20,46		0	
1	1000	40	47,635	47,642	47,635	47,642	20,99	20,99	0	0
2			47,658		47,658		20,98		0	
3			47,634		47,634		20,99		0	
1	1000	60	47,297	47,274	47,297	47,274	21,14	21,15	0	0
2			47,234		47,234		21,17		0	
3			47,291		47,291		21,15		0	
1	1000	80	47,196	47,199	47,196	47,199	21,19	21,19	0	0
2			47,178		47,178		21,20		0	
3			47,224		47,224		21,18		0	
1	1000	100	47,151	47,114	47,151	47,114	21,21	21,22	0	0
2			47,120		47,120		21,22		0	
3			47,072		47,072		21,24		0	
1	1000	120	47,349	47,115	47,349	47,115	21,12	21,22	0	0
2			47,032		47,032		21,26		0	
3			46,963		46,963		21,29		0	
1	1000	140	47,898	48,505	47,898	48,505	20,88	20,62	0	0
2			47,859		47,859		20,89		0	
3			49,758		49,758		20,10		0	
1	1000	160	47,506	48,389	47,506	48,389	21,05	20,67	42	29
2			48,506		48,506		20,62		24	
3			49,154		49,154		20,34		19	
1	1000	180	38,814	38,535	38,814	38,535	25,76	25,95	252	263
2			38,445		38,445		26,01		267	
3			38,346		38,346		26,08		270	
1	1000	200	33,021	32,888	33,021	32,888	30,28	30,41	402	407
2			32,968		32,968		30,33		405	
3			32,674		32,674		30,61		413	
1	1000	220	32,251	32,099	32,251	32,099	31,01	31,16	417	425
2			31,892		31,892		31,36		431	
3			32,154		32,154		31,10		425	
1	1000	240	31,110	30,958	31,110	30,958	32,14	32,30	453	452
2			30,900		30,900		32,36		450	
3			30,864		30,864		32,40		451	
1	1000	260	31,285	31,028	31,285	31,028	31,96	32,23	445	449
2			30,692		30,692		32,58		453	
3			31,108		31,108		32,15		448	

Legenda: Tabela dos resultados do cenário A1 (mesmo cupão para diferentes pedidos com segurança completa)

Cenário A2 (Cupão idêntico com segurança parcial, só loja)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	46,589	46,606	46,589	46,606	21,46	21,457	0	0
2			46,626		46,626		21,45		0	
3			46,604		46,604		21,46		0	
1	1000	40	46,777	46,791	46,777	46,791	21,38	21,373	0	0
2			46,733		46,733		21,40		0	
3			46,864		46,864		21,34		0	
1	1000	60	46,664	46,859	46,664	46,859	21,43	21,340	0	0
2			46,765		46,765		21,38		0	

3			47,147		47,147		21,21		0	
1			46,788		46,788		21,37		0	
2	1000	80	46,763	46,782	46,763	46,782	21,38	21,373	0	0
3			46,795		46,795		21,37		0	
1			46,861		46,861		21,34		0	
2	1000	100	46,967	46,870	46,967	46,870	21,29	21,337	0	0
3			46,782		46,782		21,38		0	
1			47,072		47,072		21,24		0	
2	1000	120	46,962	46,977	46,962	46,977	21,29	21,283	0	0
3			46,897		46,897		21,32		0	
1			46,706		46,706		21,41		0	
2	1000	140	46,905	46,769	46,905	46,769	21,32	21,383	0	0
3			46,695		46,695		21,42		0	
1			45,232		45,232		22,11		62	
2	1000	160	47,415	46,619	47,415	46,619	21,09	21,460	23	39
3			47,210		47,210		21,18		30	
1			39,111		39,111		25,57		242	
2	1000	180	39,158	39,292	39,158	39,292	25,54	25,453	246	242
3			39,606		39,606		25,25		238	
1			33,189		33,189		30,13		402	
2	1000	200	33,267	33,085	33,267	33,085	30,06	30,227	391	399
3			32,798		32,798		30,49		404	
1			32,587		32,587		30,69		407	
2	1000	220	32,311	32,254	32,311	32,254	30,95	31,007	421	418
3			31,865		31,865		31,38		424	
1			31,598		31,598		31,65		438	
2	1000	240	30,996	31,119	30,996	31,119	32,26	32,140	447	445
3			30,764		30,764		32,51		450	
1			31,266		31,266		31,98		446	
2	1000	260	31,848	31,331	31,848	31,331	31,40	31,920	441	445
3			30,879		30,879		32,38		447	

