FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Next-generation JavaScript Licensing Enforcing Techniques

Pedro José Leite da Cunha Melo Alves

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Rui Maranhão

Junho de 2015

Next-generation JavaScript Licensing Enforcing Techniques

Pedro José Leite da Cunha Melo Alves

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Doutor Raul Fernando de Almeida Moreira Vidal
Arguente: Doutor Manuel Eduardo Carvalho Duarte Correia
Vogal: Doutor Rui Filipe Lima Maranhão de Abreu
17 de Julho de 2015

Resumo

A contínua partilha, visualização e execução de conteúdos digitais de forma ilegal é um problema bem conhecido no panorama internacional. Conteúdos comerciais são partilhados sem qualquer tipo de controlo, e modificados para ultrapassar barreiras, impostas pelos produtores, ao acesso não autorizado, degradando assim a confiança entre os produtores e os seus consumidores. Além da degradação da confiança, é também reduzido o valor, cultural e monetário, que um conteúdo representa no mercado. Para evitar o uso indevido de conteúdo são então implementados mecanismos de aplicação de licenças, que garantem que o Consumidor só possa interagir com o conteúdo nos conformes da licença, associada ao conteúdo no seu processo de distribuição.

A linguagem de programação *JavaScript* é cada vez mais usada para a criação de *software* para a *Web* ou *Mobile*, no entanto, a maioria da investigação para proteger o conteúdo é focada no contexto dos ataques maliciosos, como modificação ou injecção de código. É necessário por isso criar paralelismos com mecanismos de aplicação de licenças, implementados em outras tecnologias, retirar os fundamentos básicos do licenciamento, e aplicar estas informações para esta tecnologia, possibilitando aos produtores protegerem os seus interesses, aumentando também a confiança com os consumidores.

Neste projeto é pretendido a criação de uma solução que permita aplicar licenças a conteúdo *JavaScript*, que deve garantir que um potencial utilizador só possa executar ou partilhar quando assim lho é permitido, envolvendo validação do ambiente de execução, verificação do tempo de utilização, entre outros fatores. É significativo também que a solução unifique a proteção de conteúdo para várias plataformas, móveis ou não móveis, e tenha em consideração o uso de técnicas que reduzam o custo para o Produtor e para o Consumidor. Com efeito, espera-se que esta solução seja inovadora no âmbito tecnológico atual por produzir uma ferramenta que abrange aspetos pouco explorados em *JavaScript*.

Abstract

The illegal continuous sharing, visualization and execution of digital content it's a well known problem on digital market. Commercial assets are shared without any kind of control, and are modified to break constraints, that are enforced by producers, to the unauthorized access, degrading the confidence between producers and their costumers. In addition to the degradation of confidence, the commercial and cultural value of an asset, in the market, is decreased. To avoid the unauthorized use of content, mechanisms of licensing enforcing are implemented, to assure that a Costumer can only interact with a content if he follows the license, that is associated with the content in his distribution process.

JavaScript programming language is becoming more and more used to the development of Web and Mobile software, however, the majority of research in JavaScript security is focused in malicious attacks, as code injection or tampering. For this reason, it's necessary to create parallelisms with license enforcing mechanisms, from other technologies, study the basics of licensing, and apply this knowledge to JavaScript, hence giving to producers the power to protect their assets and increasing the confidence with costumers too.

In this project it's intended the creation of a solution that allows licensing enforcing to JavaScript content, assuring that an user can only execute or share if he has the rights too, including execution environment validation and time of use verification. It's also important that the solution unifies the protection to multiple platforms, mobile or non-mobile, and the use of techniques that reduce the cost to producers and costumers is regarded. Therefore, it's expected that the final solution will be cutting-edge in this context by giving a tool that explores new areas on JavaScript.

Agradecimentos

Ao Professor Rui Maranhão e à *JScrambler S.A.*, principalmente ao Filipe Silva e Pedro Fortuna, pelo apoio prestado nos momentos onde a dúvida se instaurou.

À minha família e amigos, nos bons e menos bons momentos, pela diversão ou ajuda que proporcionaram, não só no decorrer deste trabalho, mas também nestes cinco anos que passaram.

Pedro Alves

Conteúdo

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Abreviaturas e Símbolos

ABE	*Attribute Based Encryption*
AES	*Advanced Encryption Standard*
AIR	*Adobe Integrated Runtime*
AJAX	*Asynchronous Javascript and XML*
CSS	*Cascading Style Sheets*
DMCA	*Digital Millennium Copyright Act*
DRM	*Digital Rights Management*
FPS	*Frames Per Second*
GPS	*Global Positioning System*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JADE	*JAVA Agent DEvelopment*
JSON	*JavaScript Object Notation*
MAC	*Media Access Control*
PKE	*Public Key Encryption*
PKI	*Public Key Infrastructure*
REST	*Representational State Transfer*
SDK	*Software Development Kit*
TPM	*Trusted Platform Module*
USB	*Universal Serial Bus*
UUID	*Universally Unique Identifier*
XAML	*eXtensible Application Markup Language*

Capítulo 1

Introdução

Para combater a fraude eletrónica, como aceder a conteúdo sem ter autorização, é necessário haver mecanismos que impeçam ou tornem mais difícil para o infrator atingir os seus fins. A aplicação de licenciamento é um desses mecanismos, caraterizando-se por permitir à identidade autora, associar uma licença a um determinado conteúdo. Uma licença é uma estrutura que contém informação como políticas de utilização, chaves de desencriptação ou até estipulação do período de utilização do conteúdo [LHJ+03]. As políticas de utilização definem os direitos que o Consumidor necessita de ter para poder aceder ou partilhar um conteúdo, e geralmente são definidas por linguagens próprias, devidamente criadas para a especificação dos direitos [Eri03].

Com efeito, é de cariz essencial que a aplicação das licenças se faça devidamente junto do Consumidor, protegendo os autores de utilização incorreta dos seus produtos e garantindo que o Consumidor tenha acesso a conteúdo autorizado, permitindo também que este não seja lesado por aceder corretamente, por exemplo através de uma compensação monetária ao Autor, enquanto outro Consumidor possa utilizar ilegalmente sem consequências. Neste projeto a aplicação das licenças é direcionada para a linguagem de programação *JavaScript*, cada vez mais emergente nas tecnologias referentes à *Web*, fazendo-se um estudo de novas perspetivas e métodos de licenciamento para *JavaScript*. Este documento visa expor esse estudo, analisando métodos já existentes e especificando uma solução que englobe os objetivos a propor. O presente capítulo apresenta o contexto, motivação e objetivos da dissertação, juntamente com detalhes sobre a estrutura do documento.

1.1 Contexto

Para uma parte dos consumidores é aceitável que conteúdo pertencente a uma identidade seja acedido e partilhado indevidamente sem qualquer punição, onde 42% das transações *online* têm como motivo a prática de pirataria [Sud13]. Embora existam leis, como a DMCA nos Estados Unidos da América [And06], que detalham o que é permitido, do ponto vista legal, aceder, partilhar

ou modificar, existem sempre infratores que não se regem por estas leis. Como tal, é necessário que outros tipos de proteção, mais ativos, sejam postos em funcionamento. Uma solução adotada pelos produtores de conteúdo digital é o DRM, que se traduz no nome dado a qualquer técnica ou ferramenta que aplique os direitos de utilização a um consumidor [SY06] e que tem como parte integrante o processo de aplicação de licenças. Do ponto de vista de um produtor o uso de DRM seria o ideal para o seu modelo de negócio caso não afetasse o Consumidor, no entanto este sistema envolve normalmente operações adicionais numa transação, como ativação *online* ou registo de utilizador, tornando-se um processo mais moroso para um consumidor honesto, só para garantir que o conteúdo é utilizado de forma autorizada. É de notar que este aspeto não influencia só os consumidores, os próprios produtores podem ser prejudicados por um decréscimo de potenciais clientes, não complacentes com as operações ou restrições adicionais que o DRM impõe, havendo grande importância no balanceamento do custo que este exige tanto para o Consumidor como para o Produtor, e sendo necessária a exploração de novas formas de não repudiar consumidores enquanto se garante que a utilização do conteúdo é feita nos conformes estipulados. Esta dissertação surge no âmbito de uma proposta efetuada pela empresa *JScrambler S.A.* em parceria com a FEUP, é pretendido que se faça uma exploração dos sistemas de aplicação de licenças existentes e que se crie paralelismos, de forma a que novos mecanismos que apliquem as políticas de utilização sejam implementados, tendo sempre em conta as necessidades dos produtores e consumidores.

1.2 Motivação

Embora existam muitas soluções sobre como implementar um sistema que aplique licenciamento, é ainda pouco explorado este campo no domínio do *JavaScript*. Só recentemente tem-se notado uma evolução no uso do *JavaScript* para além da utilização em páginas *Web* em *browsers*, quer pelo aparecimento de *frameworks* para criação de aplicações *Web*, quer por novas possibilidades de utilizar a linguagem para programar conteúdo para várias plataformas, como para *Android* ou *Windows Phone*. Este projeto foca na especificação e implementação de uma solução, para ambiente *JavaScript*, que integre os princípios do DRM. A esta é esperado que aplique licenças de utilização de forma segura, e que unifique os princípios de DRM para diferentes plataformas onde o conteúdo possa ser acedido.

1.3 Objetivos

Este projeto tem como objetivo principal a criação de uma nova solução *JavaScript*, que permita que conteúdo nessa linguagem seja protegido, na vertente das políticas de utilização associadas ao mesmo. Não obstante, podem-se identificar alguns objetivos mais particulares, que ajudam à criação da solução, e algumas das suas características principais da solução:

- Explorar técnicas de aplicação de licenças e criar paralelismos

- Unificar métodos de licenciamento para diferentes plataformas onde código *JavaScript* é acedido
- Balancear o custo para um Produtor e o Consumidor numa aplicação protegida
- Criação de uma solução que permita ao Produtor especificar políticas de utilização para o seu conteúdo

1.4 Estrutura do Documento

O presente documento está repartido em 5 capítulos, o primeiro com a Introdução, o segundo sobre a Revisão Bibliográfica, a Especificação e Implementação da Solução como terceiro, o quarto acerca dos Resultados e por fim, como último capítulo, as Conclusões e Trabalho Futuro. No capítulo 1, o atual, é explicado "o porquê'' da realização dissertação, havendo um enquadramento desta no contexto tecnológico, apresentado-se também os objetivos pretendidos. No capítulo 2 são exploradas as técnicas de licenciamento e soluções que aplicam licenciamento, havendo também uma revisão das plataformas em que o *JavaScript* é utilizado. No capítulo 3 é feita uma especificação da arquitetura da solução proposta, assim como discutidos pontos importantes sobre interações existentes, o seu funcionamento e implementação. No capítulo seguinte, o 4, são apresentados os principais resultados da solução, e são discutidas algumas conclusões retiradas a partir dos mesmos. No capítulo 5 é feito um resumo do trabalho realizado e analisados vários pontos passíveis de serem implementados como trabalho futuro.

Capítulo 2

Revisão Bibliográfica

No presente capítulo são exploradas áreas de interesse que sejam um apoio à construção da solução pretendida. Primeiramente são estudadas e discutidas algumas arquiteturas e técnicas de licenciamento, mais propriamente no campo de DRM, e tiradas conclusões através de uma comparação entre as funcionalidades oferecidas por elas. Depois é feita enumeração de plataformas e tecnologias onde o *JavaScript* é utilizado, e analisado o que cada uma dispõe para a implementação das tecnologias retiradas na primeira secção. Por fim, é feito um resumo da secção, para sumarizar os pontos mais importantes.

2.1 Digital Rights Management

DRM é o termo utilizado para descrever mecanismos que protejam a propriedade intelectual de conteúdos. Com o crescimento do mercado *online*, seja de qualquer tipo de conteúdo, é do interesse dos autores procurarem soluções que garantam que as condições de utilização dos seus produtos sejam respeitadas pelos consumidores [DJ11]. Surge então a necessidade de criarem-se soluções DRM, que sejam resistentes a ataques de consumidores maliciosos, e que permitam aos autores definirem os seus direitos e condições de utilização. O termo DRM pode ser considerado como sendo um pouco vago, pois abrange todo o tipo de mecanismos de proteção de ativos, seja *watermarking*, *tamper-proofing*, ofuscação ou encriptação, cada um com o seu potencial para a proteção. Nesta secção é analisado um sistema tradicional DRM, o mais frequente nas implementações DRM, com foco nas entidades essenciais que participam numa transação, e são apresentadas algumas técnicas que se baseiam ou distinguem de um sistema tradicional, e que apliquem licenciamento de uma forma ativa.

2.1.1 Sistema DRM Tradicional

No geral, um sistema DRM é composto por 3 sistemas no mínimo:

- Produtor : Aquele que é responsável por produzir o conteúdo e associar permissões ao mesmo. Estas permissões são depois necessárias para restringir o acesso ao conteúdo pelo utilizador final.
- Publicador : Encripta o conteúdo e faz o empacotamento necessário. As operações de distribuição e gestão de chaves de acesso ao conteúdo são feitas por ele. Pode ser responsável pela distribuição.
- Consumidor : Cliente final ou intermediário, contacta o distribuidor para aceder ao conteúdo, onde através de técnicas de DRM tem acesso ao produto pronto a ser consumido.

Uma licença é um conjunto de informações que associam permissões de utilização a um dado conteúdo a um utilizador [LHJ+03].

Quase todos os sistemas DRM são compostos, no mínimo, pelos 3 sistemas acima indicados. Uma arquitetura tradicional de DRM é exposta em [LHJ+03, LSnS03, PBTG11], onde os sistemas interagem entre si para criar uma solução de proteção de conteúdo. O Produtor encripta o conteúdo utilizando uma chave gerada por si, que normalmente é única por cada conteúdo, e entrega-o ao Distribuidor, enviando as permissões de utilização e a chave de desencriptação, que constituem a licença, para um intermediário responsável pelas transações de compra - o Intermediário de Compra, ou desempenhando o Distribuidor o papel de Intermediário. Na Figura 2.1 pode-se observar as interações neste sistema e as etapas na distribuição e acesso do conteúdo.

Quando o Consumidor acede ao conteúdo encriptado através do Distribuidor contacta também o Intermediário de Compra para ter acesso à licença, que realiza uma validação da identidade do Consumidor, por exemplo por meio de um certificado digital, e caso esta seja válida envia a licença para o Consumidor. No dispositivo do Consumidor encontra-se um cliente DRM que garante que as permissões especificadas na licença sejam aplicadas e que desencripta o conteúdo para a sua utilização normal.

Na maioria dos mecanismos de DRM a identidade do Consumidor está de alguma forma associada às transações efetuadas no processo de aceder a um conteúdo, por informações do Consumidor ou dispositivos deste. Estas informações não são normalmente escondidas do Publicador ou Distribuidor, quebrando alguma da confiança do Consumidor nestas identidades por terem a informação do que o Consumidor acedeu, quando acedeu, e onde acedeu. Algumas soluções com grande foco na garantia da privacidade do utilizador em transações de DRM são exploradas em [HLJ+14, YLL14, SLYK09, FZ08] , sendo a sua principal característica dissociar as informações do Consumidor do conteúdo adquirido, por exemplo por meio de *tickets* [SLYK09], que o Consumidor adquire para fazerem de ponte na comunicação entre o Distribuidor e o Consumidor, servindo estes para a validação da identidade do Consumidor.

2.1.2 Técnicas

Em DRM a maioria das técnicas baseiam-se num sistema DRM tradicional, no entanto podem-se encontrar várias que se distinguem deste e que forçam a que o licenciamento seja aplicado de

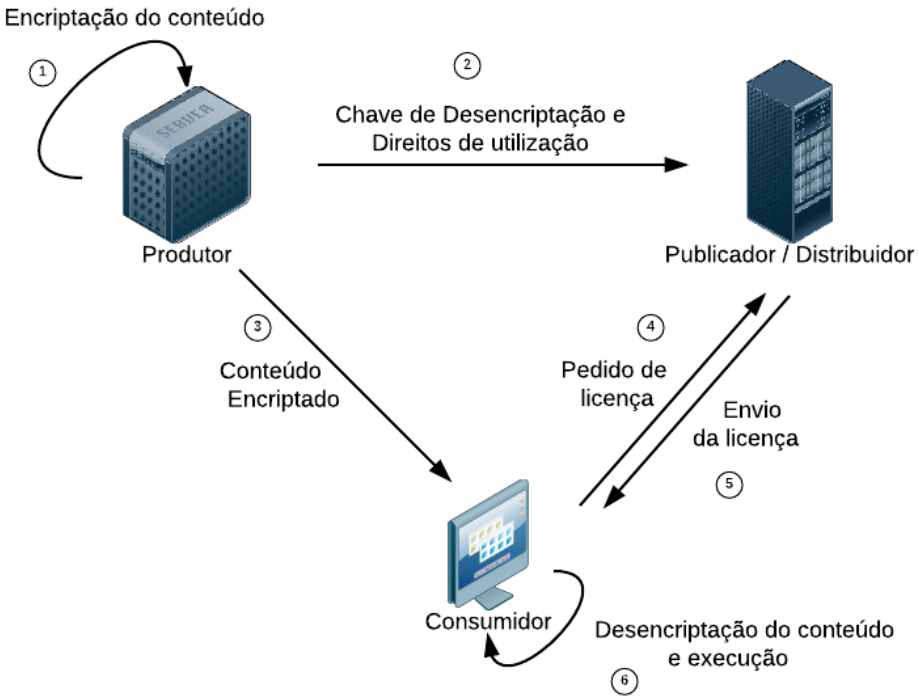

Figura 2.1: Sistema Tradicional DRM

formas distintas. Nesta secção estão as técnicas estão separadas em baseadas em criptografia e em *hardware*. Embora exista esta separação, não implica que esta seja muito restrita, algumas técnicas baseiam-se mais em criptografia mas podem necessitar de recursos a nível de *hardware* e *vice-versa*.

2.1.2.1 Sistemas baseados em criptografia

- **Public Key Encryption**

  Numa arquitetura PKI, que utiliza PKE, existe uma entidade encarregue de assegurar a confiança no sistema entre todas as entidades, através da criação de certificados que confirmam as suas identidades. Esta confiança é essencial para que ataques *Man-in-the-middle*, onde entidades maliciosas podem monitorar ou alterar informação fazendo-se passar por outras entidades, não possam ser realizados, podendo todos os intervenientes no sistema estar seguros que estão a comunicar com as entidades autorizadas.

  O sistema [VS11] especifica dois processos para aceder a conteúdo, um completamente *online* e outro com uma parte *offline*, assente numa arquitetura PKI. No modo completamente *online* o Consumidor adquire o conteúdo, encriptado com uma chave simétrica, pelo Distribuidor e cada vez que quer acede-lo pede a um servidor DRM a licença, que contém a chave simétrica para desencriptar e os direitos de utilização, através de um canal seguro de comunicação. Um cliente DRM na máquina do Consumidor desencripta em memória e aplica os direitos de utilização. Mesmo com uma vertente *offline*, o segundo modo implica a que o Consumidor esteja *online* pelo menos uma vez. A Figura 2.2 apresenta a sequência de comunicões entre as diversas entidades, na situação de um pedido de acesso a conteúdo para consumo *offline* ser feito. Quando o Consumidor autentica-se no sistema são gerados certificados *X.509* para a sua identidade e para a máquina que irá executar o conteúdo, incluindo a geração de chaves públicas e privadas. Depois de adquirir o conteúdo pelo Distribuidor, como no modo *online*, o Consumidor precisa de contactar o servidor DRM para obter a licença, que encripta a licença com a chave pública que consta no certificado do Consumidor e entrega-a ao Consumidor. O cliente DRM na máquina do Consumidor desencripta a licença com a chave privada do Consumidor e compara a identidade da máquina com o certificado desta, desencriptando o conteúdo com a chave que consta na licença e forçando os direitos de utilização. O sistema descreve ainda as medidas a tomar quando se quer que a licença seja estendida e mudança de máquina.

  Outros sistemas baseados em PKI foram introduzidos, com alterações para diferentes tipos de conteúdo ou com combinação com outras técnicas, [ZL11] mostra um sistema PKI adaptado a um contexto multimédia, com a especificação de reprodutor DRM para o conteúdo, em [US11] uma solução com recurso a *hardware*, que serve como *token*, é definida, e [LZ13] implementa um sistema com base no formato *PKCS#12*, que define uma arquitetura PKI com encriptação da própria informação criptográfica, tal como encriptação de chaves privadas e certificados.

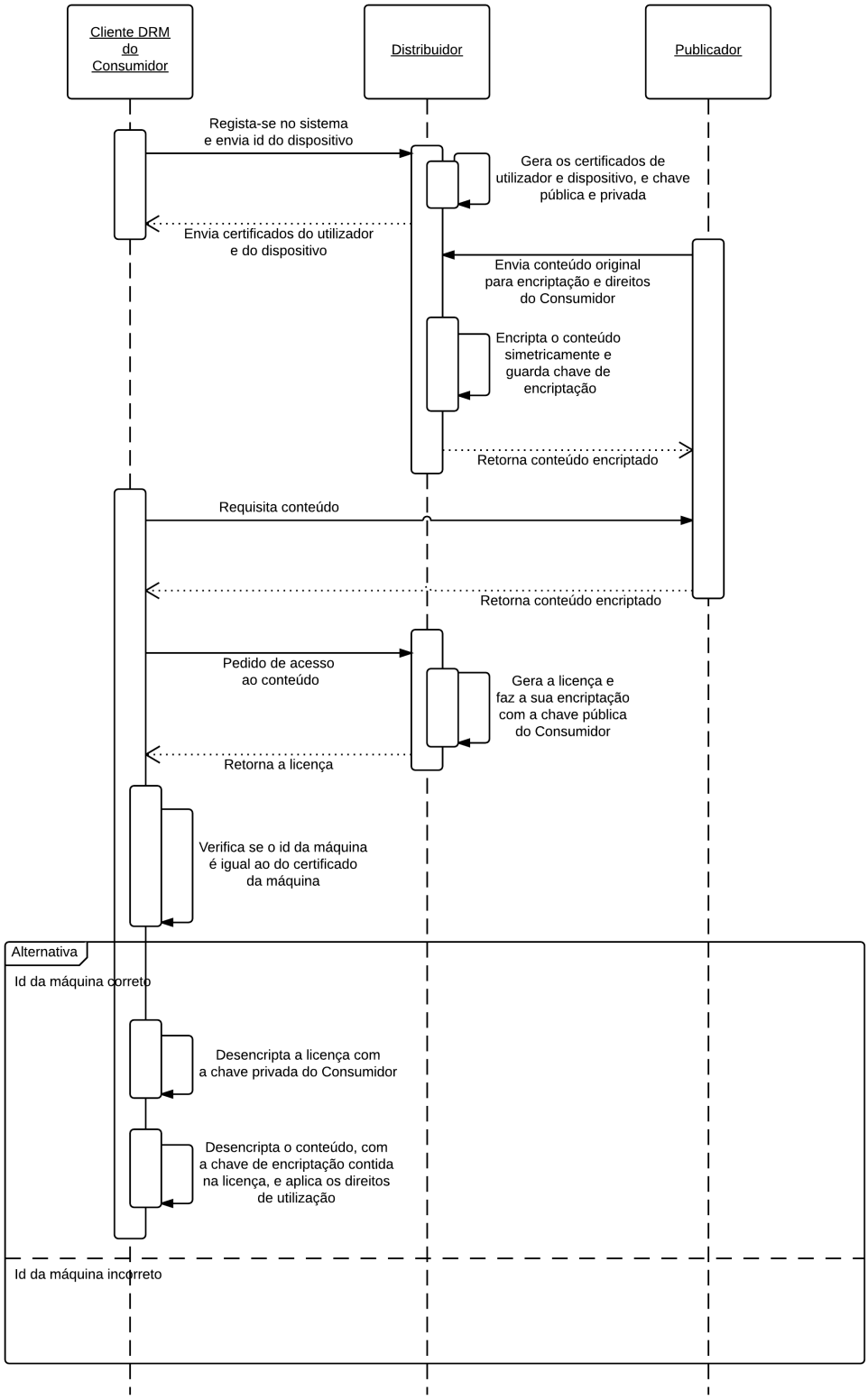

Figura 2.2: Sistema baseado em PKE

- **Attribute Based Encryption**

  Em [GVS13] é apresentado o *AtDRM*, uma solução onde a licença é associada ao Consumidor e não ao dispositivo deste, através de uma encriptação baseada em atributos - ABE. Na Figura 2.3 é representado o protocolo de comunicação para o acesso a conteúdo, que é encriptado pelo Produtor usando uma chave simétrica *Ks* gerada previamente. Uma política de utilização é criada através da especificação dos atributos que o Consumidor deve ter para poder utilizar o conteúdo, por exemplo *email*, trabalho ou morada. Esta política é então utilizada para criar uma chave pública *Kp* baseada no ABE, com o propósito de encriptar a chave simétrica *Ks*, criando uma chave pública *Kp*. O conteúdo encriptado com *Ks* e a chave simétrica *Ks*, encriptada com *Kp*, são então enviados para o Distribuidor que é responsável por efetuar o resto do processo de distribuição com o Consumidor. O Consumidor autentica-se com o Produtor por um cliente DRM, fazendo um pedido para ter acesso ao conteúdo, recebendo uma versão encriptada do conteúdo e a chave simétrica *Ks* encriptada. O Publicador verifica a existência de um atributo *dummy*, que traduz-se no acesso ou não ao conteúdo, onde caso exista, o Publicador gera uma chave privada *KpU* para o Consumidor com recurso aos atributos deste. Ao cliente DRM do Consumidor é transmitido *KpU* e as permissões de utilização. Se as permissões forem corretamente verificadas o cliente utiliza a chave *KpU* para a desencriptação de *Ks*, onde só se um número mínimo (existente na encriptação de *Ks*) de atributos do Consumidor , codificados em *KpU*, estiverem de acordo com a política de utilização que encripta *Ks*, é que *Ks* e o conteúdo, através de *Ks*, podem ser desencriptados.