Legenda: Tabela dos resultados do cenário A2 (mesmo cupão para diferentes pedidos com segurança parcial, só do lado da loja)

Cenário A3 (Cupão idêntico com segurança parcial, só Optiprizer)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	46,937	46,921	46,937	46,921	21,30	21,310	0	0
2			46,919		46,919		21,31		0	
3			46,908		46,908		21,32		0	
1	1000	40	47,155	47,184	47,155	47,184	21,21	21,197	0	0
2			47,196		47,196		21,19		0	
3			47,201		47,201		21,19		0	
1	1000	60	47,732	47,661	47,732	47,661	20,95	20,980	0	0
2			47,636		47,636		20,99		0	
3			47,616		47,616		21,00		0	
1	1000	80	47,731	47,640	47,731	47,640	20,95	20,990	0	0
2			47,549		47,549		21,03		0	
3			47,640		47,640		20,99		0	
1	1000	100	47,618	47,618	47,618	47,618	21,00	21,000	0	0
2			47,611		47,611		21,00		0	
3			47,624		47,624		21,00		0	
1	1000	120	47,464	47,480	47,464	47,480	21,07	21,063	0	0
2			47,507		47,507		21,05		0	
3			47,470		47,470		21,07		0	
1	1000	140	47,745	47,696	47,745	47,696	20,94	20,967	0	0
2			47,714		47,714		20,96		0	
3			47,629		47,629		21,00		0	
1	1000	160	47,644	48,153	47,644	48,153	20,99	20,773	30	28
2			47,574		47,574		21,02		37	
3			49,241		49,241		20,31		15	
1	1000	180	38,828	39,138	38,828	39,138	25,75	25,550	247	245
2			39,106		39,106		25,57		239	
3			39,481		39,481		25,33		247	
1	1000	200	33,021	32,931	33,021	32,931	30,28	30,367	401	404
2			33,004		33,004		30,30		401	
3			32,767		32,767		30,52		409	
1	1000	220	32,321	32,221	32,321	32,221	30,94	31,037	411	417
2			31,967		31,967		31,28		422	
3			32,376		32,376		30,89		417	
1	1000	240	31,218	31,301	31,218	31,301	32,03	31,947	443	442
2			31,201		31,201		32,05		445	
3			31,485		31,485		31,76		438	
1	1000	260	31,032	31,180	31,032	31,180	32,23	32,077	445	443
2			31,039		31,039		32,22		449	
3			31,469		31,469		31,78		434	

Legenda: Tabela dos resultados do cenário A3 (mesmo cupão para diferentes pedidos com segurança parcial, lado do Optiprizer)