  O *AtDRM* permite ainda a partilha do conteúdo por múltiplos dispositivos, pois como no *AtDRM* as licenças são baseadas na identidade do Consumidor e não dos dispositivos é fácil realizar a partilha de conteúdo, bastando cada dispositivo do Consumidor ter instalado o cliente DRM e estar de acordo com as permissões da licença, podendo haver o caso onde a licença não permita o acesso num dispositivo específico.

  Esta arquitetura descreve também um sistema de revogação de direitos de um Consumidor ou grupo destes, pois havendo uma possível organização hierárquica de Consumidores no servidor de DRM, basta alterar o atributo *dummy* para cada nível de hierarquia, que revoga o acesso à geração das chaves privadas para os Consumidores no caso de um nível hierárquico inteiro, ou para um só Consumidor. Para a transferência de direitos basta o Consumidor indicar para quem quer transferir os direitos de utilização e o Publicador trata de revogar os direitos de um Consumidor e gerar e distribuir a chave privada para o Consumidor em que são transferidos.

- **Chave de Produto**

  Um sistema de validação das permissões de utilização do conteúdo através de chaves de produtos, onde, por norma, as chaves são *strings* que garantem acesso a um conteúdo desde que se encontre numa lista de chaves, previamente gerada e aleatória.

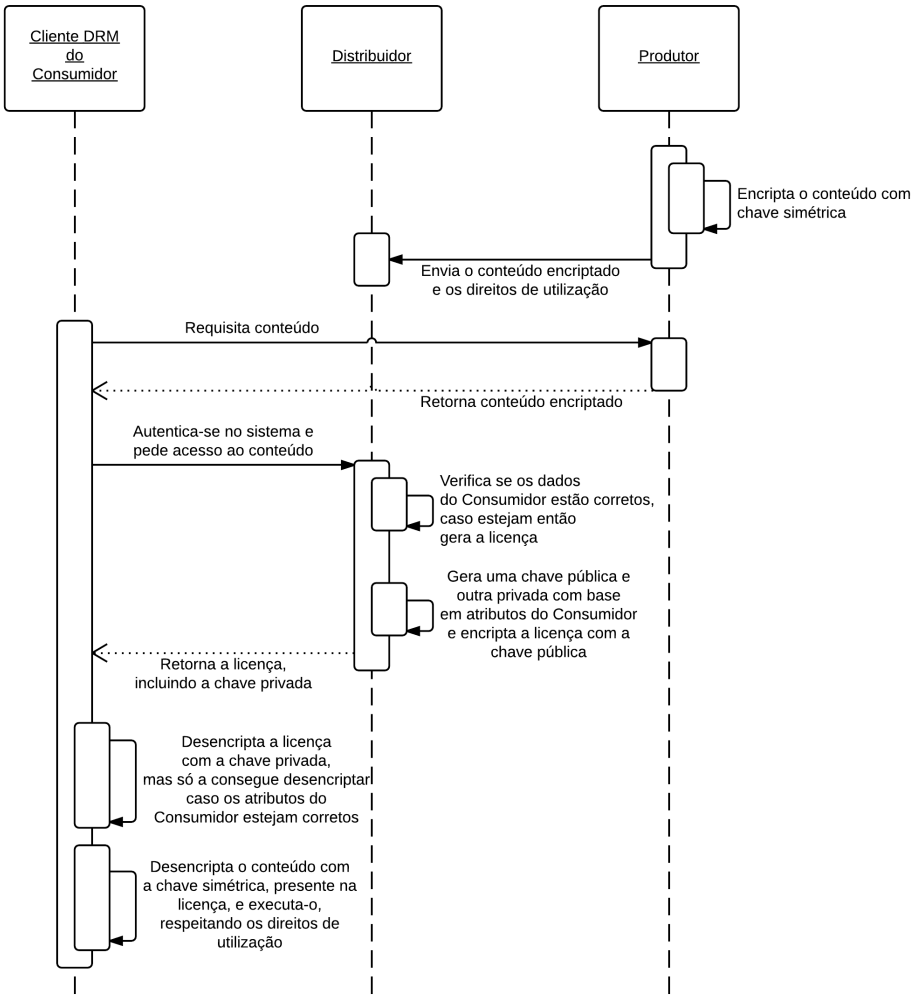

Figura 2.3: Sistema ABE

A solução exposta em [HLJ+14] resolve o problema de que desde que um Consumidor tenha acesso a uma chave que esteja na lista pode utilizá-la para se autenticar, não havendo possibilidade de saber se a chave foi usada na autenticação de outro Consumidor. Na solução, o Consumidor regista-se no sistema, do Produtor ou Distribuidor (pode não haver diferenciação entre os dois), durante a instalação do conteúdo, e autentica-se com o dados de registo transmitindo ao servidor DRM uma *hash* gerada através de identificadores de componentes *hardware* do dispositivo onde se encontra, como *serial numbers* do processador, disco rígido ou *motherboard*. Esta *hash* é utilizada então para criar uma chave de produto única, que só pode ser utilizada num dispositivo com os componentes transmitidos, podendo o Consumidor reutilizar o conteúdo, desde que no dispositivo de registo. O processo de atualização do conteúdo também é abordado por esta solução. Um diagrama com as etapas e entidades desta técnica é apresentado na Figura 2.4.

Em [ZS12] é descrita uma técnica que se baseia nos princípios tradicionais de gerar chaves aleatórias, mas que apresenta alterações ao nível da confirmação das chaves durante a tentativa de acesso ao conteúdo. Segundo esta, repartir as operações de introdução da chave pelo Consumidor e de confirmação desta por várias *threads* torna, para o atacante, muito mais difícil o ato de descobrir chaves para aceder ao conteúdo através de *debugging*.

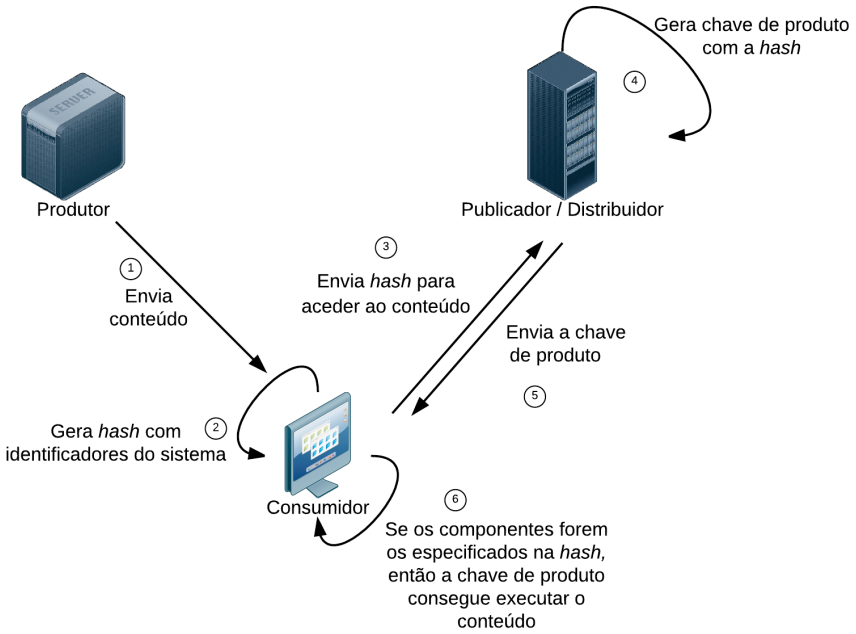

Figura 2.4: Sistema Chave de Produto

- **Mobile Code**

Mobile Code é a designação dada para a capacidade de num sistema ser possível executar fragmentos de código sem uma estrita ligação com a localização de onde são executados [FPV98]. Os fragmentos de código são englobados num programa designado de agente e o estado desse programa pode ser alterado em várias localizações, persistindo a informação sobre o seu estado em cada execução. Num sistema DRM com este tipo de tecnologia, o agente tem a função de desencriptar o conteúdo no dispositivo do Consumidor, recolhendo informação de direitos de utilização de conteúdo e de autenticação nos sistemas que visita antes de chegar ao Consumidor.

No artigo [LLZL08] um novo esquema de DRM é apresentado, com principal novidade a utilização de *Mobile Code* no processo de DRM tradicional. Nesta arquitetura, a licença, que contém a chave de desencriptação de conteúdo, é incorporada num agente, pelo Distribuidor. A autenticação entre Distribuidor e Consumidor é realizada seguindo uma arquitetura PKI, em que no caso da identidade do Consumidor seja validada, o agente é transferido para o dispositivo do Consumidor e este é executado no cliente DRM para se aceder ao conteúdo, caso as permissões de utilização contidas no agente assim o autorizem. O agente é responsável por dar ao cliente DRM as informações de desencriptação e os direitos de utilização, sendo transferido de volta para o Distribuidor após a desencriptação estar concluída. A Figura 2.5 mostra as etapas que o agente realiza nesta técnica. A segurança do agente é garantida encapsulando o agente original num novo agente, o *Time Limited BlackBox Agent*, onde por um determinado tempo não é possível analisar ou modificar a estrutura do original, e em que após esse tempo, para um atacante hipotético, o agente original já não tem qualquer uso. Este esquema possibilita também a redistribuição de acesso ao conteúdo, um Consumidor *C1* que queira aceder a um conteúdo pertencente a outro Consumidor *C2*, faz o pedido de acesso ao conteúdo a *C2*, que transmite-o ao Distribuidor. O Distribuidor fica encarregue de alterar as permissões de utilização do conteúdo na sua base de dados, ficando *C1* sem poder aceder novamente ao conteúdo e *C2* com as novas permissões.

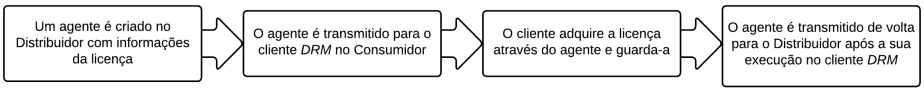

Figura 2.5: Sistema *Mobile Code Black-Box*

Para combater algumas das deficiências de [LLZL08], que apresenta o seu sistema de segurança do agente de uma forma geral e não pormenorizada, em [GPB12] uma nova solução é mostrada, recorrendo a técnicas de encriptação *White-Box*. Na solução anterior o código de desencriptação era protegido através de um processo de *Black-Boxing* do agente que continha o código, na nova solução é tido como pressuposto que o atacante conseguiu entrar no sistema com sucesso e tem acesso à leitura dos dados. Técnicas de encriptação por *White-Box*, onde algoritmos que dificultam engenharia reversa são usados, fazem parte

da encriptação do conteúdo no Produtor, gerando também um código de desencriptação simétrico. Outra diferença em relação a [LLZL08] é que o agente é transferido desde logo para o Consumidor quando recebe o conteúdo encriptado, no entanto este agente não contém qualquer parte do código de desencriptação, que é enviado por seu turno do Produtor para o Distribuidor. Sempre que o Consumidor quiser aceder ao conteúdo o agente verifica localmente a identidade do Consumidor, não sendo necessário o uso de certificados, e caso seja válido contacta o Distribuidor transferindo o agente para este. O Distribuidor injeta no agente o código de desencriptação e a licença de utilização, transmitindo o agente para o dispositivo do Consumidor que trata da desencriptação e aplicação das permissões especificadas na licença, como se pode verificar na Figura 2.6.

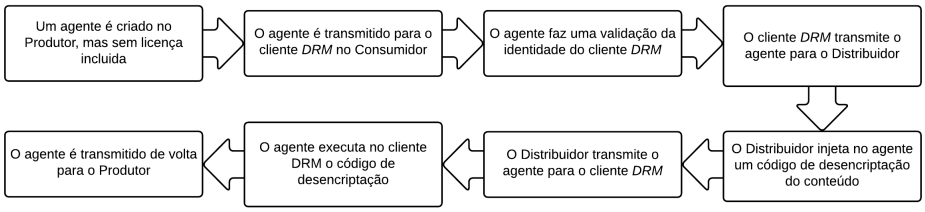

Figura 2.6: Sistema *Mobile Code White-Box*

- **Verificação da licença com os seus detalhes físicos**

  Uma perspetiva diferente, com funcionalidade *offline* sem requerer ligação continua à *Internet* para aceder ao conteúdo, é abordada em [QYTC13]. Esta solução é capaz de combater *replay attacks* [Syv94], isto é, um atacante não pode utilizar uma certa licença se os parâmetros do ambiente não se encontrarem em conformidade com os da licença, por exemplo, numa aplicação com versão de experimentação de 30 dias, um atacante não pode utilizar um estado da licença de quando ainda faltavam 29 dias para substituir a licença que restringe o acesso ao conteúdo após os 30 dias passarem. Na Figura 2.7 é exposto o processo que o cliente DRM efetua quando uma nova licença é pedida para um conteúdo.

  O mecanismo pode ser sintetizado num mapeamento da licença para um ficheiro *Log*, que contém também uma *hash* da localização física (célula no setor de dados, inicial e final) de 3 ficheiros *dummy* de tamanho arbitrário. Sempre que existe uma operação de criação, atualização ou remoção de licenças o cliente DRM substitui os 3 ficheiros *dummy* por outros, alterando a localização física e atualiza a *hash* e as informações da licença no *Log*. Mesmo que um atacante substituísse o ficheiro *Log* para o referente a uma licença maliciosa, a localização dos ficheiros *dummy* para esse estado é desconhecida pelo atacante, inviabilizando assim o ataque. Este sistema continua a necessitar de uma ativação para associar o dispositivo do Consumidor à licença, por meio de técnicas tradicionais de DRM.

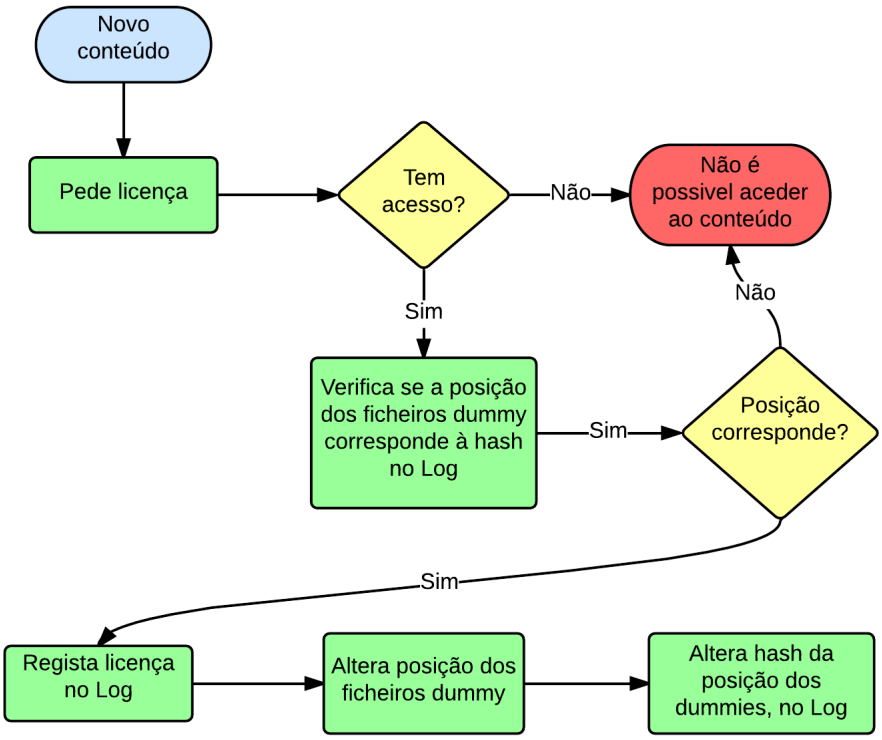

Figura 2.7: Sistema Detalhes Físicos da Licença

2.1.2.2 Sistemas baseados em hardware

O problema de o conteúdo ser acedido num ambiente possivelmente hostil afeta uma arquitetura tradicional de DRM, em que identificadores de dispositivos ou consumidores podem ser copiados para ambientes e assim possibilitarem o acesso a conteúdo por consumidores sem permissões para tal. Soluções já apresentadas resolvem esse problema por exemplo através de uma continua verificação da identidade do Consumidor, com recurso a certificados e a um servidor DRM, sempre que o conteúdo é acedido, contudo até essas informações podem ser alteradas ou copiadas caso o sistema assim o permita. As soluções com recurso a *hardware* para complementar a sua arquitetura dão acesso a um conjunto de novas oportunidades que um sistema só assente em criptografia por *software* não oferece.

- **Token**

  No contexto de DRM, um *token* é algo que é único e que possibilite a autenticação num sistema, seja virtual ou físico. Existem vários tipos de *tokens* físicos, como *Smart-cards* ou *Dongles*, que transportam informação do Consumidor para a autenticação no sistema.

  Em [ZYXZ09] é explorada uma solução, que com recurso a *Smart-Cards*, assegura a autenticação do dispositivo que quer aceder a conteúdo e o Consumidor. Na Figura 2.8 é apresentada a sequência de etapas que são necessárias para se aceder a um conteúdo com esta técnica. Sempre que um consumidor pretende aceder a um conteúdo tem que inserir um *Smart-Card*, que contenha informações associadas a si, no leitor de *Smart-Cards*, presente no dispositivo do Consumidor. O dispositivo comprova a sua identidade através de um servidor DRM, e é realizada uma troca de informações entre o *Smart-Card* e o dispositivo, incluindo operações criptográficas no *Smart-Card* para a autenticação, culminando na autenticação entre as três entidades, servidor DRM, Consumidor pelo *Smart-Card* e dispositivo. Após a autenticação o conteúdo pode ser acedido no dispositivo, descarregando-o do servidor DRM. Os sistemas [MnMnPP01, SHC07] disponibilizam o acesso *offline* ao conteúdo, mas não garantem a autenticação de todas as entidades.

  Outro *token* físico com relevância é uma USB *key*, visto que em quase todos os computadores existem conectores USB. [LLJ10] expõe uma arquitetura que depende de uma USB *key* para a autenticação, podendo o processo ser considerado como análogo à primeira solução de *Smart-Cards*, acima apresentada. O sistema permite ainda que após a autenticação do Consumidor com o servidor seja possível aceder ao conteúdo descarregando-o ou acedendo-o *online*, permitindo que se faça uso deste estando *offline*, mas com sempre com a condição da USB *key* estar presente.

- **Trusted Computing**

  O *Trusted Computing* disponibiliza ambientes seguros para um conteúdo ser acedido, recorrendo a proteção *hardware* para assegurar a segurança de informação importante como chaves de encriptação, ou validar a identidade do dispositivo através de identificadores únicos [RC05]. Essa proteção *hardware* encontra-se na forma do TPM, que inclui elementos

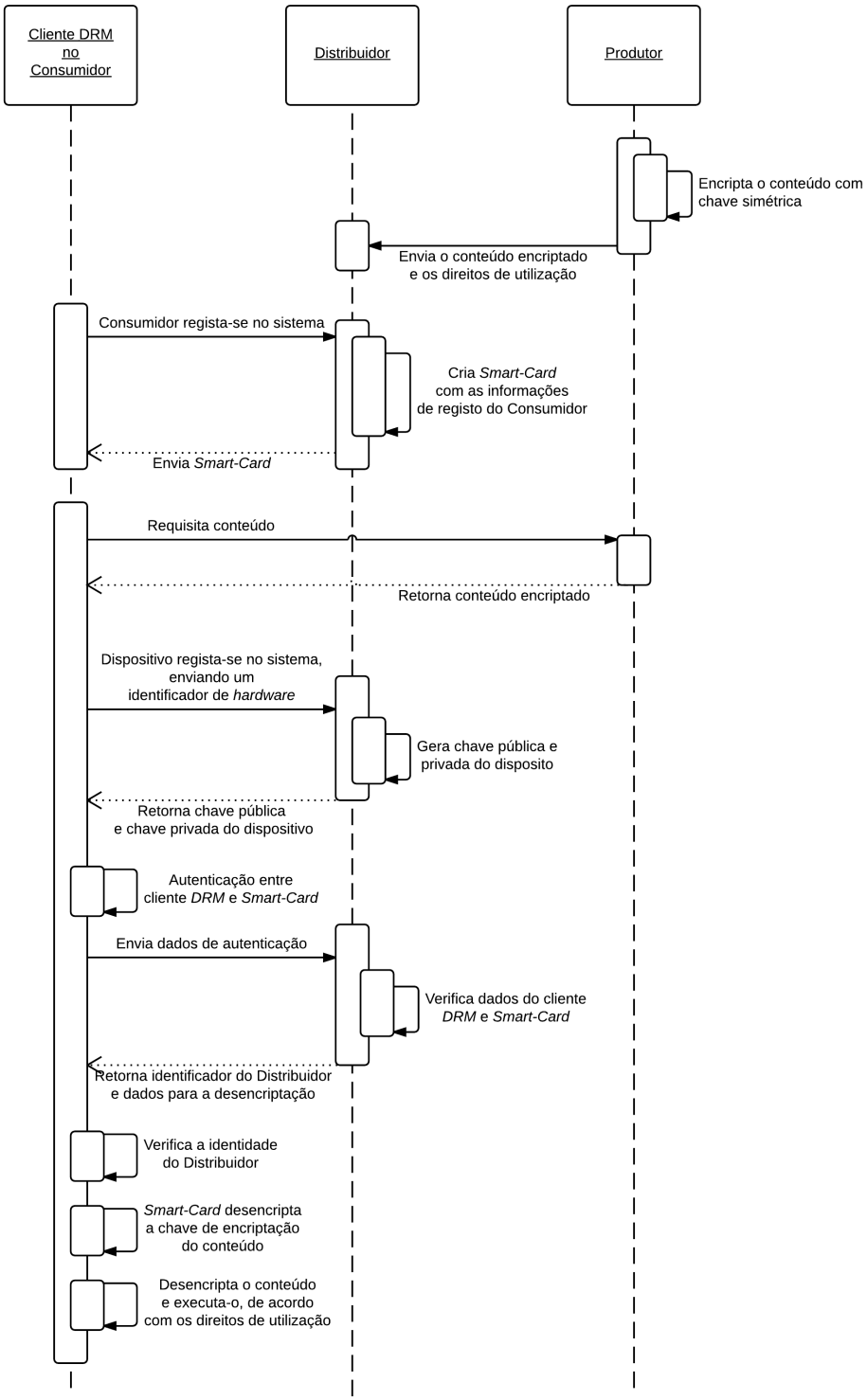

Figura 2.8: Sistema baseado em *Smart-Card*

como um processador criptográfico e um gerador de números aleatórios, e que é capaz de executar operações como encriptação ou desencriptação de conteúdo.