Cenário A4 (Cupão idêntico sem segurança implementada)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	45,791	45,803	45,791	45,803	21,84	21,833	0	0
2			45,893		45,893		21,79		0	
3			45,726		45,726		21,87		0	
1	1000	40	46,292	46,201	46,292	46,201	21,60	21,643	0	0
2			46,207		46,207		21,64		0	
3			46,105		46,105		21,69		0	
1	1000	60	46,959	46,943	46,959	46,943	21,29	21,303	0	0
2			46,914		46,914		21,32		0	
3			46,956		46,956		21,30		0	
1	1000	80	46,805	46,820	46,805	46,820	21,37	21,360	0	0
2			46,974		46,974		21,29		0	
3			46,680		46,680		21,42		0	
1	1000	100	46,822	46,692	46,822	46,692	21,36	21,417	0	0
2			46,616		46,616		21,45		0	
3			46,638		46,638		21,44		0	
1	1000	120	46,692	46,665	46,692	46,665	21,42	21,430	0	0
2			46,661		46,661		21,43		0	
3			46,641		46,641		21,44		0	
1	1000	140	46,789	46,788	46,789	46,788	21,37	21,373	0	0
2			46,838		46,838		21,35		0	
3			46,738		46,738		21,40		0	
1	1000	160	47,653	46,866	47,653	46,866	20,99	21,343	13	32
2			46,064		46,064		21,71		51	
3			46,882		46,882		21,33		31	
1	1000	180	38,270	38,117	38,270	38,117	26,13	26,237	245	249
2			37,682		37,682		26,54		259	
3			38,399		38,399		26,04		241	
1	1000	200	32,330	32,267	32,330	32,267	30,93	30,993	395	395
2			32,241		32,241		31,02		395	
3			32,229		32,229		31,03		395	
1	1000	220	29,247	29,752	29,247	29,752	34,19	33,667	464	454
2			28,552		28,552		35,02		469	
3			31,456		31,456		31,79		428	
1	1000	240	31,830	31,061	31,830	31,061	31,42	32,207	435	436
2			30,422		30,422		32,87		433	
3			30,930		30,930		32,33		440	
1	1000	260	30,808	30,918	30,808	30,918	32,46	32,343	443	445
2			30,768		30,768		32,50		449	
3			31,179		31,179		32,07		441	

Legenda: Tabela dos resultados do cenário A4 (mesmo cupão para diferentes pedidos sem segurança implementada)

Cenário B1 (Cupão diferente para cada pedido com segurança total)										
n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	150,272	144,961	150,272	144,961	6,65	6,90	23	19
2			141,182		141,182		7,08		16	
3			143,428		143,428		6,97		17	
1	1000	40	138,725	139,418	138,725	139,418	7,21	7,17	5	8
2			139,837		139,837		7,15		8	
3			139,691		139,691		7,16		11	
1	1000	60	138,878	138,042	136,878	137,376	7,20	7,24	6	9
2			138,665		138,665		7,21		1	
3			136,584		136,584		7,32		19	
1	1000	80	136,019	135,976	136,019	135,976	7,35	7,35	5	8
2			136,398		136,398		7,33		4	
3			135,510		135,510		7,38		13	
1	1000	100	133,987	134,643	133,987	134,643	7,46	7,43	11	7
2			135,127		135,127		7,40		5	
3			134,816		134,816		7,42		5	
1	1000	120	131,344	132,072	131,344	132,072	7,61	7,57	8	9
2			132,196		132,196		7,56		9	
3			132,675		132,675		7,54		8	
1	1000	140	131,312	130,967	131,312	130,967	7,62	7,64	11	10
2			130,393		130,393		7,67		6	
3			131,196		131,196		7,62		11	
1	1000	160	87,191	89,018	87,191	89,018	11,47	11,24	253	236
2			88,831		88,831		11,26		235	
3			91,033		91,033		10,99		220	
1	1000	180	48,251	47,822	48,251	47,822	20,72	20,91	540	539
2			47,593		47,593		21,01		536	
3			47,622		47,622		21,00		539	
1	1000	200	41,405	41,246	41,405	41,246	24,15	24,25	606	603
2			41,207		41,207		24,27		598	
3			41,126		41,126		24,32		604	
1	1000	220	40,173	38,438	40,173	38,438	24,89	26,04	610	626
2			37,610		37,610		26,59		632	
3			37,531		37,531		26,64		634	
1	1000	240	40,933	38,348	40,933	38,348	24,43	26,14	668	653
2			37,626		37,626		26,58		646	
3			36,484		36,484		27,41		644	
1	1000	260	37,593	36,892	37,597	36,894	26,60	27,11	638	645
2			36,762		36,762		27,20		642	
3			36,322		36,322		27,53		655	