Um sistema DRM baseado em *Trusted Computing* é apresentado [YFL09], com características semelhantes a um sistema tradicional, mas com a particularidade de usar o TPM para confirmar a identidade dos outros componentes no sistema, como do servidor DRM para o cliente DRM ou nos próprios sub-elementos do cliente DRM. Nesta solução o uso de chaves públicas e certificados é descartado pois a autenticação com uso do TPM é considerada suficientemente segura para fazer a transferência da licença, não encriptada, para o cliente DRM.

- **Localização**

  A informação da localização do dispositivo que se quer aceder a conteúdo DRM é útil para casos onde se quer restringir o uso a determinada pessoa e dispositivos, com precisão. A solução de [Mun05] baseia-se na localização por GPS, numa plataforma de *Trusted Computing*, e que requer um recetor GPS. Embora a informação GPS seja uma fonte para assegurar a localização, é uma solução demasiado complicada para se aceder a conteúdos comerciais de relevância normal, surge então o sistema de localização por *WiFi* [SSSN07]. Neste sistema a localização é adquirida com base na informação do sinal a um ponto de acesso *WiFi*, não sendo a localização absoluta, como no GPS, mas sim relativa. O identificador do ponto de acesso, a ser verificado o sinal, é colocado na licença, sabendo assim o dispositivo o que deve analisar. Este sistema é adequado para situações onde o conteúdo é acedido sempre no mesmo lugar, como bibliotecas, universidades ou empresas. A Figura 2.9 resume o acesso a conteúdo com a técnica de Localização com *WiFi*.

2.1.3 Análise

Para estabelecer uma comparação entre algumas das soluções DRM referidas é feita uma análise de cada solução em relação a um conjunto de atributos. Foram definidos 6 atributos para diferenciar as técnicas:

- **Ativação Online:** Necessidade de o cliente conectar-se, pelo menos uma vez, ao servidor DRM
- **Utilização Offline:** Acesso ao conteúdo de forma offline, sem requerer uma ligação ao servidor DRM a cada acesso (podendo no entanto haver uma ligação, inicial ou esporádica, ao servidor)
- **Múltiplos Dispositivos:** Acesso ao conteúdo em diversos dispositivos com a mesma licença
- **Hardware Adicional Necessário:** Necessidade de dispositivos adicionais aos básicos, de distribuição e consumo, para por em prática a técnica

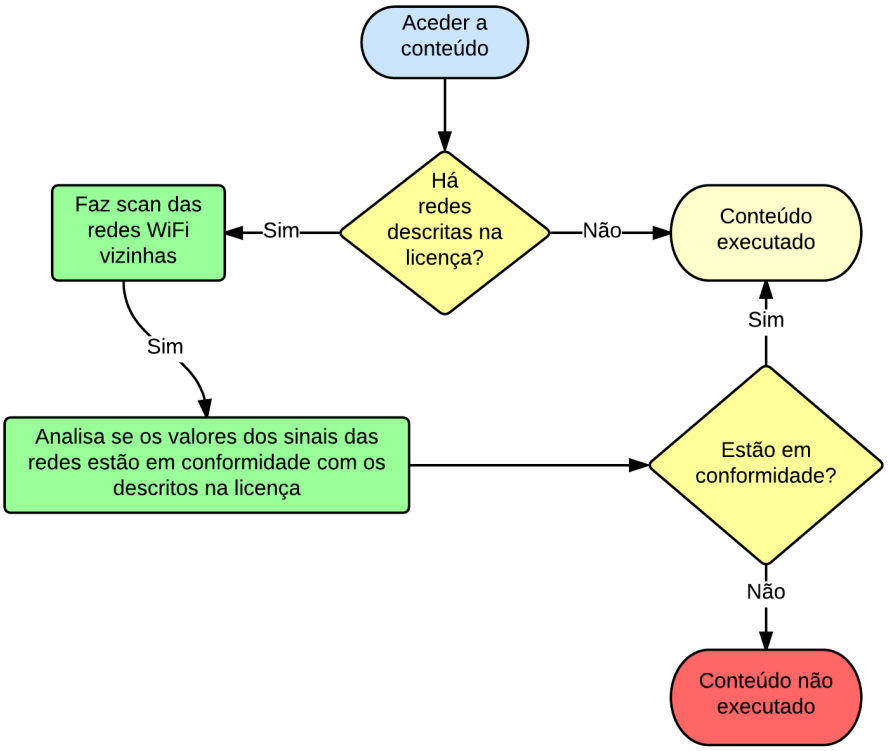

Figura 2.9: Sistema baseado em Localização por *WiFi*

- **Revogação de Direitos:** Possibilidade do proprietário do conteúdo revogar direitos de acesso ao consumidor, para um conteúdo
- **Transmissão de Direitos:** Possibilidade de um consumidor transferir ou vender a sua licença para outro consumidor

Foram escolhidas 9 técnicas que representam com mais relevância e interesse o seu tipo de técnica, a Tradicional, PKE [VS11], ABE [GVS13], Chave de Produto [HLJ+14], *Mobile Code White-Box* [GPB12], Detalhes Físicos da Licença [QYTC13], *Token* com *Smart-Card* [ZYXZ09], *Trusted Computing* [YFL09] e Localização com *WiFi* [SSSN07].

	Ativação Online	**Utilização Offline**	**Múltiplos Dispositivos**	**Hardware Adicional Necessário**	**Revogação de Direitos**	**Transmissão de Direitos**
Tradicional	Sim	Sim	Não	Não	Não	Não
PKE	Sim	Sim	Não	Não	Sim	Sim
ABE	Sim	Sim	Sim	Não	Sim	Sim
Chave Produto	Sim	Sim	Não	Não	Não	Não
Mobile Code White-Box	Sim	Não	Não	Não	Sim	Sim
Detalhes Físicos Licença	N/E	Sim	Não	Não	Não	Sim
Token com Smart-Card	Sim	Sim	Sim	Sim	Não	Não
Trusted Computing	Sim	Sim	Sim	Sim	Não	Não
Localização WiFi	N/E	Sim	Sim	Sim	Não	Não

Tabela 2.1: Comparação das técnicas DRM

Na Tabela 2.1 pode-se observar a análise realizada para técnica em relação aos atributos definidos. Quase todas técnicas precisam de uma ativação *online*, não sendo este fator especificado na Localização *WiFi* e Detalhes Físicos, mas todas possibilitam o acesso *offline* ao conteúdo. O acesso por múltiplos dispositivos é permitido com o ABE, *Smart-Card*, *Trusted Computing* e Localização *WiFi*, em que os três últimos são as únicos que requerem *hardware* adicional para serem implementados. O PKE, ABE e *Mobile Code White-Box* especificam a revogação de direitos na sua arquitetura, não acontecendo o mesmo para o resto das técnicas. Verifica-se uma situação semelhante com a transmissão de direitos, mas com o método de Detalhes Físicos da Licença a pertencer ao conjunto dos que possibilitam a transmissão.

2.1.4 Trabalho Relacionado

No contexto das soluções DRM encontram-se múltiplas ferramentas populares, que apesar da variedade do tipo de conteúdo que estas ferramentas protegem é possível dividi-las em ferramentas *Open-Source* e Comerciais. As ferramentas *Open-Source* caraterizam-se por permitirem a qualquer proprietário modificar a ferramenta e serem livres de custo, podendo ajustar a sua solução consoante o tipo de conteúdo, no entanto como o código ou estrutura são conhecidos a probabilidade de haver ataques com sucesso é bastante maior do que se não fossem *Open-source*, mesmo utilizando técnicas adicionais de segurança como obfuscação. Uma ferramenta Comercial é dotada, em principio, de uma proteção adicional por o seu código não ser conhecido, dificultando

assim possíveis ataques, havendo contundo custos monetários para quem quiser utilizar a ferramenta, mas havendo sendo normalmente especificada para um único tipo de conteúdo. Neste secção são apresentadas algumas das ferramentas consideradas populares para aplicar licenciamento, divididas em *Open-Source* e Comerciais como acima foi referido.

2.1.4.1 Open-Source

- **OpenSDRM**

  O *OpenSDRM* [Ope15b] teve a sua origem num projecto chamado *MOSES*, que após ser discontinuado partilhou a especificação do seu sistema à comunidade criando o *OpenSDRM*. Esta solução especifica um sistema DRM na sua totalidade, desde os componentes do Distribuidor, onde o conteúdo é disponibilizado, a bibliotecas para o cliente presente nos dispositivos do Consumidor, incluindo serviços de pagamento e gestão de licenças. Os diversos componentes são bem-conhecidos, com uma estrutura e interfaces definidas, podendo ser adaptados para diferentes modelos de negócio, como o caso onde o conteúdo é distribuído por *download*, *streamming* ou *broadcasting*. O conteúdo pode ser encriptado ou não, ficando encarregue o produtor de ditar as condições da encriptação, o mesmo se aplicando para outros casos como existência de licenças ou necessidade pagamento.

- **OpenIPMP**

  Tal como o *OpenSDRM*, o *OpenIPMP* [Mut15] disponibiliza uma sistema com todos os componentes básicos necessários para o processo de consumir um conteúdo, que nativamente é restringido apenas a *MPEG-2* e *MPEG-4*, mas podendo ser expandido para outros formatos. No entanto, o *OpenIPMP* assenta-se numa arquitetura mais centralizada, sem tantos componentes como o *OpenSDRM*, onde um servidor é responsável por toda a gestão do ambiente, havendo uma biblioteca que disponibiliza um conjunto de operações para que aplicações *3rd-party* possam usufruir de capacidades de encriptação e desencriptação de conteúdo, como no caso do Produtor para disponibilizá-lo e do Consumidor para o reproduzir. Esta solução permite ainda revogação de direitos de utilização, funcionalidade não presente no *OpenSDRM*.

- **OMA DRM**

  Criado pela *Open Mobile Alliance*, uma organização que define especificações de serviços para serem usados em dispositivos móveis e que tem como membros muitos dos seus principais fabricantes [Ope15a]. O *OMA DRM* pode ser referido como uma solução DRM mas não nas condições das soluções já apresentadas, pois esta não é mais que uma detalhada descrição de arquitetura de um sistema DRM, não disponibilizando bibliotecas ou ferramentas como o *OpenSDRM* e o *OpenIPMP* fazem. Não obstante, a primeira versão da solução foi implementada em cerca de 97% dos dispositivos presentes no mercado quando era a versão mais atualizada [BM07], afirmando a importância da sua criação.

O *OMA DRM* funciona base da identificação por dispositivo, isto é, uma ou mais caraterísticas do dispositivo onde o conteúdo é acedido são utilizadas para aferir se é permitido ou não o acesso, sendo estas características utilizadas no processos de criptografia existentes no sistema. Esta solução especifica o processo de distribuição e utilização de conteúdo, onde a partilha entre dispositivos ou num domínio está presente.

– **Versão 1.0**

A primeira versão do *OMA DRM* visa proteger conteúdos multimédia como imagens, vídeos ou ficheiros áudio, em dispositivos móveis. Nela são especificados 3 processos de distribuição, o *Forward-Locking*, o *Combined Delivery* e o *Separate Delivery*, que permitem ao Produtor escolher qual o nível de proteção que quer para o seu conteúdo. A Figura 2.10 mostra o que cada processo oferece.

No *Forward-Locking* o conteúdo é transmitido ao dispositivo do Consumidor sem qualquer informação adicional, sendo o dispositivo responsável por impedir que este seja transferido e que a sua reprodução seja unicamente local.

No caso do *Combined Delivery*, os ficheiros multimédia são associados a direitos de utilização, definidos pelo Produtor, na mensagem DRM, que é criada para a transferência entre o Distribuidor e o dispositivo do Consumidor. O dispositivo força então que o conteúdo seja reproduzido de acordo com os direitos de utilização, não permitindo que tanto o conteúdo e os direitos sejam transferidos.

O *Separate Delivery* possibilita a transferência de conteúdos entre dispositivos e utilização de canais inseguros para a mesma. Neste processo o conteúdo é encriptado com uma chave simétrica e pode ser transmitido ao dispositivo, que só consegue reproduzi-lo contactando o Distribuidor para ter acesso à chave simétrica para desencriptá-lo e direitos de utilização, ou seja a licença, que é entregue por um canal seguro. É possível a transferência de conteúdos pois o ficheiro encriptado pode ser transferido sem nunca poder ser reproduzido, até que um Consumidor faça o pedido de acesso ao mesmo e receba então as informações adicionais, no entanto a um Consumidor só é permitido partilhar o conteúdo encriptado e não a licença.

– **Versão 2.1.2**

A versão mais recente da especificação do *OMA DRM* é a 2.1.2, que parte dos princípios criados na primeira versão, as noções de direitos de utilização e encriptação de conteúdo, melhora e expande o sistema para abranger novos tipos de conteúdo, tais como jogos e aplicações. Esta especificação detalha os intervenientes do processo de distribuição e as suas interações, por exemplo, a interação entre o Distribuidor e o Consumidor. Introduz também a partilha de conteúdo num domínio, acesso a dados protegidos em *streamming*, distribuição para dispositivos *offline*, distribuição de aplicações com publicidade e retira a partição de processos de distribuição existente anteriormente.

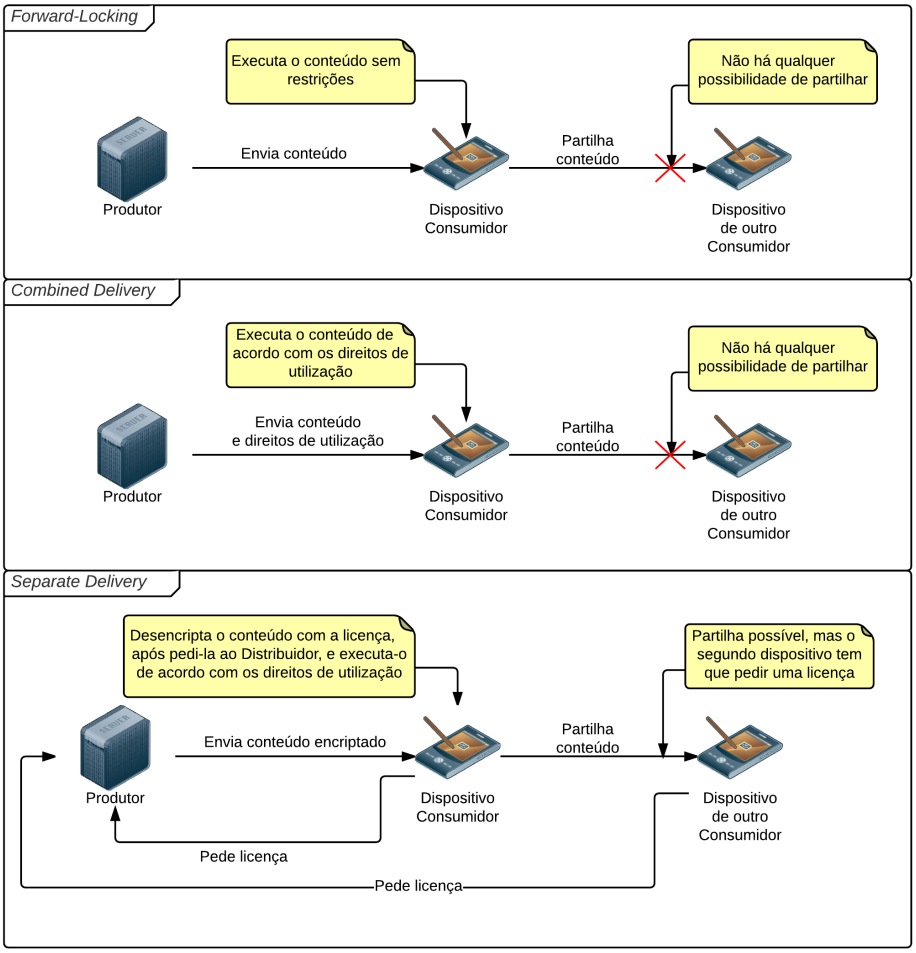

Figura 2.10: *OMA DRM* Versão 1.0

Uma dos aspetos com mais relevância em relação à primeira versão é a autenticação de dispositivos e o Distribuidor (ou a entidade que entrega as licenças). Na 1.0 não existe uma autenticação dos dispositivos móveis robusta, fazendo com que o conteúdo pudesse facilmente ser reproduzido e transferido sem restrições por dispositivos que se fizessem passar por seguros. Nesta versão recorre-se a técnicas PKI para atribuir chaves públicas, privadas e certificados às entidades que atuam no processo de distribuição, incluindo os dispositivos móveis. Com a utilização destas técnicas a operação para desencriptação do conteúdo só é realizada com a chave pública do dispositivo que desencriptar a licença, que por sua vez contém a chave de desencriptação pretendida, havendo mecanismos de comunicação bem definidos pelo *Rights Object Acquisition Protocol* para se obter as licenças com autenticidade. Na Figura 2.11 são expostos alguns processos de comunicação nesta versão do *OMA DRM*, com destaque para a partilha e transferência de conteúdo.

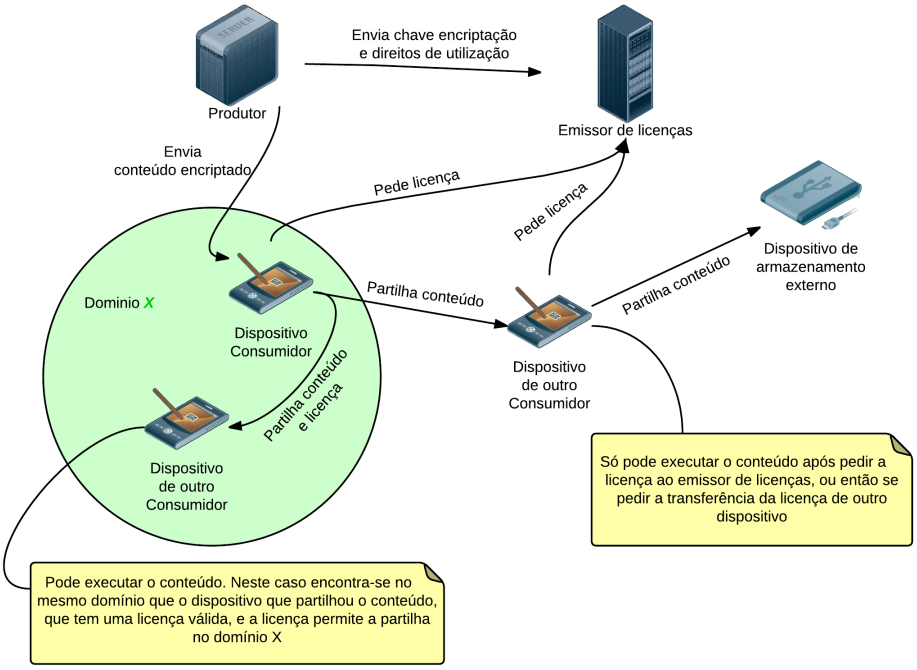

Figura 2.11: *OMA DRM* Versão 2.0

2.1.4.2 Comerciais

- **Adobe Adept**

  Considerada a maior solução DRM no campo dos livros eletrónicos, controla o mercado dos leitores eletrónicos que recorrem a esta ferramenta para restringir o uso dos seus conteúdos [Zha11]. Cada documento é encriptado e uma chave é associada ao Consumidor para poder aceder à chave de desencriptação. A generalidade dos leitores eletrónicos faz a desencriptação dos conteúdos sempre que o Consumidor acede a estes, em vez de ser só no ato de compra / descarregamento.

- **Apple FairPlay**

  O *FairPlay* da *Apple* protege conteúdo multimédia que se encontra em dispositivos e aplicações pertencentes à mesma empresa, como o *iTunes*, *iPhone* ou *iPod* [Zha11]. O sistema segue uma arquitetura tradicional de DRM, onde o conteúdo é encriptado previamente com uma chave privada e depois a chave de desencriptação é encriptada com um indentificador único associado ao sistema do Consumidor. Nesta solução os sistemas onde o conteúdo é reproduzido são registados, podendo assim haver uma monitorização deste e também controlo de acesso.

- **SecuROM**

As primeiras versões do *SecuROM* [Son15] foram destinadas à proteção de *software* armazenado em discos óticos, *CD-ROMs* ou *DVD-ROMs*. Através de uma chave de desencriptação contida no disco, o *software* é desencriptado se for introduzida uma chave de produto válida durante a instalação. A proteção da chave de desencriptação é garantida impossibilitando a cópia do conteúdo do disco, ou por verificações se o disco é o original.

Mais tarde foi criada uma nova solução pela *SecuROM*, que aplica o licenciamento com ativação *online*, mesmo que o *software* esteja num disco ótico. Na ativação, o Consumidor introduz uma chave de produto e uma identificação do dispositivo é enviada juntamente com a chave para um servidor DRM, ficando o *software* associado a esse dispositivo.

- **StarForce**

  Com um propósito semelhante ao do *SecuROM*, de proteção de discos óticos, o *StarForce* [Sta15] instalava no dispositivo do Consumidor uma *driver* que dava acesso à area de execução no nível de proteção mais alto do sistema operativo, o *ring0*. Com este sistema o *StarForce* conseguia limitar a cópia de discos óticos e outros processos para aceder a *software* indevidamente, como emulação de discos óticos.

  Atualmente, o *StarForce* comercializa várias soluções de segurança de *software*, em discos óticos, mas sem ser necessário recorrer à instalação de *drivers*, ou com a distribuição de conteúdo com ativação *online*, ficando o *software* associado a um dispositivo específico.

- **Microsoft PlayReady**

  Desenvolvido pela Microsoft, o *PlayReady* [Mic15a] específica e implementa um sistema DRM na sua totalidade, desde a parte do servidor DRM, com os serviços de distribuição de conteúdo, à parte do cliente DRM no dispositivo de leitura, disponibilizando bibliotecas para a integração com a aplicação do cliente. Esta solução é maioritariamente dedicada para conteúdo multimédia, visto que nativamente suporta apenas formatos multimédia, mas que pode ser expandida para outros formatos, como *ebooks*. No *PlayReady* encontram-se vários tipos de distribuição, por *streamming*, *download* progressivo ou acesso *offline*.

- **JScrambler**

  Esta ferramenta [Aud15], produzida pela mesma empresa onde esta dissertação se enquadra, a *JScrambler S.A.*, possibilita a implementação de uma série de proteções e otimizações a conteúdo *JavaScript*. No contexto do DRM ela aplica licenciamento na variante do controlo do período de utilização do conteúdo e quais os sistemas ou domínios autorizados. Um Produtor pode, através de uma aplicação *Web*, enviar um projeto *JavaScript* e selecionar quais as opções de proteção que pretende aplicar, fazendo com que o *JScrambler* se adapte facilmente a uma grande variedade de projetos, inclusive para dispositivos móveis.

2.2 JavaScript

A linguagem *JavaScript* foi inicialmente criada para alterar o comportamento de páginas *Web* em tempo real, uma extensão às funcionalidades dadas pelo HTML e CSS. É largamente utilizada na implementação de soluções *Web* por permitir um estilo de programação dinâmico e orientado a objetos, mantendo a sua característica principal de ser uma linguagem de *scripting*, isto é, passível de ser interpretada e não compilada e sendo suportada pela maioria dos *browsers* existentes.

A utilização do *JavaScript* foi alargada para outras necessidades, porque além de ser uma linguagem versátil e de fácil aprendizagem, é atualmente um *standard* para páginas *Web*. É também lógica a programação numa só linguagem para as diferentes aplicações de um projeto, quer seja a implementação de um servidor, a criação de páginas Web ou o desenvolvimento de aplicações móveis para diferentes plataformas. O uso do *JavaScript*, ou ligeiras modificações do mesmo, encontra-se mesmo em outras áreas que não só a criação de aplicações, como a criação de extensões para *browsers* ou programação de microcontroladores, comprovando uma vez mais a sua natureza adaptativa.

Nesta secção são enumeradas algumas das principais plataformas e *frameworks* onde o *JavaScript* é utilizado. Esta está dividida em tecnologias *Client-Side*, *Server-Side*, *Mobile*, *Scripting* e Microcontroladores. É feita, no final da secção, uma análise de algumas plataformas em relação às técnicas DRM estudadas.