Legenda: Tabela dos resultados do cenário B1 (cupão diferente para cada pedido com segurança completa)

Cenário B2 (cupão diferente para cada pedido com segurança parcial, só do lado da loja)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	142,684	141,578	143,789	141,946	7,01	7,067	13	17
2			142,136		142,136		7,04		12	
3			139,914		139,914		7,15		24	
1	1000	40	137,676	138,656	136,676	138,323	7,26	7,210	6	4
2			139,576		139,576		7,16		1	
3			138,716		138,716		7,21		4	
1	1000	60	136,713	136,786	136,713	137,119	7,31	7,310	5	7
2			136,852		136,852		7,31		5	
3			136,793		137,793		7,31		9	
1	1000	80	135,979	135,424	135,979	135,424	7,35	7,383	3	9
2			134,368		134,368		7,44		13	
3			135,926		135,926		7,36		9	
1	1000	100	133,150	132,524	133,150	132,524	7,51	7,547	12	13
2			132,634		132,634		7,54		10	
3			131,788		131,788		7,59		17	
1	1000	120	131,019	130,871	131,019	130,871	7,63	7,640	19	13
2			130,200		130,200		7,68		8	
3			131,393		131,393		7,61		12	
1	1000	140	130,314	131,136	130,314	126,153	7,67	7,623	5	11
2			132,737		132,737		7,53		15	
3			130,356		115,409		7,67		12	
1	1000	160	74,973	75,898	74,973	75,898	13,34	13,193	287	276
2			79,794		79,794		12,53		198	
3			72,928		72,928		13,71		342	
1	1000	180	43,082	44,148	43,082	44,148	23,21	22,737	590	571
2			47,957		47,957		20,85		553	
3			41,405		41,405		24,15		569	
1	1000	200	38,354	39,874	38,354	39,874	26,07	25,107	645	625
2			39,581		39,581		25,26		621	
3			41,686		41,686		23,99		609	
1	1000	220	37,152	39,168	37,152	39,168	26,92	25,567	655	647
2			40,112		40,112		24,93		619	
3			40,239		40,239		24,85		666	
1	1000	240	37,026	41,016	37,026	41,016	27,01	24,660	660	649
2			47,321		47,321		21,13		644	
3			38,701		38,701		25,84		641	
1	1000	260	36,711	37,331	36,711	37,331	27,24	26,810	651	651
2			36,426		36,426		27,45		667	
3			38,856		38,856		25,74		635	

Legenda: Tabela dos resultados do cenário B2 (cupão diferente para cada pedido com segurança parcial, lado da loja)