2.2.1 Client-Side

As plataformas *client-side* caraterizam-se por serem maioritariamente distribuídas a um consumidor, executando código em qualquer dispositivo que esta afeto à plataforma. Nestas plataformas o código é executado localmente, mas numa tecnologia ligada à *Web* como o *JavaScript*, muitas da vezes existe comunicação remota. Esta secção visa explorar as plataformas *client-side* mais proeminentes em *JavaScript*.

2.2.1.1 Páginas Web

O primeiro uso dado à linguagem *JavaScript* foi alterar e dinamizar o conteúdo apresentado numa página *Web* no lado do utilizador, em tempo real. Atualmente, o seu propósito é basicamente o mesmo, mas com introdução de protocolos de comunicação como AJAX ou de *frameworks* que potencializam funcionalidades como *jQuery* ou *AngularJS*.

Segundo dados do *W3Schools*, o *Chrome*, *Internet Explorer*, *Firefox*, *Safari* e *Opera* são os *browsers* com mais uso. É de notar que existem vários motores de interpretação de *JavaScript*, cada um com variâncias de desempenho e que podem suportar ou não versões em particular da linguagem *JavaScript*. Cada *browser* implementa o motor que escolher, podendo haver disparidades entre execuções de um código *JavaScript* entre *browsers* distintos.

2.2.1.2 Extensões

Extensões, também chamadas de *plugins*, são ferramentas criadas para uma aplicação que adicionem funcionalidades não implementadas por defeito. A linguagem *JavaScript* é utilizada em algumas aplicações para o desenvolvimento de extensões para estas, entre elas encontram-se o *Google Chrome*, *Mozilla Firefox* e *Opera*. As extensões facilitam a tarefa de integração de funcionalidades com as aplicações, em parte pois, quando a opção de criação de extensões está disponível, existe uma biblioteca com a acesso às funcionalidades da aplicação.

No caso dos *browsers* a implementação de extensões, que são muito populares, pode não ser sempre em *JavaScript*. Para combater este problema existem soluções que visam a utilização de *JavaScript* para criar código que gere extensões para vários *browsers*, mesmo que não tenham suporte a extensões escritas em *JavaScript*. De entre estas soluções podem-se referir várias como o *Crossrider* ou o *KangoExtensions*.

2.2.1.3 Adobe AIR

Criado pela *Adobe*, o AIR [Ado15a] é uma ferramenta de desenvolvimento de aplicações para várias plataformas, *desktop* e *mobile*, com recurso a uma única fonte de código. Nesta ferramenta pode-se implementar aplicações em *Flash* e *ActionScript*, uma versão modificada do *JavaScript*, e com grande suporte ao nível de plataformas, ou em HTML e *JavaScript*, suportado apenas para aplicações *desktop*.

2.2.2 Server-Side

Uma plataforma para aplicações *server-side* é basicamente o oposto de *client-side*, no sentido em que o código é executado no servidor, e que normalmente tem o propósito de fornecer serviços, como por exemplo para aplicações *client-side*. Em *JavaScript* tem-se observado uma crescente utilização de plataformas *server-side* para fornecer serviços baseados nesta linguagem, apontando esta secção as duas plataformas mais utilizadas.

2.2.2.1 Node.js

O *Node.js* [Joy15, TV10] é um ambiente de execução de aplicações *JavaScript* idealizado para servidores. Com um funcionamento assíncrono por eventos e não bloqueador, não recorrendo a paradigmas de *multi-threading*, serve-se do *JavaScript* pela suas características também assíncronas, existência de *callbacks* e por ser o standard da criação de conteúdo *Web*.

Esta ferramenta está disponível para sistemas operativos como *Windows*, *Mac OS X* ou *Linux*, contendo nas suas bibliotecas chamadas ao sistema de ficheiros, operações de comunicação ou de criptografia. É possivel também aumentar as funcionalidades de uma aplicação para *Node.js* através de uma extensa biblioteca de *packages*, mantida pela comunidade, contado com mais de 100 mil destes [npm15].

2.2.2.2 Wakanda

Um motor de base de dados *NoSQL*, o *WakandaDB*, está nativamente incluido no *Wakanda* [Wak15], um ambiente de execução *JavaScript*, em semelhança com o *Node.js*, mas que usa um interpretador diferente, juntamente com bibliotecas de comunicação REST com a base de dados. Uma das diferenças em relação a outros servidores *JavaScript* é a capacidade de serem criadas várias *threads* para se realizarem operações, distinguindo-se da implementação do *Node.js* com uma única *thread* com eventos e *callbacks*. O *Wakanda* não tem contudo uma comunidade tão grande como o *Node.js*, onde o número de *packages* é muito superior.

2.2.3 Mobile

Esta secção tem o intuito de estudar as plataformas *JavaScript* mais importantes. Embora estas se incluam na vertente das aplicações *client-side*, estas apresentam restrições e particularidades inerentes aos dispositivos onde executam, sendo então lógico que se faça uma distinção destas em relação às outras.

2.2.3.1 Apache Cordova

Dedicado a facilitar a tarefa de implementar uma aplicação para diferentes plataformas, o *Apache Cordova* [Apa15] é uma ferramenta que permite a criação de aplicações em HTML, CSS e *JavaScript* com suporte para *iOS*, *Android*, *Windows Phone*, *BlackBerry*, entre outros.

O mecanismo que reutiliza o mesmo código para as várias plataformas assenta-se na encapsulação das páginas *Web* em *WebViews*. A aplicação age como se fosse nativa mas não sendo totalmente na realidade, onde os elementos visuais são os mesmos que uma página criada em HTML e mostrada num *browser*. No entanto, são disponibilizados *plugins* que fazem a ponte entre operações chamadas no *JavaScript* e os recursos do dispositivo móvel, característica que diferencia estas aplicações de páginas normais num *browser*, que não têm acesso a todos, ou pelo menos grande parte, dos recursos disponíveis no dispositivo. O uso de *WebViews* produz aplicações com um desempenho semelhante às nativas, contudo a visualização de elementos HTML e não nativos distancia a aparência das primeiras em relação às últimas [HHM13]. O *Apache Cordova* é também integrável com projetos que utilizem as bibliotecas de *JavaScript* de uma plataforma, como o *Tizen*, podendo estender as funcionalidades implementadas por essas bibliotecas nativas.

2.2.3.2 PhoneGap

O *PhoneGap* [Ado15d] é uma versão mais "trabalhada'' do *Apache Cordova*, tendo sido criado a partir do mesmo. No entanto, o *PhoneGap* contém funcionalidades que o *Cordova* não possui, como integração com os serviços da *Adobe*, empresa detentora do *PhoneGap*, que facilitam operações aos programadores como a compilação dos seus projetos nas diferentes plataformas, sem precisarem de ter no seu dispositivo de compilação as bibliotecas para cada plataforma. Mesmo com funcionalidades extras em relação ao *Apache Cordova* a arquitetura das aplicações é a mesma,

estando as versões mais recentes, menos estáveis, disponíveis no *Apache Cordova*, e versões do *Cordova*, mais testadas e estáveis, integradas no *PhoneGap*.

2.2.3.3 Appcelerator Titanium Mobile

Semelhante ao *PhoneGap*, o *Titanium Mobile* [App15] produz aplicações para várias plataformas, *Android*, *iOS* e *Windows Phone*, com um único código fonte. A diferença para o *PhoneGap* encontra-se no recurso a elementos nativos dos sistemas operativos e não *WebViews* para apresentar a aplicação.

Com o *Titanium Mobile* programa-se em *JavaScript*, onde o código é interpretado em tempo real para código nativo. As aplicações têm uma interface com os elementos disponibilizados pelo sistema operativo, criando uma experiência mais parecida com aplicações de código nativo do que com o uso de *WebViews*. São também dadas bibliotecas que fazem a relação com os recursos dos dispositivos, como sensores ou câmaras.

2.2.3.4 Windows Phone WinJS

A primeira versão do *Windows Phone* [Mic15b], da *Microsoft*, foi publicada em 2010, com suporte para o desenvolvimento de aplicações nativas em *C#* e XAML. Na versão atual, a 8.1, está integrada nas ferramentas de desenvolvimento de aplicações a biblioteca *WinJS*, que introduz a capacidade de o sistema operativo executar aplicações criadas em HTML e *JavaScript* de uma forma idêntica às nativas, havendo a hipótese de criar aplicações para *desktop* e dispositivos móveis com um único código fonte.

2.2.3.5 Tizen SDK

O *Tizen*, criado pela *Linux Foundation*, é um sistema operativo *open-source* para dispositivos móveis tais como *smartphones*, *smart tvs* ou *tablets* [Lin15]. Para a criação de aplicações para o sistema operativo estão disponíveis dois tipos, a criação de aplicações *Web* ou nativas.

Nas aplicações nativas um leque de funcionalidades do telemóvel é mais alargado do que nas aplicações *Web*, no entanto as últimas servem-se de linguagens HTML, CSS e *JavaScript* que permitem um desenvolvimento mais rápido e simples, ao invés da utilização da linguagem *C++*. As aplicações *Web* têm um desempenho comparado às nativas e um aspeto visual semelhante a estas por defeito, caso não haja modificação dos elementos por parte do programador.

O *Tizen* SDK é uma ferramenta que dá acesso às bibliotecas para a criação de aplicações para *Tizen*, quer sejam nativas ou *Web*, e que engloba um IDE e um emulador.

2.2.4 Scripting

Pela natureza do JavaScript, existe uma série de plataformas, que com recurso a esta linguagem, tiram partido de o código ser interpretado em *run-time*, possibilitando casos de uso para *scripting*. O *scripting* pode ter várias funções, como automatizar tarefas ou servir como ponte

entre aplicações e a execução de funções, expondo esta secção algumas plataformas onde o *JavaScript* é utilizado para efeitos de *scripting*.

2.2.4.1 Adobe Reader

A ferramenta *Adobe Reader*, na sua versão atual XI [Ado15b], inclui um interpretador de *JavaScript* que permite que documentos possam ter código *JavaScript* associado, e *plugins* sejam criados para a aplicação e que tarefas sejam realizadas, como combinação de documentos [Ado15c]. Nos documentos é possível obter e modificar informação, ou criar formulários que podem ser conectados a bases de dados ou sistemas de informação, dinamizando a interação entre o utilizador e o documento.

2.2.4.2 ExtendScript

O *ExtendScript* [Ado15e] é uma versão modificada do *JavaScript* criada pela *Adobe* para a sua *suite* de produtos, que engloba ferramentas como o *Photoshop* ou o *After Effects*. Contendo uma série de bibliotecas que permitem ao utilizador interagir com os ficheiros em produção em cada ferramenta através de *JavaScript*, pode-se criar soluções automatizadas para ficheiros, como colocar logótipos em muitos ficheiros de imagem ou video numa só operação.

2.2.4.3 MongoDB

O *MongoDB* [Mon15] é uma base de dados *open-source* e *NoSQL*. No *MongoDB* a linguagem *JavaScript* é usada para realizar *queries* à base de dados, através de uma *shell*, por defeito, com a introdução de comandos ou então pela leitura de ficheiros *Javascript*. Associadas ao *MongoDB* estão bibliotecas escritas em *JavaScript* que implementam as funcionalidades de acesso à base de dados e permitem também a fácil integração com outros sistemas.

2.2.4.4 Google Apps Script

Desenvolvido pela *Google*, o *Google Apps Script* [Goo15] é uma linguagem baseada em *JavaScript* que pode ser utilizada em vários serviços da *Google*, tais como o *Google Docs* ou *Google Apps*.

Com esta linguagem pode-se criar aplicações *Web* integradas nos serviços da *Google*, relacionar diferentes serviços, adicionar funcionalidades a documentos, conectar documentos a base de dados, entre outras possibilidades. O código é escrito e interpretado nos sistemas da *Google*, sendo o desenvolvimento de scripts inteiramente remoto.

2.2.5 Microcontroladores

A programação de microcontroladores é uma das mais recentes aplicações da linguagem *JavaScript*. Normalmente programados em linguagens como *C*, de mais baixo nível, os recursos da linguagem podem ser postos em ação em algumas plataformas, que são apontadas nesta secção.

2.2.5.1 Espruino

O *Espruino* [Pur15] é um microcontrolador que contém um interpretador de *JavaScript*, podendo ser instruído com código criado nessa linguagem e em tempo-real, isto é, sem haver a necessidade de compilar código para executar operações. Tem um maior poder de poupança de energia em relação a outros conhecidos microcontroladores, como o *Arduino*, que não é programado em *JavaScript*, em parte pelas características assíncronas e por eventos do *JavaScript*, que permitem estados de repouso sem estar à escuta de novas instruções.

Esta tecnologia permite aos utilizadores programarem diretamente num *browser* através da extensão *Espruino Web IDE* para o *Google Chrome*, oferecendo uma programação em código ou então por blocos, graficamente.

2.2.5.2 Tessel

Com especificações superiores ao *Espruino*, com mais memória e velocidade de processador, é permitido no *Tessel* [Tec15] a comunicação em rede nativamente através de um adaptador *WiFi* incluído na placa, e facilmente serem adicionados novos módulos de *hardware* em vez de serem soldados a conexões. O *Tessel* utiliza o *Node.js* como seu ambiente de execução de código, havendo a possibilidade de instalar *packages* para o *Node.js* e assim tirar partido do seu gestor de *packages* e bibliotecas. A linguagem *JavaScript* é utilizada para a programação do microcontrolador, estando o *Node.js* encarregue de interpretar o código fornecido.

2.2.5.3 NodeBots

Chamados de *NodeBots* [Nod15], microcontroladores *Arduino* são programados com recurso a *JavaScript* e a *Node.js*. Nos microcontroladores é necessário estar instalado uma biblioteca de comunicação com um computador chamada de *Firmata*, podendo ser criado código em *JavaScript* e através do pacote *johnny-five*, para *Node.js*, executar o código no microcontrolador, com este ligado ao computador.

Esta solução dá acesso a que microcontroladores com menos recursos do que o *Espruino* ou o *Tessel*, ou que não possuam interpretadores de *JavaScript*, possam ser programados em *JavaScript* com as bibliotecas definidas no *johnny-five*.

2.2.6 Análise

O conceito de DRM está intrinsecamente ligado com o mercado digital, visto que o principal interesse é proteger os direitos da propriedade intelectual, que tem um valor económico. É feita nesta secção uma seleção das plataformas e ferramentas *JavaScript* que apresentam um maior interesse para aplicações comerciais, por exemplo, no caso dos microcontroladores é pouco provável a existência de um grande número de aplicações comerciais.

Em cada plataforma é realizada uma análise das capacidades que esta tem para implementar algumas das técnicas estudadas na secção do DRM. Para as técnicas são estipulados atributos que

representam as características que uma plataforma deve possuir para conseguir implementar uma técnica, podendo no entanto existir o caso de uma técnica funcionar parcialmente se um ou mais atributos estiverem em falta.

2.2.6.1 Atributos

- Comunicação Online - **CO** : Possui algum meio de conexão online, para comunicar com um servidor DRM
- Bibliotecas PKE - **BPKE** : Existe alguma biblioteca, nativa ou externa, que disponibilize operações criptográficas PKE
- Bibliotecas ABE - **BABE** : Existe alguma biblioteca, nativa ou externa, que disponibilize operações criptográficas ABE
- Bibliotecas Encriptação Simétrica - **BES** : Existe alguma biblioteca, nativa ou externa, que disponibilize operações criptográficas de encriptação simétrica, por exemplo AES
- Bibliotecas de Mobile Code - **BMC** : Existe alguma biblioteca, nativa ou externa, que disponibilize um mecanismo de criação e transmissão de agentes, por exemplo JADE para *Java*
- Bibliotecas de Hashing - **BHASH** : Existe alguma biblioteca, nativa ou externa, que disponibilize operações de hashing, como *MD5* ou *SHA256*
- Leitura de certificados - **LCERT** : É possível fazer o parsing de certificados, como *X.509*
- Acesso ao sistema de ficheiros - **AFICH** : O acesso para leitura e escrita no sistema de ficheiros é disponibilizado
- Acesso a propriedades do dispositivo - **APROP** : Informações do dispositivo, como identificadores de componentes hardware, são disponibilizadas
- Acesso a componente WiFi - **AWIFI** : As informações sobre o estado do dispositivo *WiFi* e sobre as redes que se pode conetar são disponibilizadas
- Leitor de Smart-Cards - **LSMART** : Caso a plataforma possua algum tipo de leitor de *Smart-Cards*, são dadas bibliotecas para se aceder ao leitor
- Trusted Plataform Module - **TPM** : Caso a plataforma possua algum tipo de TPM, são dadas bibliotecas para se aceder ao módulo

2.2.6.2 Atributos necessários para implementar uma técnica

- PKE [VS11]: CO, BES, BPKE, LCERT, AFICH, APROP
- ABE [GVS13]: CO, BES, BABE, LCERT, AFICH, APROP

- Chave de Produto [HLJ+14]: CO, BHASH, AFICH, APROP
- Mobile Code White-Box [GPB12]: CO, BMC, BES
- Detalhes Físicos da Licença [QYTC13]: AFICH, BHASH
- Token com Smart-Card [ZYXZ09]: CO, LSMART, BHASH, BES, APROP
- Trusted Computing [YFL09]: CO, TPM, BES
- Localização com WiFi [SSSN07]: AWIFI, AFICH

2.2.6.3 Comparação

	Extensões	**AIR**	**Node.js**	**Cordova & PhoneGap**	**Titanium Mobile**	**WinJS**	**Tizen**
PKE	CO, BPKE, BES, LCERT, AFICH	CO, BPKE, BES, LCERT, AFICH, APROP	CO, BPKE, BES, LCERT, AFICH, APROP	CO, BPKE, BES, LCERT, AFICH, APROP	CO, BPKE, BES, LCERT, AFICH, APROP	CO, BPKE, BES, LCERT, AFICH, APROP	CO, BPKE, BES, LCERT, AFICH, APROP
ABE	CO, BES, AFICH	CO, BES, AFICH, APROP	CO, BES, AFICH, APROP	CO, BES, AFICH, APROP	CO, BES, AFICH, APROP	CO, BES, AFICH, APROP	CO, BES, AFICH, APROP
Chave Produto	CO, AFICH, BHASH	CO, AFICH, , BHASH	CO, AFICH, APROP, BHASH	CO, AFICH, APROP, BHASH	CO, AFICH, APROP, BHASH	CO, AFICH, APROP, BHASH	CO, AFICH, APROP, BHASH
Mobile Code White-Box	CO, AFICH, BES	CO, AFICH, BES	CO, AFICH, BES	CO, AFICH, BES	CO, AFICH, BES	CO, AFICH, BES	CO, AFICH, BES
Detalhes Físicos Licença	AFICH, BHASH	AFICH, BHASH	AFICH, BHASH	AFICH, BHASH	AFICH, BHASH	AFICH, BHASH	AFICH, BHASH
Token com Smart-Card	CO, BHASH, BES	CO, BHASH, BES	CO, BHASH, BES, APROP, LSMART	CO, BHASH, BES, APROP	CO, BHASH, BES, APROP	CO, BHASH, BES, APROP	CO, BHASH, BES, APROP
Trusted Computing	CO, BES	CO, BES	CO, BES	CO, BES	CO, BES	CO, BES	CO, BES
Localização WiFi	AFICH	AFICH	AWIFI, AFICH	AWIFI, AFICH	AWIFI, AFICH	AFICH	AFICH

Tabela 2.2: Capacidades das plataformas *JavaScript*

Na Tabela 2.2 está representada a análise entre as plataformas e as técnicas. As células marcadas a cor indicam que a plataforma e técnica indicada são compatíveis, e como tal é possível usar essa combinação para aplicar licenciamento. Todavia, existem casos onde mesmo a plataforma não tendo um certo atributo, não significa que não o poderá ter, por exemplo, no ABE não

existem bibliotecas de encriptação por atributos para *JavaScript*, mas podem ser criadas, com a implementação dos algoritmos criptográficos.

2.3 Resumo

Há uma urgência de existir aplicação de licenciamento de uma forma ativa, em dispositivos que consomem conteúdo. Surgem então mecanismos DRM que englobam esse licenciamento, com diferentes características e níveis de proteção. Deduz-se desde logo uma separação entre técnicas DRM que fazem uso de *hardware* adicional, como *Smart-Cards*, e técnicas com proteção puramente por *software*, com operações criptográficas. Sabe-se também que quase todas as técnicas necessitam de sistemas com mais do que uma entidade, recorrendo a servidores para comunicar com os clientes DRM, maximizando o número de verificações de acesso realizadas na aplicação de licenças. A maioria das soluções comerciais e *open-source*, que implementam técnicas DRM, não recorrem a *hardware* adicional, baseando-se em arquiteturas PKI com encriptação PKE para proporcionar a proteção.

A linguagem *JavaScript* está em expansão, abrangendo outras áreas que não só a criação de conteúdo *Web*. Nas plataformas onde existe conteúdo comercial a ser produzido, criado em *JavaScript*, como no *PhoneGap* ou *Windows Phone WinJS*, não há a possibilidade de implementar todas as técnicas DRM, por causa de limitações de cada plataforma. Estas limitações prendem-se com fatores como o tipo de acesso ao de ficheiros, que permissões o dispositivo tem para aceder a recursos do sistema operativo, ou se existem bibliotecas que implementem elementos necessárias para uma técnica. No entanto, a arquitetura PKI afirma-se, uma vez mais, como a mais versátil, sendo a sua implementação possível em quase todas as plataformas *JavaScript*.

Capítulo 3

Especificação e Implementação da Solução

3.1 Conceito

Numa época onde a linguagem *JavaScript* encontra-se a ser cada vez mais utilizada para os mais diversos tipos de aplicações, surge a necessidade de proteger conteúdo produzido com esta, associando mecanismos que protejam os interesses de todas as partes que intervenham na sua produção ou utilização. Pretende-se que para *software JavaScript*, seja possível integrar-se código de licenciamento, estando disponível uma monitorização e gestão dessa proteção. Assim sendo, é criada uma ferramenta que contém estas funcionalidades, com o nome de *license.js*.

3.2 Plataformas

O software *JavaScript* diverge no que diz respeito às funcionalidades que cada plataforma oferece. O *license.js* faz a proteção de aplicações em *Node.js* [Joy15], por ser uma plataforma versátil e em crescente utilização em desenvolvimento *Web*, permitindo a execução de uma aplicação em múltiplos sistemas operativos. É feita também a proteção em *Cordova* [Apa15], uma ferramenta dedicada à criação de aplicações multi plataforma, com foco nos sistemas operativos *iOS* e *Android*, por serem os dois sistemas operativos móveis mais utilizados atualmente, segundo dados de [IDC15]. Além disso, a escolha destas plataformas recai na liberdade que estas disponibilizam para a implementação de mecanismos de licenciamento.

3.3 Arquitetura

O *license.js* é composto por dois sistemas, o Servidor e o Módulo de licenciamento, que integra-se com o *software JavaScript* a proteger. A arquitetura é apresentada na Figura 3.1, e

descreve algumas das interações e componentes dos dois sistemas.

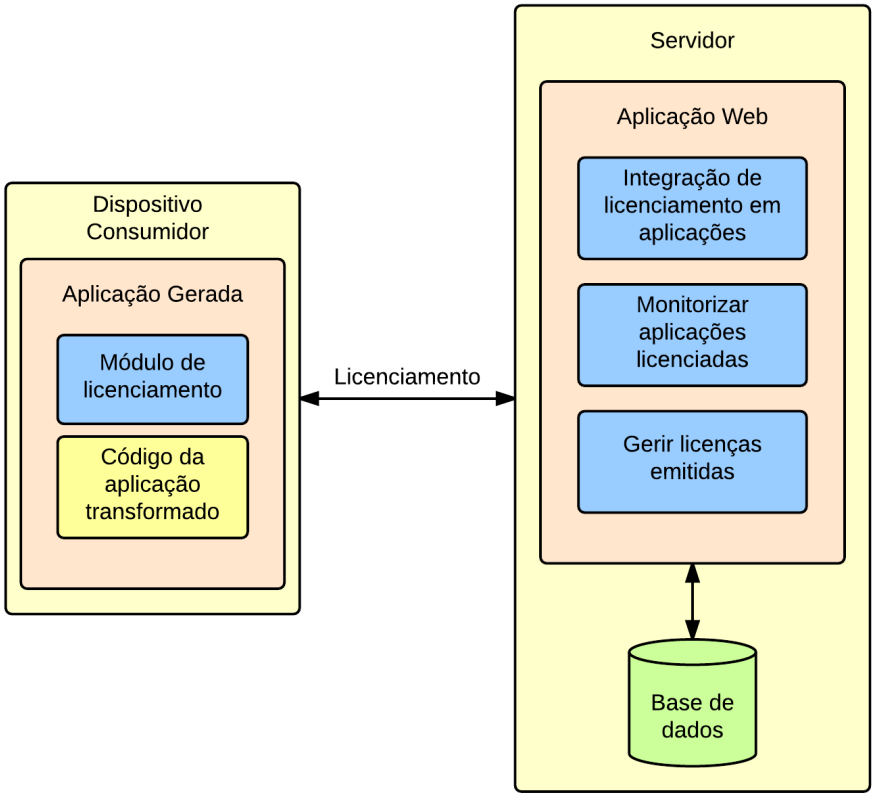

Figura 3.1: Arquitetura da ferramenta

O funcionamento geral do *license.js* é simples, uma aplicação a proteger é submetida numa interface *Web*, sofrendo transformações na sua estrutura que providenciam a prática de licenciamento.