Cenário B3 (cupão diferente para cada pedido com segurança parcial, só do lado do Optipricer)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	139,045	140,724	139,045	140,724	7,19	7,11	11	11
2			141,510		141,510		7,07		12	
3			141,617		141,617		7,06		10	
1	1000	40	138,813	138,793	138,813	138,793	7,20	7,20	5	6
2			137,977		137,977		7,25		8	
3			139,589		139,589		7,16		4	
1	1000	60	137,294	137,579	137,294	137,579	7,28	7,27	13	11
2			139,725		139,725		7,16		10	
3			135,717		135,717		7,37		8	
1	1000	80	135,179	134,990	135,179	134,990	7,40	7,41	8	7
2			134,241		134,241		7,45		6	
3			135,551		135,551		7,38		7	
1	1000	100	132,378	132,963	132,378	132,963	7,55	7,52	21	12
2			133,024		133,024		7,52		9	
3			133,487		133,487		7,49		6	
1	1000	120	131,626	131,124	131,626	131,124	7,60	7,63	6	8
2			132,659		132,659		7,54		10	
3			129,088		129,088		7,75		7	
1	1000	140	130,785	130,449	130,785	130,449	7,65	7,67	8	8
2			129,841		129,841		7,70		10	
3			130,722		130,722		7,65		6	
1	1000	160	88,641	89,553	88,641	89,553	11,28	11,17	241	234
2			87,638		87,638		11,41		248	
3			92,379		92,379		10,83		211	
1	1000	180	48,876	48,483	48,876	48,483	20,46	20,63	527	532
2			48,525		48,525		20,61		532	
3			48,047		48,047		20,81		535	
1	1000	200	39,893	39,995	39,893	39,995	25,07	25,03	614	615
2			38,472		38,472		25,99		630	
3			41,620		41,620		24,03		600	
1	1000	220	38,632	38,327	38,632	38,326	25,88	26,09	627	628
2			37,911		37,910		26,38		629	
3			38,437		38,437		26,02		626	
1	1000	240	35,995	36,162	35,995	36,162	27,78	27,65	656	652
2			36,638		36,638		27,29		645	
3			35,854		35,854		27,89		654	
1	1000	260	36,220	36,433	36,220	36,433	27,61	27,45	647	648
2			36,628		36,628		27,30		643	
3			36,452		36,452		27,43		653	

Legenda: Tabela dos resultados do cenário B3 (cupão diferente para cada pedido com segurança parcial, só do lado do Optipricer)

Cenário B4 (Cupão diferente para cada pedido sem segurança implementada)

n	Pedidos	Pedidos Concorrentes	Tempo Total (seg)	Média Tempo Total (seg)	Tempo p/ pedido (ms)	Média Tempo p/ Pedido (ms)	Pedidos p/ seg	Média Pedidos p/ seg	Erros (500)	Média Erros (500)
1	1000	20	141,989	140,800	141,489	140,634	7,04	7,10	12	15
2			138,385		138,385		7,23		13	
3			142,027		142,027		7,04		18	
1	1000	40	137,443	138,392	137,443	138,392	7,28	7,23	10	7
2			137,577		137,577		7,27		6	
3			140,156		140,156		7,13		5	
1	1000	60	135,931	136,410	135,931	136,410	7,36	7,33	7	5
2			136,283		136,283		7,34		3	
3			137,017		137,017		7,30		5	
1	1000	80	135,328	134,863	135,328	134,863	7,39	7,41	7	7
2			134,358		134,358		7,44		6	
3			134,902		134,902		7,41		6	
1	1000	100	131,983	132,874	131,982	132,873	7,58	7,53	10	9
2			133,442		133,442		7,49		7	
3			133,196		133,196		7,51		8	
1	1000	120	132,183	131,980	132,183	131,980	7,57	7,58	10	9
2			132,671		132,671		7,54		10	
3			131,086		131,086		7,63		5	
1	1000	140	130,210	129,359	130,210	129,359	7,68	7,73	6	8
2			128,525		128,525		7,78		12	
3			129,342		129,342		7,73		6	
1	1000	160	86,045	86,101	86,045	86,101	11,62	11,62	253	256
2			86,169		86,169		11,61		259	
3			86,089		86,089		11,62		256	
1	1000	180	48,952	48,221	48,952	48,221	20,43	20,74	546	540
2			47,928		47,928		20,86		540	
3			47,782		47,782		20,93		534	
1	1000	200	42,091	40,901	42,091	40,901	23,76	24,50	591	603
2			38,333		38,333		26,09		625	
3			42,279		42,279		23,65		593	
1	1000	220	37,768	38,685	37,768	38,685	37,77	29,62	626	623
2			39,111		39,111		25,57		625	
3			39,177		39,177		25,53		618	
1	1000	240	37,955	37,415	37,955	37,415	26,35	26,73	629	639
2			37,041		37,041		27,00		643	
3			37,248		37,248		26,85		643	
1	1000	260	36,393	37,219	36,393	37,219	27,48	26,88	644	641
2			37,753		37,753		26,49		638	
3			37,511		37,511		26,66		639	

Legenda: Tabela dos resultados do cenário B4 (cupão diferente para cada pedido sem segurança implementada)

Apêndice D: Testes unitários

Teste	Objetivo do teste	Resultado esperado	Resultado obtido
Secure Content With Invalid Task	Analisar o comportamento da função “secureContent()” com uma <i>task</i> inválida	<i>false</i>	<i>false</i>
Secure Content For Cipher Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de cifra, comparando o valor retornado no “smode”	SECURE_CIPHER	SECURE_CIPHER
Secure Content For CipherSign Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de cifra e assinatura (conjunta), comparando o valor retornado no “smode”	SECURE_CIPHER_SIGN	SECURE_CIPHER_SIGN
Secure Content For Sign Task	Analisar o comportamento da função “secureContent()” com a <i>task</i> de assinatura, comparando o valor retornado no “smode”	SECURE_SIGN	SECURE_SIGN
Encrypt Content With Success	Analisar o comportamento da função “encryptContent()” com argumentos corretos	Is object = <i>true</i> Object has “content”, “iv”, “alg”, “smode” “smode” = SECURE_CIPHER	Is object = <i>true</i> Object has “content”, “iv”, “alg”, “smode” “smode” = SECURE_CIPHER
Encrypt Content With Key Too Big	Analisar o comportamento da função “encryptContent()”, com uma chave demasiado grande para o algoritmo escolhido	<i>false</i>	<i>false</i>
Sign Content With Success	Analisar o comportamento da função “signContent()” com argumentos corretos	Is object = <i>true</i> Object has “content”, “alg”, “smode”, “sign” “smode” = SECURE_SIGN	Is object = <i>true</i> Object has “content”, “alg”, “smode”, “sign” “smode” = SECURE_SIGN
Sign Content With Invalid Private Key	Analisar o comportamento da função “signContent()”, com uma chave privada inválida	<i>false</i>	<i>false</i>
Encrypt Sign Content With Success	Analisar o comportamento da função “encryptSignContent()” com argumentos corretos	Is object = <i>true</i> Object has “content”, “iv”, “alg”, “smode”, “sign” “smode” = SECURE_CIPHER_SIGN	Is object = <i>true</i> Object has “content”, “iv”, “alg”, “smode”, “sign” “smode” = SECURE_CIPHER_SIGN
Encrypt Sign Content With Sharedkey Too Big	Analisar o comportamento da função “encryptSignContent()” com a chave partilhada demasiado grande para o algoritmo escolhido	<i>false</i>	<i>false</i>
Encrypt Sign Content	Analisar o comportamento da função	<i>false</i>	<i>false</i>

With Invalid Private Key	“encryptSignContent()” com chave privada incorreta		
Decrypt Content With Success	Analisar o comportamento da função “decryptContent()” com argumentos corretos	“This content is valid and secret!”	“This content is valid and secret!”
Decrypt Content With Invalid Content	Analisar o comportamento da função “decryptContent()” quando o conteúdo não é o esperado	<i>false</i>	<i>false</i>
Decrypt Content With Invalid Key	Analisar o comportamento da função “decryptContent()”, quando a chave para decifrar não é a correta	<i>false</i>	<i>false</i>
Decrypt Content With Violated Encrypted Content	Analisar o comportamento da função “decryptContent()”, quando o conteúdo cifrado (“content”) foi adulterado	<i>false</i>	<i>false</i>
Decrypt Content With Violated IV	Analisar o comportamento da função “decryptContent()”, quando o IV foi adulterado	<i>false</i>	<i>false</i>
Decrypt Content With Wrong Algorithm	Analisar o comportamento da função “decryptContent()”, quando utilizado um algoritmo incorreto	<i>false</i>	<i>false</i>
Decrypt Content With Invalid Algorithm	Analisar o comportamento da função “decryptContent()”, quando utilizado um algoritmo inválido (inexistente)	<i>false</i>	<i>false</i>
Verify Signature With Success	Analisar o comportamento da função “verifySignature()” com argumentos corretos	<i>true</i>	<i>true</i>
Verify Signature With Invalid Content	Analisar o comportamento da função “verifySignature()” com conteúdo inválido	<i>false</i>	<i>false</i>
Verify Signature With Violated Content	Analisar o comportamento da função “verifySignature()” com “content” manipulado	<i>false</i>	<i>false</i>
Verify Signature With Violated Signature	Analisar o comportamento da função “verifySignature()” com a assinatura inválida	<i>false</i>	<i>false</i>
Verify Signature With Invalid Public Key	Analisar o comportamento da função “verifySignature()” com chave pública inválida	<i>false</i>	<i>false</i>
Verify Signature	Analisar o comportamento da função “verifySignature()”	<i>false</i>	<i>false</i>