A Figura 3.2 apresenta esse funcionamento num panorama mais geral do *license.js*, após a transformação da aplicação, a nova versão é distribuída aos consumidores, que ao ativarem as suas cópias desencadeiam um processo de verificações dos direitos de utilização para confirmar que a aplicação está a ser executada corretamente. O *license.js* permite ainda que um produtor possa gerir as suas aplicações, licenças e subscrições, como transferir licenças entre utilizadores ou modificar parâmetros de direitos de utilização, e caso a licença de um consumidor expirar, este pode pedir ao produtor para renovar a subscrição, utilizando o segundo os serviços do *license.js* para criar uma nova subscrição.

3.3.1 Servidor

O Servidor é o *back-end* do*license.js*, e é responsável por funcionar como Produtor e Distribuidor de um sistema DRM tradicional. É neste sistema que um autor de conteúdo *JavaScript* pode

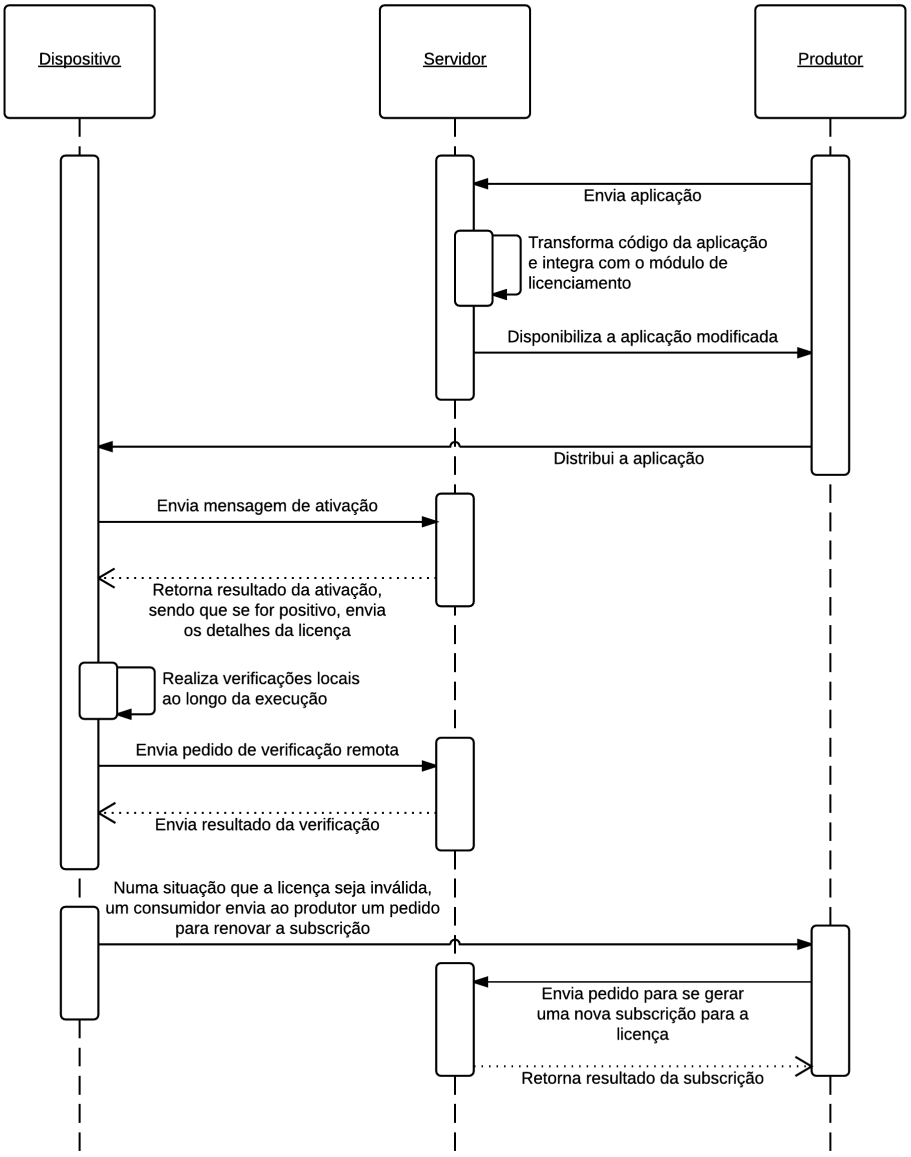

Figura 3.2: Funcionamento geral do license.js

submeter o seu *software* através de uma aplicação *Web*. Este é composto por 3 funcionalidades principais, a geração de aplicações com licenciamento, a monitorização e gestão de aplicações e o serviço de licenciamento.

- **Geração de aplicações com licenciamento:** O Servidor é capaz de através da submissão de uma aplicação *JavaScript*, para uma plataforma qualificável, adaptá-la e integrá-la com código de licenciamento para a plataforma em questão.
- **Monitorização e gestão de aplicações:** Quem submete aplicações tem, na sua conta, uma lista das aplicações submetidas e das licenças emitidas para cada aplicação. Pode ainda gerir as licenças, podendo executar operações como transferir e revogar licenças, ou alterar parâmetros de dados de utilização.
- **Serviço de licenciamento:** No servidor é implementado um serviço que trata dos processos de licenciamento entre a aplicação gerada, presente nos dispositivos dos consumidores, e o Servidor. É responsável ainda por dar acesso a quem submeteu uma aplicação à geração de licenças, com a especificação dos direitos de utilização, e de *tokens*.

O servidor integra ainda uma base de dados, que contém as informações das aplicações e licenças, assim como dados de *tokens* gerados.

3.3.2 Módulo de licenciamento

A aplicação gerada pelo Servidor é o produto resultante da injeção de código de verificação de licenciamento, no código original, e da integração com o Módulo de licenciamento. O Módulo de licenciamento engloba as técnicas de licenciamento a serem executadas no dispositivo do Consumidor, tratando da comunicação com o servidor e gerindo as verificações de direitos de utilização que são executadas. O Módulo tem diferentes versões, dependendo da plataforma a que se destina, mas que, no entanto, tem algumas características principais, inerentes a todas as versões criadas:

- Gere o acesso a uma aplicação *JavaScript*
- Aplica os direitos de utilização
- Implementa verificações de licenciamento que são chamadas ao longo do código da aplicação

3.4 Instrumentação do código

Quando um produtor submete uma aplicação *JavaScript* para o *license.js*, faz o carregamento do código fonte desta. Com este código presente nos serviços da ferramenta, são injetados no código um conjunto de transformações e adicionados novos elementos ao projeto para acrescentar uma camada de licenciamento.

A injeção no código é proporcionada pelo *parser Esprima* [Esp15], que faz o *parsing* de código *JavaScript* e produz uma árvore de sintaxe estruturada em JSON. Com estes dados em JSON é possível modificar os nós relevantes, como blocos de códigos, e injetar expressões de licenciamento. Além da injeção em *JavaScript*, o *license.js* também modifica ficheiros HTML e XML, funcionalidade importante na proteção em *Cordova*, através do parser *HTML Parser 2* [fee15], que tal como o *Esprima*, gera uma árvore de sintaxe em JSON. Para fazer o processo inverso, de gerar o código modificado a partir da árvore, é utilizado o *escodegen* [est15] para *JavaScript*, e um módulo do *JScrambler* para HTML e XML.

Com estas modificações realizadas, o código fica disponível para o produtor poder distribuir aos seus consumidores. As transformações do código criadas pelo *license.js* foram também integradas com os serviços do *JScrambler*, como é um dos objetivos deste projeto.

3.4.1 Node.js

Para projetos *Node.js*, o módulo de licenciamento é dividido em dois sub-módulos, o de ativação e outro responsável pelas verificações dos direitos. Esta separação existe pois uma aplicação é comprimida e encriptada com uma chave simétrica, gerada aleatoriamente, quando é submetida, visto que, tal como num sistema DRM que funcione com base em criptografia [LHJ+03, VS11, GVS13], o conteúdo fica pronto a ser distribuído com uma barreira adicional de proteção contra a leitura do código. A encriptação permite também que a distribuição da aplicação seja realizada apenas com um ficheiro, o do projeto encriptado. Esta tipo de distribuição também garante que a aplicação só é executada após o Consumidor executar o módulo de ativação, sendo intuitivo desde logo para este, a necessidade da ativação.

Numa aplicação é injetado um módulo de verificação de licenciamento, que contêm o código relativo à comunicação com o servidor e verificações, locais e remotas, controlando o acesso à aplicação depois de ser ativada. Neste módulo é injetado o identificador da aplicação na base de dados, para o servidor reconhecer qual a aplicação a comunicar, e no módulo de ativação é adicionado uma UUID, que age como ponte entre o dispositivo e a licença, durante a ativação.

3.4.2 Cordova

Ao contrário dos módulo para *Node.js*, não existe separação entre a ativação e verificações. Devido a restrições dos sistemas operativos a que as aplicações *Cordova* se destinam, o sistema de ficheiros não é completamente acessível e passível de ser modificado como em *Node.js*. A inabilidade de modificar os ficheiros após uma aplicação ser instalada, impede que o projeto ao ser submetido no *license.js*, seja encriptado e depois desencriptado após a ativação, pois os ficheiros desencriptados não podem substituir os existentes antes da ativação.

Não havendo esta camada de encriptação, é adicionado ao projeto o módulo de licenciamento, que contêm tanto os passos de ativação e verificação, sendo injetado o identificador da aplicação neste, tal como em *Node.js*.

3.4.3 Verificações

O módulo de licenciamento é semelhante nas duas plataformas, embora uma o tenha separado da ativação e a outra não. Neste, estão implementados métodos que tratam da comunicação com o servidor, para as verificações *online*, e das verificações locais, disponibilizando para o código de uma aplicação uma função de verificação.

No código original do projeto, tanto em *Node.js* como *Cordova*, são adicionadas chamadas à função de verificação do módulo de licenciamento. As chamadas são injetadas em blocos de código, como funções ou em condições *if*, no entanto, existe uma restrição sobre a injeção em ciclos, para evitar uma quebra de desempenho, pela execução de verificações continuadamente, sem tal ser necessário. As injeções podem ainda serem controladas, existindo um fator de probabilidade, que pode ser alterado, se uma chamada é injetada ou não num bloco, permitindo que a dispersão das chamadas pelo código seja alterada e como tal, ajudando no ajuste do desempenho do *license.js* numa aplicação. É ainda importante notar que as chamadas são assíncronas, não bloqueando a execução do código da aplicação, e portanto reduzindo o impacto no desempenho desta.

3.4.4 Anotações de código

Mesmo com a opção de ajustar a probabilidade de injeção de chamadas de verificação no código da aplicação, pelo produtor, a granularidade da personalização do licenciamento pelo código é diminuta. Visto que o desempenho é um dos fatores importantes a ter em conta na execução de uma aplicação, a verificação por parte do *license.js* se uma chamada é injetada em ciclos pode não ser suficiente, pois um chamada pode não ser incluída num ciclo, mas ser incluída numa função, que por sua vez é executada dentro de um ciclo.

Para dar ao produtor a oportunidade de ter um controlo mais rígido sobre as transformações aplicadas na aplicação, o *license.js* implementa anotações de código. As anotações de código permitem que um produtor possa adicionar uma série de comentários, com uma determinada estrutura, que forçam, desativam ou ajustam a probabilidade das injeções em blocos de código

```
1   /** @licensejs enable */
2   function a() { console.log('a'); }
3
4   /** @licensejs disable */
5   function b() { console.log('b'); }
6
7   /** @licensejs 50% */
8   function c() { console.log('c'); }
9
10  function d() { console.log('d'); }
```

Figura 3.3: Exemplo anotações de código antes de aplicar transformações

A Figura 3.3 retrata uma situação onde estão dispostas várias funções pelo código, em que todos os tipos de anotações compreendidos pelo *license.js* estão aplicados. A função *a* tem a anotação *@licensejs enable* a antecede-la, o que força a que uma chamada de verificação seja injetada no bloco de código desta. Na função *b* acontece o efeito contrário do que a função *a*, com a utilização da anotação *@licensejs disable* que desativa a injeção. Na última função que tem uma anotação a anteceder está comentada a expressão *@licensejs 50%* que, neste caso, modifica a probabilidade ser inserida no bloco uma chamada de verificação em 50%, podendo o produtor alterar o valor para uma probabilidade entre 0% e 100%. Na função *d* não está aplicada qualquer anotação, e como tal deve reger-se pelas opções por defeito das transformações.

Depois de serem aplicadas as transformações ao código da situação anterior, as funções são modificadas para a forma apresentada na Figura 3.4, com as injeções a terem um valor por defeito de 100%. Neste exemplo sabe-se que a função *d*, não tendo anotação, segue os 100% de probabilidade e como tal sofre injeção. A anotação a função *c* faz o *override* do valor por defeito de 100% para 50%, estando no código transformado com uma chamada, mas que poderia não ter em outras transformações.

```
1  function a() { licensejs.checkLicense(); console.log('a'); }
2
3  function b() { console.log('b'); }
4
5  function c() { licensejs.checkLicense(); console.log('c'); }
6
7  function d() { licensejs.checkLicense(); console.log('d'); }
```

Figura 3.4: Exemplo anotações de código depois de aplicar transformações

3.4.5 Duplicação de chamadas

O facto de as injeções de licenciamento no código da aplicação serem todas iguais, facilita a tarefa de um atacante para as remover, isto é, descobrindo este que uma chamada a uma função corresponde à verificação no módulo de licenciamento, basta remover todas as chamadas à função pelo código. Para tornar esta tarefa menos trivial, o *license.js* utiliza *alias* para o *namespace* do módulo de funcionamento e para a função de verificação, que está presente no módulo.

A Figura 3.3 representa uma situação em que o código é injetado com apenas um tipo de chamada de verificação. Na Figura 3.5 pode-se perceber como o *license.js* faz realmente a injeção no código da aplicação. São implementados vários módulos que funcionam apenas como *alias*, tendo o papel de apenas reencaminhar para o módulo principal. Nestes módulos são criadas ainda outras funções com o intuito de serem *alias* para a chamada de verificação do módulo principal, dificultando ainda mais a tarefa para um atacante, não bastando este remover a mesma chamada para os diferentes módulos de *alias*. No entanto, esta técnica não resolve o problema de o atacante

```
1    function a() { licensejsAlias3.checkLicenseAlias1();
2        console.log('a'); }
3
4    function b() { console.log('b'); }
5
6    function c() { licensejsAlias1.checkLicenseAlias10();
7        console.log('c'); }
8
9    function d() { licensejsAlias5.checkLicenseAlias7();
10       console.log('d'); }
```

Figura 3.5: Exemplo de código transformado com duplicação de chamadas

atacar o módulo de licenciamento diretamente e afetar o licenciamento, visto que caso este consiga fazer isto então os *alias* vão chamar o módulo acedido pelo atacante.

3.5 Licenciamento

O Licenciamento traduz-se na capacidade de uma aplicação garantir que os direitos de utilização estipulados pelo produtor são cumpridos. Nesta ferramenta, informação sobre o sistema operativo, identificadores únicos de dispositivos e tempo de expiração, são os parâmetros utilizados para formular os direitos de utilização, podendo serem expandidos caso assim seja pretendido.

O mecanismo de licenciamento divide-se em duas partes, primeiramente a ativação e, depois desta, a verificação continua de direitos de utilização. Os processos implementados para a segunda parte são semelhantes para todas as plataformas, no entanto, tendo em conta que cada plataforma tem diferentes restrições para aceder ou modificar informação do sistema, a ativação não é igual para todas.

3.5.1 Ativação

O primeiro passo para uma aplicação protegida executar pela primeira vez num dispositivo é sempre a ativação. Este passo consiste em informar ao servido que o dispositivo, que quer executar a aplicação, está associado à licença. Para identificar o dispositivo, é enviado um identificador do sistema, que analogamente a um *token* físico, funciona como uma referência para o processo de licenciamento, interligando o dispositivo à identidade do Consumidor e das suas subscrições. Depois de ativada a aplicação, é transferida para o dispositivo a informação relativa à licença e subscrição ativa, que está encriptada simetricamente com AES, sendo a chave de encriptação o identificador de dispositivo.

O mecanismo de ativação varia entre as duas plataformas, e até intra-plataforma, em diferentes sistemas operativos, como é o caso do *Cordova*. As restrições de acesso a funcionalidades nas plataformas e sistemas operativas, são os fatores que ditam as diferenças no mecanismo.

3.5.1.1 Node.js

Os sistemas operativos onde o *Node.js* opera dão acesso a uma certa gama de informações sobre si, como informação sobre os detalhes o processador ou qual o nome do sistema. Não obstante, a informação sobre a identidade de quem possui o dispositivo não existe, excetuando o *hostname*, que por si só não é suficiente para a ativação.

O *license.js* gera um identificador único - UUID*v4* [LMS05] para cada ativação, sempre que há uma nova licença a ser criada, e associa-o à licença. Este *token*, quando é gerado, é injetado no módulo de ativação, ficando o módulo pronto a ser distribuído para o Consumidor, e faz com que cada ativação esteja associada a um módulo único. A Figura 3.6 apresenta em pormenor os passos que a ativação requer para ser completada com sucesso.

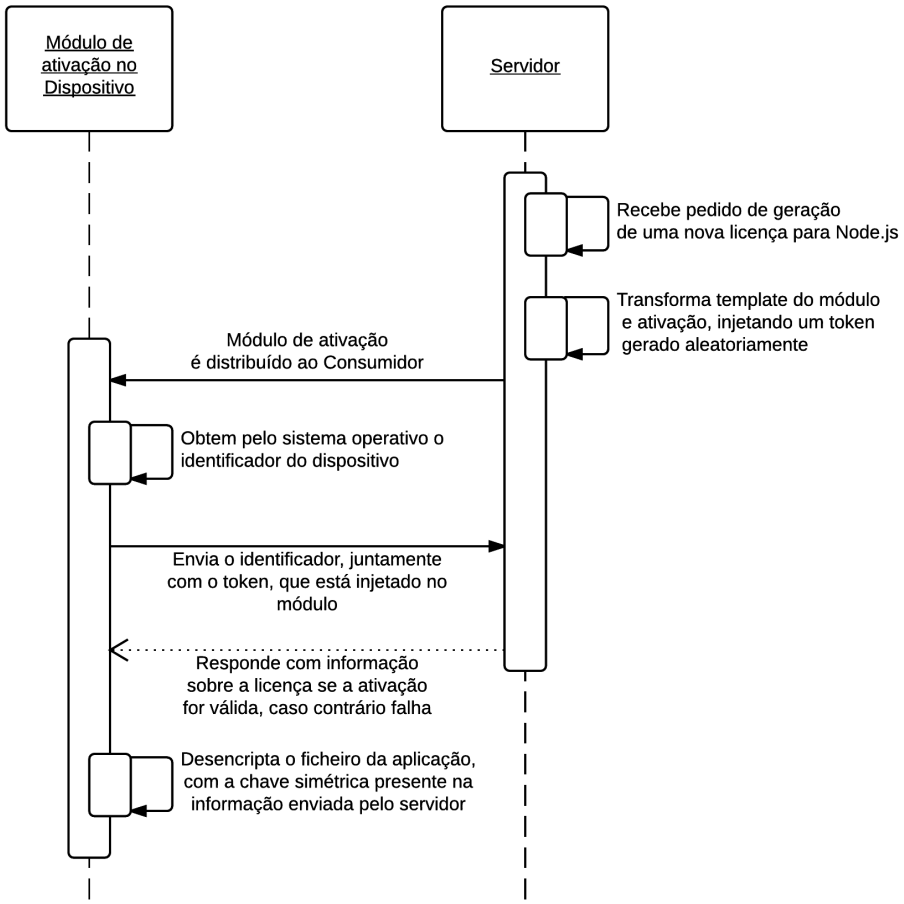

Figura 3.6: Processo de ativação em *Node.js*

Com o *token* presente no módulo, este só necessita de enviar ao servidor mais uma informação para iniciar o processo de ativação, um identificador único do sistema. Devido às características da plataforma e sistemas operativos, um identificador mais propício a ser utilizado é um que

esteja relacionado com componentes físicos, o *hardware*, visto que a informação destes componentes é menos facilmente modificada. Uma *hash* SHA-256, criada pela concatenação de todos os endereços MAC dos componentes de rede, é usada como identificador do dispositivo, permitindo que as comunicações e verificações associadas à aplicação, no dispositivo de ativação, sejam potencializadas com esta.

3.5.1.2 Cordova - Android

Para a ativação, o sistema operativo *Android* providencia uma forma de reconhecer a identidade de quem possui o dispositivo, o email. O *license.js* utiliza então o *email* associado ao dispositivo, ou dá a possibilidade do Consumidor escolher caso tenha mais que um, para agir como *token* de ativação, podendo o servidor reconhecer a que licença se destina, já que este contém informação sobre o *email* do Consumidor.

Como se vê na Figura 3.7, que representa o mecanismo de ativação para *Cordova*, o *email* é utilizado como *token* e um identificador do sistema é providenciado pelo sistema operativo, no caso do sistema operativo *Android*. Embora o *email* pudesse ser um identificador do sistema, este é facilmente extraviável e modificado, tendo um propósito muito mais viável numa ativação, do que no processo de verificações após a ativação.

O identificador dado pelo sistema é utilizado pois, além do *Android* atribuir um identificador único por cada dispositivo, este é permanente, não sendo eliminado ou mudado com atualizações ou restauros do sistema. A falta de informação providenciada pelo sistema operativo, sobre identificadores de componente físicos, faz com que a escolha do identificador recaia no gerado pelo *Android*.

3.5.1.3 Cordova - iOS

Semelhante ao *Node.js*, o sistema *iOS* não providencia nenhuma informação às aplicações sobre a identidade de quem possui um dispositivo. No entanto, visto que para instalar uma aplicação, neste sistema operativo, é necessário que seja via *AppStore*, uma aplicação incluída por defeito, implica que as aplicações tenham uma só versão para todos os consumidores, pois só é possível submeter-se uma versão da aplicação na *AppStore*. Com efeito, nem a estratégia de se utilizar um *email* ou injetar um *token* por aplicação, para a ativação, estão disponíveis.

O *license.js* gera um *token* aleatório de caracteres alfa-numéricos, com 6 dígitos, quando uma licença é gerada. Ao Consumidor é transmitido este *token* através do produtor, e é pedido que este introduza os caracteres numa *pop-up*, mostrada nas aplicações não ativadas em *iOS*.

Em *iOS* não há forma de obter dados que identifiquem um dispositivo, sejam físicos ou não, e que garantam a unicidade do identificador. Para colmatar esta insuficiência de informação, o *license.js* cria uma UUID numa ativação e guarda-o na *keychain*, uma área protegida do sistema operativo *iOS* com encriptação. O identificador fica guardado permanentemente, sobrevivendo a reinstalações de uma aplicação ativada, contudo caso haja um restauro do sistema o identificador é eliminado, tendo o Consumidor que fazer uma nova requisição de licença, ou informar que tem

um novo dispositivo. O diagrama da Figura 3.7 apresenta as interações efetuadas na ativação, para *iOS*.