With Wrong Public Key	com chave pública errada (não corresponde à chave privada)		
Verify Signature With Invalid Algorithm	Analisar o comportamento da função “verifySignature()” com algoritmo inválido (inexistente)	<i>false</i>	<i>false</i>
Verify Signature With Wrong Algorithm	Analisar o comportamento da função “verifySignature()” com algoritmo errado	<i>false</i>	<i>false</i>
Get Content With Success For Signed Content	Analisar o comportamento da função “getContent()” para conteúdo assinado, com argumentos corretos	“This content is valid and secret!”	“This content is valid and secret!”
Get Content With Success For Encrypted Content	Analisar o comportamento da função “getContent()” para conteúdo cifrado, com argumentos corretos	“This content is valid and secret!”	“This content is valid and secret!”
Get Content With Invalid Content	Analisar o comportamento da função “getContent()” com conteúdo inválido	<i>false</i>	<i>false</i>
Get Content With Invalid SMODE	Analisar o comportamento da função “getContent()”, com “smode” inválido	<i>false</i>	<i>false</i>
Get Content With Wrong SMODE	Analisar o comportamento da função “getContent()”, com “smode” errado (ex: decifra com “smode” = SECURE_SIGN)	<i>false</i>	<i>false</i>
Decrypt And Verify Content With Success	Analisar o comportamento da func. “decryptVerifyContent()” com argumentos corretos	“This content is valid and secret!”	“This content is valid and secret!”
Decrypt And Verify Content With Invalid Content	Analisar o comportamento da func. “decryptVerifyContent()” com conteúdo inválido	<i>false</i>	<i>false</i>
Decrypt And Verify Content With Invalid Alg	Analisar o comportamento da func. “decryptVerifyContent()” com algoritmo inválido	<i>false</i>	<i>false</i>
Decrypt And Verify Content With Violated Content	Analisar o comportamento da func. “decryptVerifyContent()” com “content” manipulado	<i>false</i>	<i>false</i>
Decrypt And Verify Content With	Analisar o comportamento da func. “decryptVerifyContent()” com assinatura manipulada	<i>false</i>	<i>false</i>

Violated Signature			
Decrypt And Verify Content With Invalid Shared Key	Analisar o comportamento da func. “decryptVerifyContent()” com chave compartilhada inválida	<i>false</i>	<i>False</i>
Decrypt And Verify Content With Invalid Public Key	Analisar o comportamento da func. “decryptVerifyContent()” com chave pública inválida	<i>false</i>	<i>false</i>
Decrypt And Verify Content With Invalid Public Key	Analisar o comportamento da func. “decryptVerifyContent()” com chave pública errada (não corresponde à chave privada)	<i>false</i>	<i>false</i>
Get Content With Success For Encrypted Signed Content	Analisar o comportamento da função “getContent()”, com argumentos corretos, para “smode” = SECURE_CIPHER_SIGN	“This content is valid and secret!”	“This content is valid and secret!”

Legenda: Tabela com os resultados dos testes unitários efetuados sobre a nossa biblioteca