3.5.2 Verificações

Após uma aplicação ser ativada num dispositivo entra em ação o mecanismo de verificação de licenciamento, do módulo de licenciamento. Estas verificações recaem sobre os direitos de utilização, especificados para cada licença sob o titulo de subscrições, podendo ser geradas sem qualquer restrição de quantidade, isto é, uma licença pode ter uma ou mais subscrições associadas, ativas ou não. Os direitos de utilização que são verificados são os seguintes:

- Tempo de expiração
- Nome de sistema operativo
- Bloqueio a um dispositivo, pelo identificador

O tempo de expiração dita qual o período de tempo que um dispositivo pode executar uma aplicação, o nome do sistema operativo é importante para a comparação com a versão do sistema operativo instalado no dispositivo, e o bloqueio a um dispositivo permite que uma licença só seja válida no dispositivo de ativação. A expansão dos direitos de utilização é possível de ser efetuada, como a existência de um contador de utilizações, útil numa versão de experimentação de uma aplicação, mas que implica no entanto algumas alterações às verificações e mensagens trocadas com o servidor.

Para cada verificação, caso o resultado seja positivo, a aplicação continua a ser executada na normalidade, sem qualquer *feedback* ou necessidade de intervenção do Consumidor, contudo, caso seja negativo, o acesso à aplicação será terminado até haver uma subscrição ativa e em concordância com os direitos que o dispositivo oferece. Nesta situação, em *Node.js* a aplicação é terminada, apresentando o *license.js* uma mensagem informativa na consola, e em *Cordova* é mostrada uma *pop-up*, tanto em *Android* como em *iOS*, com uma mensagem semelhante à de *Node.js*. Em *Cordova* a aplicação continua a executar, embora não funcionar por haver a *pop-up* a bloquear o acesso, para dar ao Consumidor uma hipótese de continuar a utilizar a aplicação caso a subscrição seja renovada (pelos serviços do Produtor), e também para este ficar ciente da causa que impede o acesso à aplicação. Uma verificação só é realizada caso não haja nenhuma verificação em ação no momento, para reduzir o overhead no caso de haverem muitas chamadas próximas no código, e também por ser redundante haver verificações a serem feitas quase ao mesmo tempo.

A Figura 3.8 expõe o fluxo que o módulo de licenciamento segue, na tomada de decisão de que tipo de verificação deve realizar, e de quais as reações às respostas das verificações.

3.5.2.1 Verificação local

O tipo mais frequente de verificação é o local, em que os dados do sistema são comparados aos direitos de utilização presentes na licença, guardada no dispositivo após a ativação. Como a

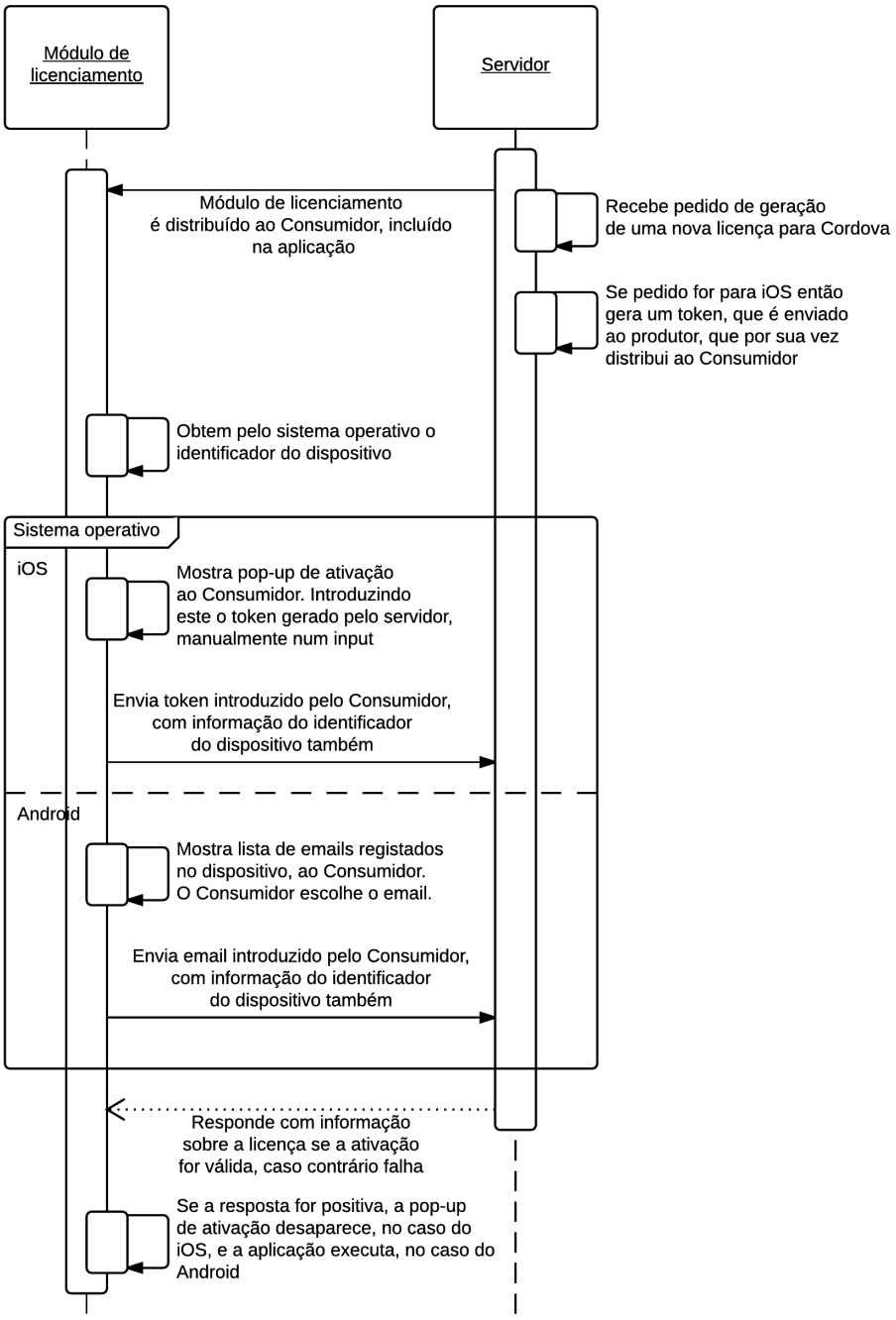

Figura 3.7: Processo de ativação em *Cordova*

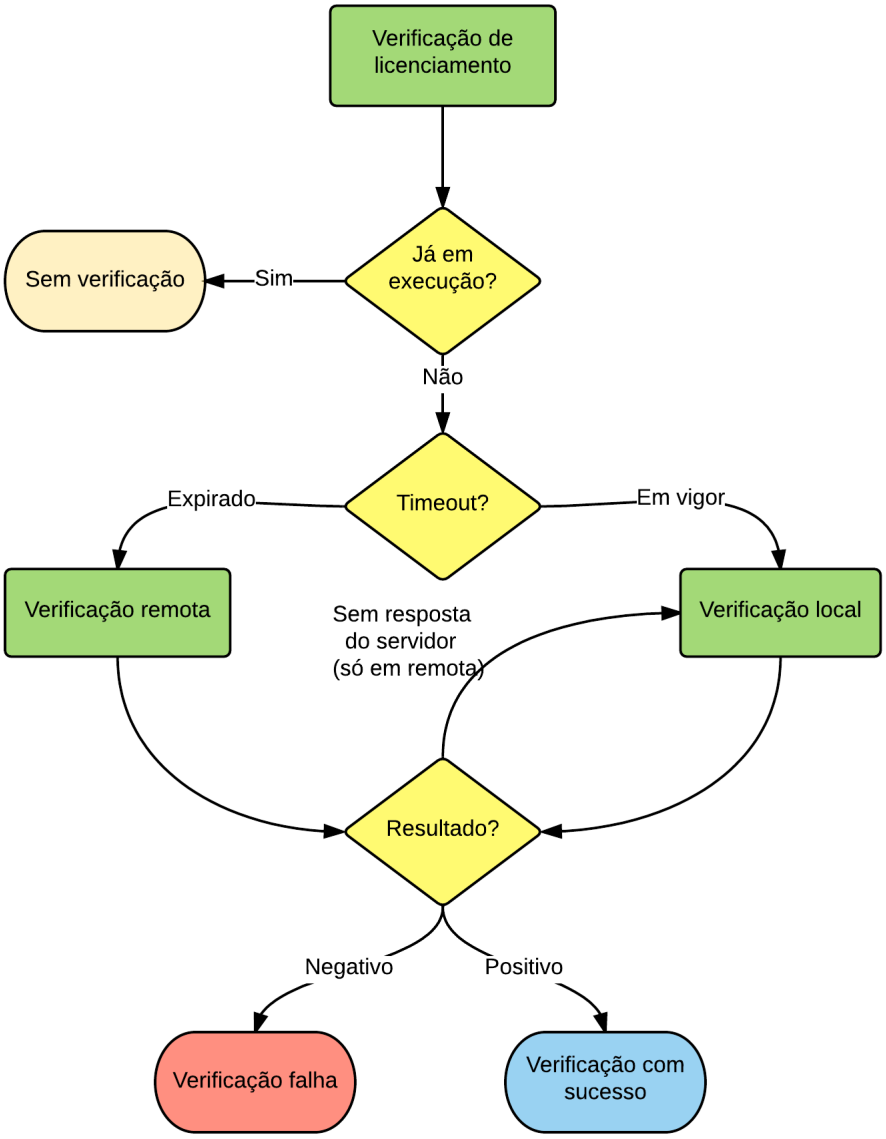

Figura 3.8: Fluxograma de uma verificação de licenciamento

licença está encriptada é necessário realizar uma desencriptação para obter a informação contida na licença, assim sendo, o módulo de licenciamento utiliza o identificador do dispositivo para esse efeito.

Estando o ficheiro da licença encriptado com o identificador, significa que só o dispositivo com o identificador correto é que poderá aceder à aplicação associada à licença, impossibilitando que caso haja a cópia da licença para outro dispositivo, que não o de ativação, este possa executar a aplicação. A desencriptação da licença é realizada uma vez por execução da aplicação, para reduzir os tempos de acesso ao sistema, sendo a informação guardada em memória.

Sempre que uma verificação local é realizada, os dados da licença em memória são comparados com dados disponibilizados pelo sistema operativo, as comparações efetuadas são as seguintes:

- Tempo de expiração, na licença - **T**
- Data atual - **T'**
- Lista de sistemas operativos, na licença - **SO**
- Nome do sistema operativo do dispositivo - **SO'**

$$\text{Validade} = \text{T'} \leq T \otimes SO' \in SO$$

3.5.2.2 Verificação remota

Além das verificações locais são efetuadas verificações remotas, ao servidor, para garantir a autenticidade da licença, presente no dispositivo. Numa verificação deste tipo, a licença já se encontra desencriptada e guardada em memória, visto que são realizadas verificações locais antes da primeira verificação remota. Este tipo de verificação é realizado com um intervalo mínimo entre execuções, para reduzir o *overhead* de uma comunicação a um servidor, fixado em 25 segundos pelo *license.js*. Caso o dispositivo não se consiga conectar com o servidor, o módulo de licenciamento realiza somente verificações locais, havendo no entanto sempre a tentativa de fazer-se remotamente se assim for permitido.

- Licença com identificador enviado - **LIdent**
- Lista de licenças, na BD - **ListLic**
- Tempo de expiração, na licença - **T**
- Data atual - **T'**
- Subscrição ativa - **S**
- Lista de sistemas operativos, na licença - **SO'**
- Lista de sistemas operativos, na BD - **SO'**

$$\text{Validade} = LIdent \in ListLic \otimes \exists S \otimes T' \leq T \otimes SO \equiv SO'$$

O módulo de licenciamento inicia a comunicação com o servidor enviando os dados da licença, ao que o servidor realiza uma comparação, apresentada acima, entre os dados guardados na base de dados e os recebidos, transmitindo ao módulo de licenciamento o resultado da operação.

3.6 Segurança

Embora o módulo de licenciamento, juntamente com a instrumentação do código da aplicação, seja suficiente para controlar o acesso a uma aplicação *JavaScript*, existem ameaças que põem em causa a estabilidade e segurança do *license.js*.

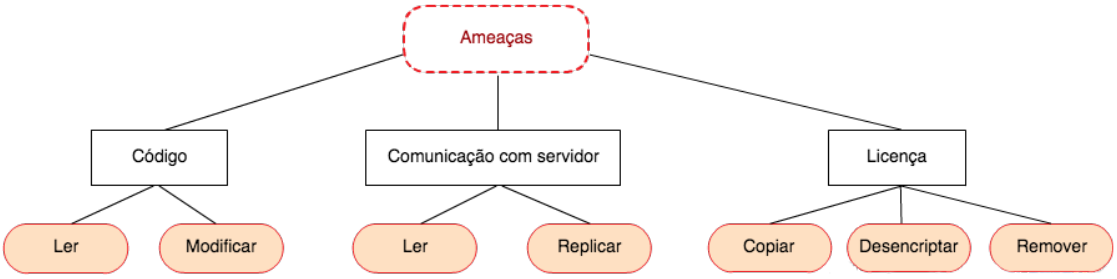

Figura 3.9: Ameaças ao *license.js*

A Figura 3.9 apresenta as principais ameaças à integridade do *license.js*, com uma divisão das ameaças ao código, à comunicação com o servidor e com a licença.
Caso um atacante consiga chegar ao código *JavaScript* da aplicação, pode analisá-lo e perceber quais os algoritmos utilizados e as estruturas de dados utilizadas para a troca de informação com o servidor. Esta análise cria outra vulnerabilidade, a modificação do código, em que o atacante tem a possibilidade de transformar o código para fins maliciosos, como a desativação do licenciamento providenciado pelo *license.js*.

As comunicações com o servidor providenciam outro ponto crítico para a segurança do *license.js*, visto que caso haja uma falha neste componente, o atacante passa a ter controlo quase total da aplicação, visto que um dispositivo não é considerado um ambiente totalmente seguro de execução, e a aplicação passa a ser verificada apenas localmente ou *tampering* de informação com o servidor. Neste caso, uma das principais ameaças é a leitura dos dados trocados entre servidor e módulo de licenciamento, que podem revelar informação importante para o licenciamento, como direitos de utilização ou chaves usados em algoritmos pelo código. A réplica de mensagens também pode afetar o sistema, pois caso um atacante utilize uma mensagem que sabe que a resposta do servidor é positiva, numa verificação, pode enviar essa mensagem em todas as comunicações ao servidor, para produzir o mesmo resultado em todas as situações.

O último componente crítico é a licença, isto é, o ficheiro que é guardado no dispositivo do Consumidor, após uma ativação, e que contém os direitos de utilização da aplicação. A cópia,

a desencriptação e a remoção desta podem causar entraves ao funcionamento correto do mecanismo de licenciamento. Se um consumidor copiar a licença para outro dispositivo e alterar os identificadores de acordo com o dispositivo de ativação, fica com acesso à aplicação, ou se conseguir desencriptar a licença pode alterar os dados lá colocados e voltar a encriptar, não estando o servidor ciente desta alteração.

De seguida são expostos alguns mecanismos implementados no *license.js*, que visam criar barreiras contra as ameaças referidas.

3.6.1 One Time Password (OTP)

Um algoritmo OTP é um gerador de *passwords* únicas como [Hal94] específica. Num sistema deste tipo as *passwords* são geradas com base numa *seed*, representada por uma *string*, que produz resultados diferentes a cada execução do algoritmo.

Existem dois tipos de algoritmos OTP, o HOTP e TOTP. Num algoritmo HOTP [MBH+05], ou *HMAC-based One-Time Password*, é implementado um contador que, consoante o número do contador, produz uma *password* diferente, sempre com base na *seed*. O algoritmo *Time-base One-time Password*, ou TOTP [MMPR11], tem um funcionamento similar ao HOTP, no entanto as *passwords* são geradas com base no tempo e não num contador. Nesta versão, existe uma janela de um tempo pré-definido, de 60 segundos no *license.js*, em que as *passwords* são renovadas, isto é, se num determinado momento uma nova *password* é gerada, sabe-se que exatamente 60 segundos adiante esta vai expirar, e uma nova vai substituí-la.

No *license.js* o OTP é utilizado para combater a réplica de mensagens para o servidor. Quando uma nova licença é criada, gera-se uma *seed* aleatória, e guarda-se na licença.

Tal como a Figura 3.10 apresenta, o dispositivo tem acesso à *seed* pois esta encontra-se na licença, que lhe é transmitida. Com este acesso, sempre que existe um pedido de verificação, por parte do módulo de licenciamento, é executado o algoritmo TOTP, com a *seed* como argumento, e incluída no corpo da mensagem HTTP a *password* gerada. O servidor encarrega-se então de gerar do seu lado uma *password* tal como no dispositivo, quando a mensagem de verificação é recebida, e compara a sua *password* com a recebida, em que caso sejam iguais a verificação pode continuar, mas que, caso contrário, a verificação é terminada e uma resposta negativa é transmitida ao dispositivo. Na última situação, a verificação é terminada pois se as *passwords* forem diferentes, significa então que o tempo em que foram geradas difere em pelo menos 60 segundos, sugerindo que a *password* recebida seja uma réplica, ou que o relógio do dispositivo esteja dessincronizado com o do servidor, sendo necessário uma sincronização manual deste.

O servidor guarda ainda todas as últimas *passwords* recebidas, que sejam válidas, para cada licença, permitindo excluir a vulnerabilidade de dois dispositivos, com o mesmo identificador, executarem a aplicação ao mesmo tempo, pois vão eventualmente enviar *passwords* geradas na mesma janela de tempo, rejeitando o servidor as *passwords* repetidas, o que impede o funcionamento normal da aplicação nos dispositivos a infringir o licenciamento. Este pormenor ajuda a colmatar as vulnerabilidades em praticamente todos os campos, no código, na comunicação e

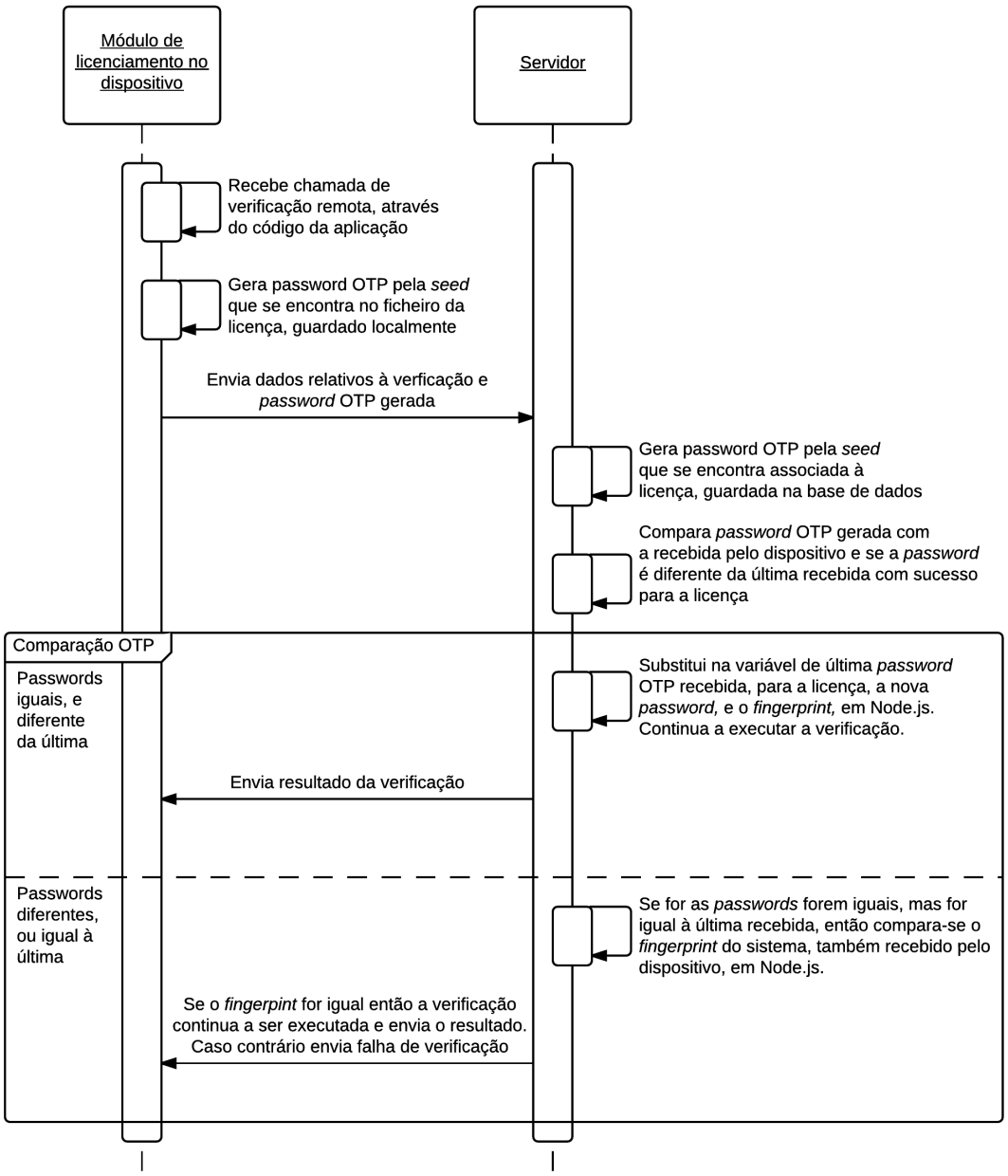

Figura 3.10: Mecanismo OTP

licença, já que mesmo que a proteção na aplicação seja quebrada, a execução da aplicação em massa, com o mesmo identificador, não vai ser permitida.

No *Node.js* está presente uma funcionalidade em que o facto de o servidor guardar as últimas *passwords* OTP é prejudicial, fazer-se *clustering* de uma aplicação. Caso um consumidor deseje executar a sua aplicação em diferentes processos ou *threads*, ao mesmo tempo, o *license.js* irá produzir resultados negativos nas verificações, porque as várias instâncias da aplicação a correrem em simultâneo vão gerar *passwords* OTP iguais na mesma janela de tempo, levando o servidor a deduzir que são dois dispositivos, com o mesmo identificador, a correrem a aplicação simultaneamente, e não várias instâncias no mesmo dispositivo. Para evitar esta situação, o *license.js* utiliza outros dados do sistema, providenciados pelo *Node.js*, como dados sobre o CPU, *hostname*, memória total e detalhes do sistema de ficheiros, para criar uma hash SHA-256 com a concatenação dos dados numa *string* - uma *fingerprint* do sistema. A *fingerprint* adiciona uma camada adicional de dados recebidos pelo servidor, e que permite a execução das verificações, eliminando as respostas negativas por OTP, nos casos das várias instâncias no mesmo dispositivo, pois mesmo que sejam geradas *passwords* OTP idênticas, a *fingerprint* é utilizada para se perceber se a aplicação está a ser executada no mesmo dispositivo ou não.

3.6.2 Licença

A licença é um ponto crítico do mecanismo de licenciamento, estando presente no dispositivo do Consumidor, que não é um ambiente seguro, pode ser facilmente ser acedida e informações importantes, como a *seed* OTP, serem lidas ou mesmo modificadas. Para evitar isto, a licença é encriptada com uma chave simétrica, o identificador do dispositivo, com o algoritmo AES.

O AES é um algoritmo largamente utilizado para a encriptação de dados nos sistemas atuais, que encripta e desencripta com base numa só chave, e fundamenta-se na encriptação por blocos, sendo o sucessor do DES, um dos standards de encriptação mais antigos, e que aceita chaves de 128, 192 ou 256 *bits* [DR98]. O *brute-forcing* à licença é considerado infazível, pois o AES é considerado um dos algoritmos simétricos mais seguros, tal como [AHH08] analisa. A escolha do *license.js* deste algoritmo simétrico tem também em a conta o seu desempenho. Mais rápido que um algoritmo assimétrico, o AES equilibra o custo de uma maior segurança com o tempo de encriptação, tendo um menor desempenho que o DES, por exemplo, mas por outro lado tendo uma segurança muito maior [NJ05]. Algumas análises de desempenho [NJ05, MSAM14] que comparam os algoritmos de encriptação mais utilizados na atualidade, indicam que o AES é uma escolha correta para uma encriptação segura de dados.

Outros pontos com relevância para a proteção da licença, como estrutura, é resistência à cópia e eliminação da licença. Embora outros mecanismos, como o OTP, ajudem a resistir a várias ameaças da licença, incluído a cópia, a sua intervenção não é direita. Um mecanismo direto contra a cópia é a própria encriptação da licença com identificador do dispositivo como chave. Ao executar-se a aplicação noutro dispositivo, que não o de ativação, a desencriptação da licença falha, dado que o identificador no dispositivo é diferente, e como tal não é a chave correta para o processo de desencriptação. O *license.js* não impede a remoção da licença do dispositivo, no entanto não

permite a execução da aplicação caso esta já se encontre ativada previamente e a licença não exista localmente. Para evitar que uma remoção, propositadamente ou acidental, bloqueie para sempre os serviços da aplicação, o *license.js* encarrega-se de tentar obter informação da licença através do servidor sempre que esta não exista ou seja inválida, havendo então um bloqueio do acesso à aplicação caso a licença seja removida, mas com alternativa para a recuperação do acesso.

3.6.3 Ofuscação

O código *JavaScript* é interpretado em tempo real, como um *script*, em que não há compilação de código fonte como em outras linguagens, e que por sua vez facilita a tarefa de aceder e modificar o código fonte de uma aplicação, se esta estiver nesta linguagem. Este fator é bastante preocupante no *license.js*, já que um atacante tem ao seu dispor os meios para aceder ao código da aplicação sem grandes restrições, por exemplo em *Node.js* onde a própria aplicação está diretamente disponível no sistema de ficheiros, e pode analisar e modificar o código para retirar as barreiras que o *license.js* impõe para o licenciamento, muito rapidamente.

O *license.js* dificulta a leitura do código através da ofuscação do código fonte das aplicações que protege. A ofuscação [CTL97] é o ato de transformar o código de forma a ser difícil para um humano entendê-lo, que pode alterar o fluxo de execução, mas que não altera a finalidade do código. Para realizar esta operação, o *license.js* recorre às funcionalidades do *JScrambler*, que implementa muitas das transformações expostas por [S+13]. Todos os ficheiros *JavaScript* de uma aplicação são ofuscados, por predefinição, no entanto, caso o produtor assim o entenda, pode-se fazer com que o *license.js* ignore ficheiros e diretórios, para obter um maior desempenho. As ofuscações são realizadas após todas as injeções e integrações da aplicação com o *license.js*.

Não obstante, apesar dos ficheiros estarem ofuscados, não significa que estejam completamente ilegíveis e protegidos. Um atacante tem acesso total ao código, e como tal pode analisar as transformações aplicadas e fazer *reverse engineering* para obter o código original. Contudo, as transformações do *JScrambler* são bastante fortes, e sendo utilizados vários tipos de transformações em conjunto, apresenta um custo muito elevado para um atacante desofuscar, tornando a ofuscação um método bastante viável para a proteção do código fonte *Javascript*, e assim é utilizada no *license.js*.

3.6.4 HTTPS

A troca de informação entre dispositivo e servidor é essencial para o mecanismo de licenciamento poder funcionar corretamente, pois mesmo quando o dispositivo não tem acesso ao servidor, sabe-se que, pelo menos a ativação foi realizada através da comunicação ao servidor. Posto isto, é necessário haver uma proteção das mensagens trocadas entre servidor e dispositivo, para evitar ataques *Man-in-middle*, em que um atacante à escuta no canal de onde as mensagens são trocadas, pode capturar mensagens e analisar ou modificar o conteúdo destas, revelando dados importantes como chaves de encriptação, *seeds*, direitos de utilização ou identificadores.

A forma mais fácil de implementar uma proteção para a comunicação é encriptar as mensagens trocadas, e dado que o servidor oferece uma API REST, significa que existe suporte para HTTPS por TLS. O HTTPS por TLS é especificado em [Res00] e o *Node.js*, onde o servidor é implementado, tem suporte para o mesmo. Neste protocolo o servidor tem um certificado, devidamente assinado por uma entidade acreditada, que dá oportunidade que os clientes que a ele façam pedidos possam verificar o certificado, e como tal a sua identidade, e estabelecer uma sessão criptográfica, onde todas as mensagens são encriptadas, e que deste modo, dificulta um ataque *Man-in-middle*. Este protocolo é considerado seguro, havendo no entanto pequenas falhas de segurança, principalmente nas entidades responsáveis por assinar os certificados e que representam um ponto crítico de todo o sistema PKI em que o HTTPS se baseia [AAVEVE14].

Com efeito, o *license.js* utiliza o protocolo HTTPS para proteger as suas mensagens, havendo um certificado presente no seu sistema de ficheiros, emitido por um entidade autorizada. Os módulos de licenciamento têm todos suporte para HTTPS nas suas comunicações. Embora uma infra-estrutura PKI preveja que tanto o servidor como o dispositivo devem ter certificados, a necessidade de gerar certificados assinados por entidade autorizada para cada dispositivo, e o facto de o ambiente de execução dos dispositivos, para todas as plataformas, não ser considerado seguro, fazendo com que guardar o certificado e chaves privadas, localmente, seja inviável, é motivo para só o servidor utilizar certificados, havendo outros métodos, como o identificador, para autenticar o dispositivo.

3.7 Conclusões

Para a gestão de licenciamento de aplicações *JavaScript* especifica-se o *license.js*, que oferece uma proteção para *Node.js* e *Cordova*, ambas escolhas propícias pela sua popularidade e funcionalidades que dispõem para aplicar licenciamento. A ferramenta disponibiliza uma área para os produtores gerirem as suas aplicações e especificarem as licenças a emitir para os consumidores, e integra com as aplicações diversas transformações e módulos que, em conjunto, verificam os direitos de utilização estipulados.

A ferramenta conta com uma arquitetura servidor-cliente, em que o servidor é o ponto central que gere o licenciamento, para todas as aplicações e dispositivos, havendo uma estreita comunicação entre este e os módulos de licenciamento. O facto de haver um sistema deste género implica algumas vulnerabilidades que o *license.js* precisa de combater, utilizando diversos mecanismos como OTP ou ofuscação de código para aumentar em grande quantidade o custo que um atacante necessita para quebrar o licenciamento, que o *license.js* providencia.

As restrições de cada plataforma e sistemas operativos forçam a que o *license.js* especifique várias estratégias para alguns passos do licenciamento, como a ativação. Na ativação poderia ser utilizada apenas a estratégia correspondente ao *iOS*, em que um utilizador introduz manualmente um *token* na aplicação, que recebe por algum meio via produtor, e a aplicação é única para todos os consumidores. Todavia, procurou-se por explorar as possibilidades que cada plataforma dá para reduzir a intervenção do Consumidor no licenciamento, como em *Node.js*, onde é gerada

um módulo de licenciamento único para cada consumidor, não necessitando do Consumidor ter que memorizar um *token*, mas que não é possível num sistema operativo como o *iOS* em que o *Cordova* opera.

Capítulo 4

Resultados

O objetivo do *license.js* é providenciar uma proteção do licenciamento para uma gama de aplicações *JavaScript*, em determinadas plataformas. É necessário que haja uma validação desta proteção, através da análise das funcionalidades que o *license.js*, retirando-se conclusões a partir dos dados obtidos. Este capitulo apresenta uma análise de segurança ao *license.js*, expondo os resultados para cada tipo de análise, e as conclusões que se retiram deles.

A análise de segurança tem em vista o estudo de quais as estratégias de ataque que um utilizador malicioso pode tentar realizar, e de que forma a solução implementada - o *license.js* - resiste. Nesta secção são apresentados vários diagramas que representam um leque de estratégias de ataque, e são discutidas as defesas do *license.js* relativamente a cada estratégia.

Embora uma proteção viável seja o ponto mais importante do *license.js*, é necessário também que esta proteção não afete, em demasia, o desempenho das aplicações protegidas. São executados então uma série de testes de desempenho, em aplicações criadas unicamente para o processo de teste de desempenho e também em casos reais. Na secção de desempenho são apresentados os dados produzidos pelos testes, e é feita uma análise do que estes representam acerca do impacto do *license.js*.

4.1 Análise de segurança

Numa ferramenta como o *license.js*, que visa criar uma proteção de licenciamento para aplicações *JavaScript*, é útil perceber quais as estratégias que um atacante pode utilizar para quebrar o licenciamento. Foi então feito um estudo de várias estratégias, através de uma série de árvores de ataque, apresentadas nas seguintes subsecções, que a partir de um objetivo, como ler dados da licença, ou bloquear chamadas de licenciamento, são especificados que caminhos um atacante pode tomar para chegar ao seu objetivo, e se é possível concretizar-se a estratégia, ou se o custo é elevado. Esta análise ajuda também a que em futuras iterações da ferramenta se possa entender quais as vulnerabilidades ainda existentes, e se podem ser colmatadas.

Nesta secção são apresentadas as árvores de ataque criadas, em forma de diagrama, e discutidos alguns pontos importantes sobre as estratégias. Em cada nó é indicado se é possível realizar-se essa operação, marcado com um **P** (de possível), e no caso contrário com um **I** (de impossível), sendo estas indicações passadas para os nós pais, que só são possíveis caso pelo menos um dos nós filhos seja possível, ou se obrigatoriamente todos os filhos forem possíveis, na situação de filhos executados em cadeia.

4.1.1 Aceder a dados na licença

Uma ameaça para o *license.js* é um atacante poder aceder a dados contidos na licença. A licença contém informação importante, como chaves simétricas para o OTP e direitos de utilização, que o utilizar poderia modificar para as verificações locais. A Figura 4.1 apresenta um diagrama com esta ameaça.

Para concretizar o seu desejo, um atacante pode optar por desencriptar a licença ou ler as mensagens trocadas com o servidor, que contêm esta informação da licença. No entanto, para esta última opção o *license.js* resiste através da implementação de HTTPS nas mensagens trocadas, o que faz com que o conteúdo esteja sempre encriptado e o atacante não tenha acesso direto, embora existam hipóteses de este mecanismo ser violado [JU12]. Se a sua tentativa for pela desencriptação direta do ficheiro da licença, este necessita de primeiro localizar o ficheiro, que só conseguirá se tiver acesso completo ao sistema. Não obstante, é necessário que o tenha conhecimento sobre qual a chave de desencriptação e qual o algoritmo de encriptação, sendo o *Brute-force* uma opção não viável [AHH08], pela segurança que o AES proporciona, podendo no entanto este desofuscar o código da aplicação e ter este conhecimento, mas sendo uma opção extremamente custosa a nível temporal, pela dificuldade em perceber e decompor as transformações realizadas, faz com que a intenção de um atacante seja possível, mas com um custo elevado.

4.1.2 Bloquear chamadas de licenciamento

Como é mostrado na Figura 4.2, outra vulnerabilidade para o *license.js* poderia ser a existência de uma estratégia que bloqueasse as chamadas de licenciamento no código da aplicação, permitindo assim que esta tivesse um funcionamento esperado, mas sem aplicar os direitos de utilização. O *license.js* resiste a estratégia de remoção através da utilização da ofuscação, visto que para remover as chamadas, um atacante necessita de compreender o código e depois remover todas as referências existentes, em todos os ficheiros da aplicação, não bastando remover unicamente os ficheiros do módulo licenciamento, pois ao existirem chamadas não existentes no código, este quebra. Tal como na estratégia anterior, a ofuscação pode ser ultrapassada, mas o tempo necessário é elevado, principalmente com as transformações aplicada pela ferramenta *JScrambler* que apoia o *license.js* neste campo.

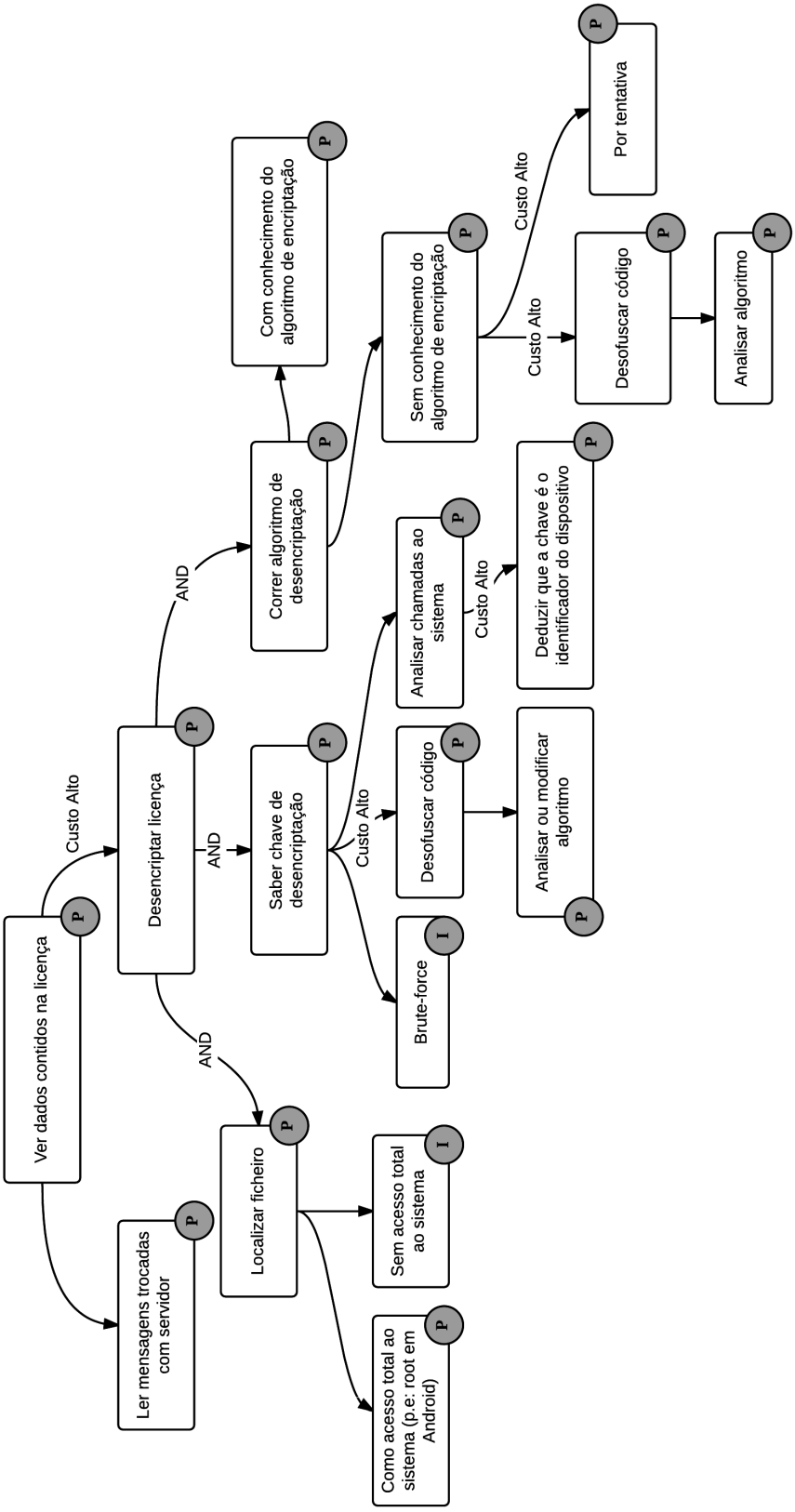

Figura 4.1: Aceder a dados na licença

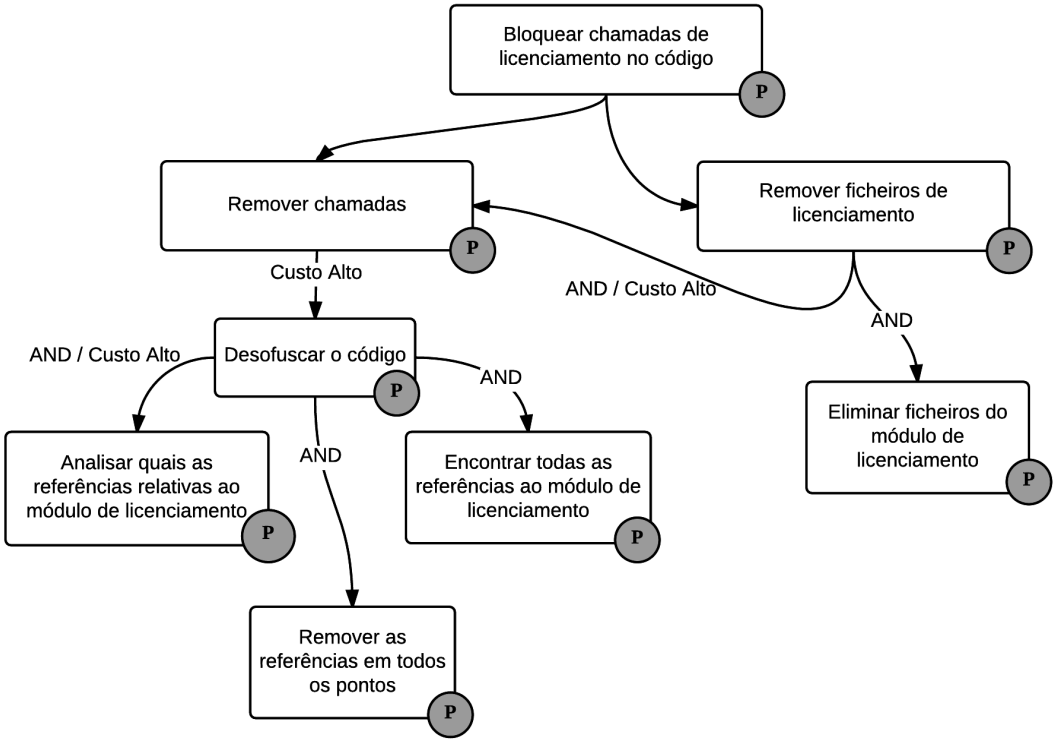

Figura 4.2: Bloquear chamadas de licenciamento

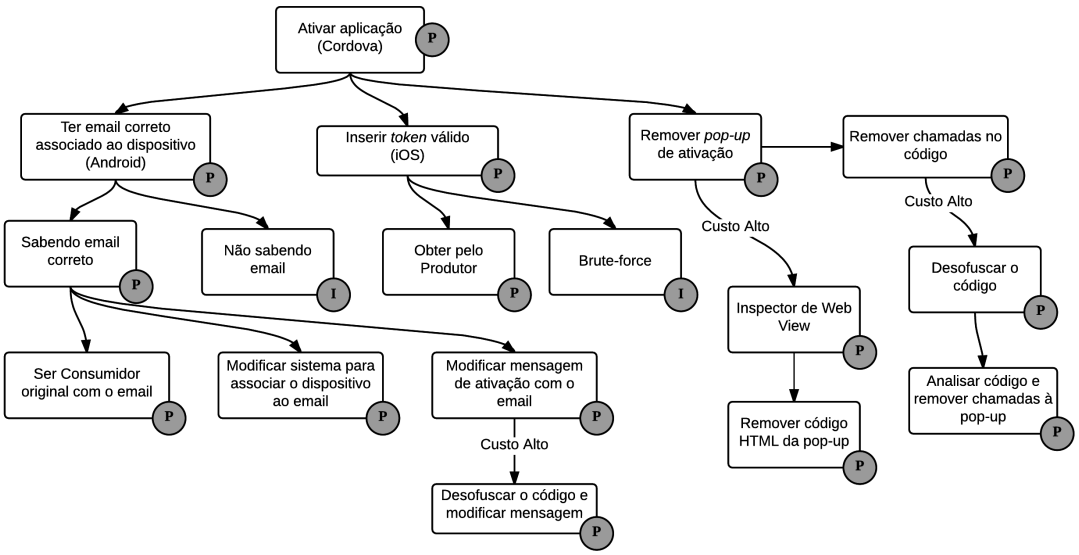

Figura 4.3: Ativação em Cordova

4.1.3 Ativação em Cordova

Para ativar-se uma aplicação em *Cordova* pode-se seguir a via normal, com a introdução de um *token* válido, para *iOS*, ou ter-se um *email* correto, para *Android*. Pode-se ainda remover a *pop-up* de ativação, no caso de *iOS*, com recurso de um inspetor de elementos HTML da *Web View*, por exemplo através do dispositivo ligado a um computador, e assim remover o elemento HTML da *pop-up* e aceder à aplicação, ou então removendo todas as chamadas de licenciamento, mas que, como se analisou na estratégia anterior, implica um custo muito alto. A remoção de *pop-ups* também implica um custo alto, pois a cada verificação irá ativar outra vez a inserção de *pop-ups* na aplicação, que bloqueiam o funcionamento normal, não sendo uma só operação para toda a execução.

Com efeito, para um atacante chegar ao seu objetivo de aceder à aplicação, ou faz pela via correta, com um *email* associado ao dispositivo de forma válida, ou com um *token* válido, ou então a única possibilidade é em *Android* modificar o sistema para ter o *email* correto, ou desofuscar o código e assim modificar a mensagem que é enviada para o servidor, mas que implica saber à parte qual o email correto associado à licença a ativar, tornando a situação muito mais difícil, como se pode observar na Figura 4.3.

4.1.4 Ativação em Node.js

Observando-se o diagrama apresentado na Figura 4.4, tal como em *Cordova*, se um atacante quiser aceder a uma aplicação, necessita de ter um *token* válido para a ativar. Visto que a aplicação está inicialmente encriptada com uma chave simétrica com AES, que só é transferida com um *token* válido, e não sendo o brute-force uma opção, só um *token* válido possibilita a desencriptação. Este *token* é injetado pelo servidor módulo, para cada consumidor, o que faz com que caso um atacante tenha acesso a outro módulo de ativação, de outra aplicação, mesmo desofuscando o código, iria necessitar de saber qual o *token* correto para injetar no módulo, o que invalida esta estratégia de ataque.

4.1.5 Receber resposta positiva de verificação

Se a intenção de um atacante for receber uma resposta positiva, numa verificação remota, este pode seguir o caminho de contactar o servidor original, como acontece por predefinição, ou criar a sua própria versão do servidor para servir a aplicação. A segunda hipótese implica que os *endpoints* REST sejam iguais aos do original e que a estrutura das mensagens trocadas seja a mesma, ora um atacante só conseguirá fazer isto se conseguir compreender o código, tendo que desde logo desofuscar, o que tem um custo muito alto. Para criar os *endpoints* pode no entanto perceber que chamadas são feitas ao servidor, através de uma aplicação a correr num dispositivo de forma normal. Com efeito, o atacante pode criar a sua própria versão do servidor e responder a pedidos de verificação de forma positiva, mas além do alto custo, também faz com que tivesse que desviar as chamadas ao servidor original, pela aplicação, para o servidor personalizado, o que faz com que alterações na rede também tivessem que ser realizadas.

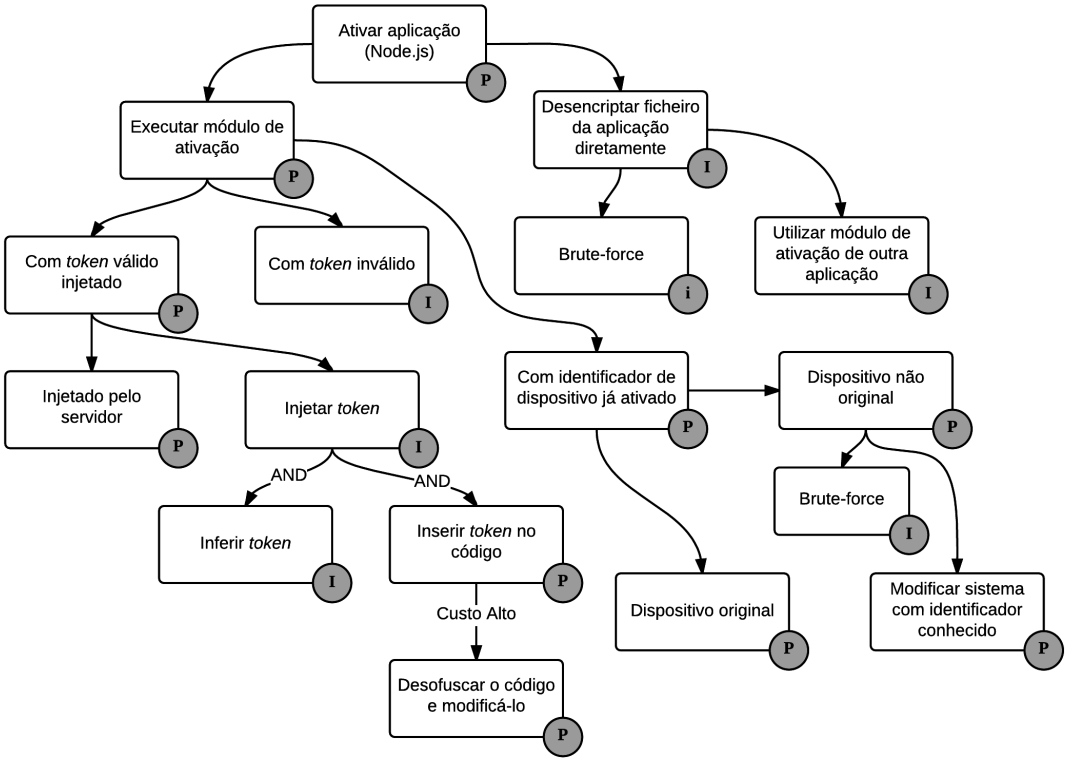

Figura 4.4: Ativação em *Node.js*

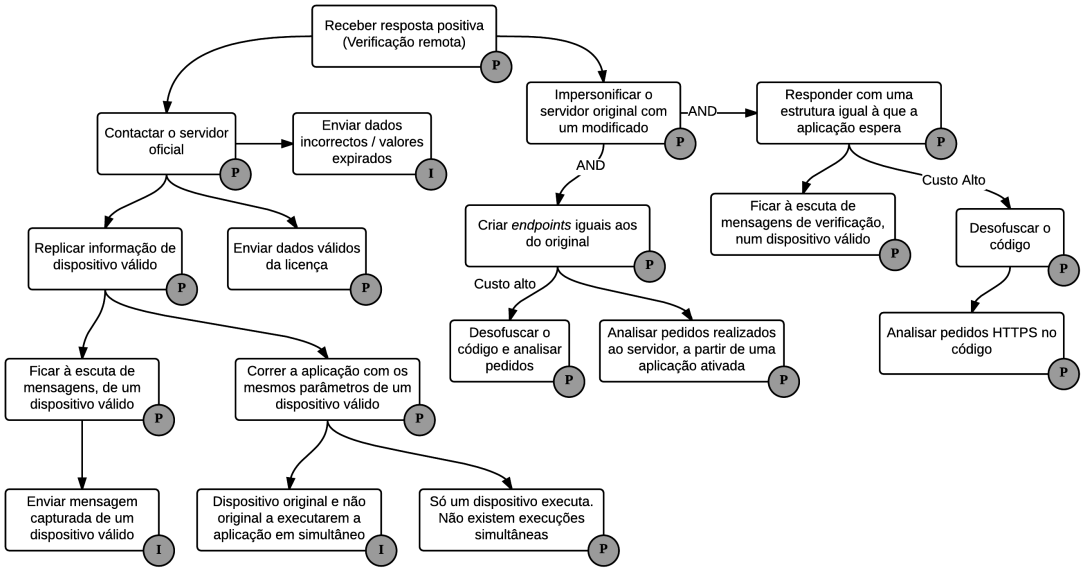

Figura 4.5: Receber resposta positiva de verificação

Na situação de contactar o servidor oficial para a verificação, o atacante necessita, obrigatoriamente, de enviar dados válidos para obter a validação que pretende, e portanto ou tem um dispositivo válido a correr a aplicação ou então tem que replicar a informação de um dispositivo válido no dispositivo que não o é. O atacante não pode replicar mensagens pois, além de estarem protegidas por HTTPS, o sistema OTP não deixa que isto aconteça. Caso consiga modificar o sistema do dispositivo malicioso para ter os mesmos dados que o dispositivo original, então pode executar a aplicação, mas não simultaneamente com o dispositivo original, pois uma vez mais, o sistema OTP não permite que mais que um dispositivo executa uma aplicação, com a mesma licença, ao mesmo tempo. A Figura 4.5 mostra os caminhos possíveis para se ter uma resposta positiva.

4.1.6 Executar aplicação copiada

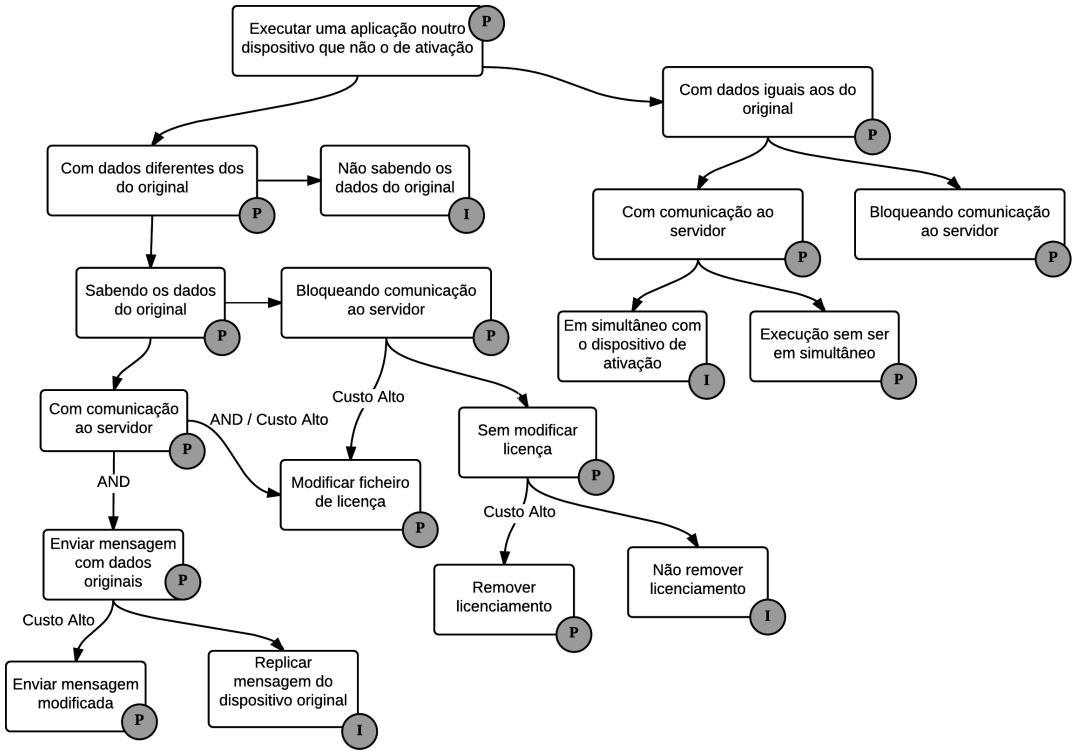

Figura 4.6: Executar aplicação copiada

Numa situação mais geral, caso alguém queira executar a aplicação, integrada com o *license.js*, noutro dispositivo que não o de ativação, pode fazê-lo, mas o *license.js* dificulta de forma impetuosa esta situação. Caso o dispositivo malicioso não tenha os mesmos dados que o original, o atacante precisa de saber os dados do original obrigatoriamente, o que só por si pode ser difícil, por o atacante não saber quais os dados necessários, mas podendo saber se desofuscar o código e analisá-lo, no entanto, tal como nas outras estratégias, esta desofuscação apresenta-se com um

custo elevado. Na eventualidade de saber os dados, pode optar por bloquear a comunicação ao servidor e ter só verificações locais, ou comunicar com o servidor e enviar mensagens modificadas, no entanto, ambas as hipóteses tem um custo muito elevado, como já se analisou em outras estratégias.

Se um atacante conseguir modificar o seu dispositivo para ter os dados iguais ao de ativação a tarefa torna-se mais fácil, podendo o dispositivo executar a aplicação da mesma forma que o original. Todavia, o mecanismo OTP impede que vários dispositivos executem a aplicação ao mesmo tempo, podendo o atacante bloquear a comunicação com o servidor, e assim servir-se das verificações locais apenas, que dão resultados positivos por os dados serem iguais. Esta situação pode-se apresentar um problema em algumas situações, mas o produtor pode tomar medidas que obriguem a aplicação a contactar o servidor ou estar online para funcionar, ou então optar-se por uma solução unicamente online, em casos onde existe uma urgência muito grande em aplicar corretamente o licenciamento, mas que são hipóteses a ter em contas em iterações futuras do *license.js*. Um diagrama detalhado com as estratégias que o atacante pode tomar é exposto na Figura 4.6.

4.2 Desempenho

O desempenho de uma aplicação pode ser um fator essencial. Dado que a natureza das aplicações que o *license.js* protege é desconhecida, é importante que o *trade off* entre proteção e desempenho não seja elevado, para não quebrar os princípios da aplicação.

Foram realizados dois tipos de testes, um com a execução de uma bateria de algoritmos, e outra com a execução de um caso de uso real, um jogo. Esta diferenciação de situações permite tirar ilações em casos onde há um esforço máximo para completar uma tarefa pelo sistema, a bateria de algoritmos, e uma situação real, onde, normalmente, o esforço necessário não é tão elevado. Os testes destinaram-se a cobrir todas as plataformas, *Node.js* e *Cordova*, tendo os dispositivos de execução as caraterísticas apresentadas pela Tabela 4.1.

Para obter dados que pudessem dar resultados objetivos e que validassem o *license.js*, os testes são executados em vários níveis de transformação de código e integração com o módulo de licenciamento. Este níveis compreendem-se em código original, sem qualquer tipo de integração com o *license.js*, em código integrado com o módulo de licenciamento e com chamadas injetadas no código, e com o código transformado e integrado da mesma forma que o último ponto, mas ofuscado, sendo a versão final que pode ser entregue a um consumidor. As injeções no código estão dispostas com uma probabilidade de 100%, para testar o desempenho do código no seu estado mais transformado.

4.2.1 Bateria de algoritmos

Pela execução de uma série de algoritmos em sequência pode-se utilizar ao máximo alguns dos recursos de um sistema, como o *workload* de um processador. Esta execução permite que

Sistema Operativo	CPU	RAM	Tipo	Plataforma
OS X 10.10	Intel i5-4200U 1,6Ghz até 2.6Ghz - Dual Core	8GB	Nativo	*Node.js*
iOS 8.3	Intel i5-4200U 1,6Ghz até 2.6Ghz - Dual Core - Dual Core	8GB	Emulador	*Cordova*
Android 4.2.2	Snapdragon S4 Plus 1Ghz - Dual Core	1GB	Nativo	*Cordova*

Tabela 4.1: Sistemas de testes

se obtenham dados importantes sobre como uma aplicação se comporta em casos extremos de utilização.

4.2.1.1 Algoritmos e métricas

Foram utilizados 3 tipos de algoritmos, com diferentes naturezas. O primeiro algoritmo é o *Quick Sort*, que organiza listas de números, colocando-os em ordem crescente. Depois deste é executado um algoritmo que lista todos os números primos até um máximo, definido no código em 100 000, o *Sieve of Eratosthenes*. Por fim encontra-se um algoritmo de compressão, o *LZW*, responsável por comprimir uma *string*.

As métricas utilizadas são relativas ao espaço temporal que é necessário para executar um algoritmo. É registado o tempo, em milissegundos, de execução, e também o número de operações por segundo que foram realizadas nessa execução. Estes dados possibilitam um análise posterior entre diferentes situações, mas que tenham as mesmas variáveis de execução, como a comparação do tempo de execução entre a aplicação sem licenciamento e com licenciamento.

4.2.1.2 Testes

Os testes realizados foram distribuídos pelos sistemas apresentados na Tabela 4.1, com diferentes níveis de licenciamento. Para a execução em cadeia dos algoritmos foi utilizado uma biblioteca *JavaScript* com o nome *benchmark.js* [Mat15], que pela execução em cadeia, revela estatísticas sobre o teste. Foi também imposto um tempo de pelo menos 120 segundos de execução do teste, em que ao acabar um ciclo de testes outro é iniciado, tentando-se sempre correr o maior número de ciclos no tempo mínimo estipulado. Este tempo mínimo é definido também para permitir ao módulo de licenciamento fazer verificações remotas, que têm um *timeout* de 25 segundos, e

também para baixar variância dos tempos de cada ciclo, para cada algoritmo, em valores mínimos de 1%, de forma a ter resultados mais estáveis e que possam ser reproduzidos.

Nos testes as aplicações encontram-se num estado de verificação, já estando ativadas, pois o uso das aplicações é maioritariamente após um estado de ativação, em casos de uso. As tabelas seguintes apresentam os resultados para *Node.js* em OS X, e para *Cordova*, em *iOS* e *Android*. É apresentada em cada célula dos resultados a média do número de operações por segundo e do tempo de execução, de cada algoritmo.

Licenciamento	Quick Sort	Sieve	LZW
Sem Licenciamento	704,909 ops/sec (0.0014ms)	493 ops/sec (2.0300ms)	7,151 ops/sec (0.1398ms)
Com Licenciamento	513,500 ops/sec (0.0019ms)	491 ops/sec (2.0366ms)	7,412 ops/sec (0.1349ms)
Com Licenciamento e Ofuscação	56,085 ops/sec (0.0178ms)	13.65 ops/sec (73.2481ms)	4,696 ops/ (0.2130ms)

Tabela 4.2: Teste de algoritmos em *Node.js*

Como se pode observar na Tabela 4.2, foram realizados os testes com os 3 algoritmos, para os 3 níveis de licenciamento, em *Node.js*. A diferença entre o uso de licenciamento e com licenciamento é mais significativa no *Quick Sort*, com um decréscimo de 27,15% de operações por segundo, mas que é expectável dada a introdução de novas operações durante a execução, pelo módulo de licenciamento. Para o resto dos algoritmos, nestes dois níveis, a diferença de desempenho é marginal. A ocorrência de uma grande quebra de desempenho é pelo uso da ofuscação nos ficheiros da aplicação, com uma redução de cerca de 92%, 97% e 34% relativamente à execução dos algoritmos com o código não transformado e sem licenciamento.

Licenciamento	Quick Sort	Sieve	LZW
Sem Licenciamento	271,361 ops/sec (0.0037ms)	86.19 ops/sec (11.6025ms)	9,592 ops/sec (0.1043ms)
Com Licenciamento	144,060 ops/sec (0.0069ms)	84.98 ops/sec (11.7670ms)	9,617 ops/sec (0.1040ms)
Com Licenciamento e Ofuscação	28,131 ops/sec (0.0355ms)	12.27 ops/sec (81.4980ms)	5,273 ops/sec (0.1896ms)

Tabela 4.3: Teste de algoritmos em *iOS*

A situação para *iOS* é muito semelhante ao teste para *Node.js*, como é mostrado na Tabela 4.3. No entanto, o desempenho é menor no *Quick Sort* para sem licenciamento e com licenciamento, com uma diminuição em 46,91% do segundo em relação ao primeiro, de cerca 271 mil operações

por segundo para 144 mil. A diferença é marginal para o resto dos algoritmos para estes dois níveis de licenciamento, havendo até um aumento de desempenho de 2.6%, mas que é descartado com variações em torno dos 1%. Tal como na plataforma para OS X, a ofuscação tem um impacto muito grande neste *benchmark*, com uma redução em 93% no *Quick Sort* para o código sem licenciamento, e até de 97% no algoritmo *Sieve of Eratosthenes*.

Licenciamento	Quick Sort	Sieve	LZW
Sem Licenciamento	59,884 ops/sec (0.0167ms)	111 ops/sec (9.0136ms)	1,138 ops/sec (0.8784ms)
Com Licenciamento	41,171 ops/sec (0.0243ms)	95.99 ops/sec (10.4178ms)	1,166 ops/sec (0.8579ms)
Com Licenciamento e Ofuscação	4,134 ops/sec (0.2419ms)	2.37 ops/sec (422.7432ms)	640 ops/sec (1.5613ms)

Tabela 4.4: Teste de algoritmos em *Android*

O último caso de testes foi realizado em *Android*, na plataforma *Cordova*, e tal como os outros, os resultados são muito idênticos, a nível de variações de desempenho entre níveis de licenciamento. O *Quick Sort* continua a ter alguma quebra de desempenho entre sem e com licenciamento, com 31% de diminuição, e tem 93% de redução de desempenho entre sem licenciamento e com licenciamento e ofuscação. O algoritmo de *Sieve* até tem alguma redução entre os primeiros níveis de licença, de aproximadamente 13,5%, no entanto o algoritmo LZW comprova que nem em todos os casos há uma redução de desempenho, com um incremento de 2,40%, que é marginal tal como nos outros casos de *benchmarking*.

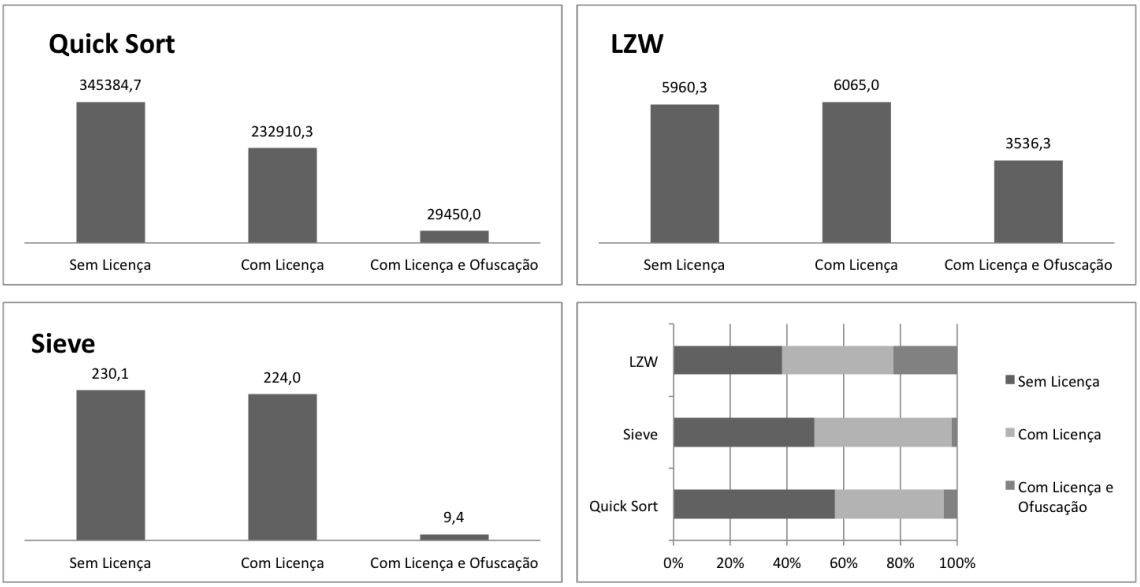

Figura 4.7: Médias dos resultados dos testes

A Figura 4.7 expõe as médias de operações por segundo dos 3 sistemas de teste, para cada algoritmo, em cada nível de licenciamento. Com estes dados conclui-se que o impacto do licenciamento é semelhante para todos os algoritmos, com o licenciamento sem ofuscação a variar em pouca percentagem em relação ao *benchmark* sem verificação de licença.

O caso do *Quick Sort*, onde se nota uma maior variação nestes dois níveis de licenciamento (32,56%), deve-se à presença de mais chamadas ao módulo de licenciamento do que em relação aos outros algoritmos, que são algoritmos unicamente sequenciais sem qualquer tipo de recursividade, enquanto o *Quick Sort* apresenta mais blocos de código, logo mais injeções de chamadas, e recursividade.

No último gráfico da Figura 4.7 consegue-se perceber que a ofuscação produz uma grande quebra em todos os algoritmos em relação ao código sem licenciamento, só tendo o *LZW* um impacto menos grave. Esta diminuição drástica no desempenho pela ofuscação dos ficheiros da aplicação era esperada num tipo de teste como este, que faz um *stress* dos recursos do sistema. As transformações de ofuscação do *JScrambler*, usado no *license.js*, introduzem um *overhead* na interpretação do código, com alterações como *splitting* de *strings*, reordenação de funções, injeção de código morto, entre outras. compreensível que a ofuscação tenha um impacto no sistema, no entanto, não está no âmbito do *license.js* melhorar as técnicas existentes no *JScrambler*. Não obstante, o *license.js* alivia este problema com a possibilidade de um produtor excluir ficheiros de serem transformados, potencializando que ficheiros com mais impacto no sistema fiquem no seu estado original.

4.2.2 Aplicação real

Embora um teste com uma bateria de algoritmos possa potencializar os recursos de um sistema, e serem retiradas métricas com valor objetivo como tempos de execução, as aplicações normalmente utilizadas por um consumidor têm fins mais comerciais e menos exigentes. É importante por isso testar o impacto no desempenho de uma aplicação criada para uso de um consumidor, com diferentes níveis de licenciamento.

4.2.2.1 Aplicação e métricas

Para testar uma aplicação real torna-se difícil conseguir métricas que comprovem, de forma válida, o seu desempenho. Não obstante, a escolha de um jogo como aplicação a testar, permite que se possa recolher informação sobre os FPS durante uma gama de tempo, comparando-se as variações entre os testes, e que é considerado como um elemento válido de avaliação. O jogo escolhido foi o *Spuds in Space* [for15], um jogo HTML e *JavaScript*, com suporte para *touch*, importante nos sistema móveis.

4.2.2.2 Testes

Tal como nos testes da bateria de algoritmos, o jogo é testado já na fase de verificação, por ser a mais importante na análise do desempenho. Os sistemas de teste são *iOS* e *Android*, não

sendo executado em *Node.js* por não ser uma aplicação objetivo para o tipo de código a proteger em *Node.js*, isto é, o jogo seria executado num *browser*, que não suporta requisitos mínimos do *license.js*, como acesso ao sistema de ficheiros. O tempo de execução de cada jogo é de cerca de 4 minutos, para garantir uma estabilidade de resultados e para permitir a comunicação do módulo de licenciamento com o servidor. Em cada sistema é executado o jogo 3 vezes durante 4 minutos, sendo os resultados, a seguir apresentados, a média dessas 3 vezes. São apresentados também os máximos e os mínimos de FPS nos testes.

Licenciamento	Mínimo	Máximo	Média
Sem Licenciamento	48	61.1	58.24
Com Licenciamento	47.8	60.7	57.22
Com Licenciamento e Ofuscação	38.6	60.5	56.68

Tabela 4.5: Resultados do jogo em *iOS*

Os resultados dos testes em *iOS*, expostos na Tabela 4.5 revelam que o impacto do licenciamento é bastante diminuto, com uma média de FPS a baixar apenas 1 unidade entre sem licenciamento e com licenciamento. Tal como nos testes da bateria de algoritmo, a ofuscação introduz mais *overhead* ao sistema, no entanto a diferença de valores continua a ser bastante reduzida, com 1.56 FPS abaixo da média do jogo a ser executado sem licenciamento. Todavia nota-se uma quebra maior nos FPS mínimos em relação ao jogo com licenciamento e ofuscação, de 9.4 valores, mas que é normal, por a dificuldade do jogo subir ao longo do tempo, e serem renderizados mais elementos, que, juntamente com a ofuscação, provocam esta quebra mais notória em algumas situações, mas que, como se pode observar pela média, ocorre em poucas situações.

Licenciamento	Mínimo	Máximo	Média
Sem Licenciamento	37.7	66.6	57.72
Com Licenciamento	33.9	66.3	54.74
Com Licenciamento e Ofuscação	32.8	65.8	52.17

Tabela 4.6: Resultados do jogo em *Android*

A Tabela 4.6 apresenta os resultados dos testes realizados em *Android*. Os dados obtidos diferem um pouco dos de *iOS*, mas que se devem a diferenças dos recursos disponibilizados para cada sistema operativo. A média de valores sofre uma queda de aproximadamente 3 FPS do jogo não modificado e com licenciamento, e de 5.5 unidades entre o primeiro e este com licenciamento e ofuscação. Os valores mínimos são mais baixos que em *iOS*, no entanto a variação entre níveis de licenciamento é menor, com apenas 4.9 FPS entre o nível mais alto de licenciamento e o jogo original, estando os máximos com valores bastante próximos. A menor memória disponibilizada para o *Android* é um fator importante a ter em conta, assim como uma menor capacidade de processamento, visto que há medida que mais elementos são introduzidos no jogo, os mínimos serão menores, e o *overhead* da ofuscação é cada vez maior.

4.3 Conclusões

Neste capítulo foi analisada a segurança do *license.js* e o seu desempenho quando integrado em aplicações. Na secção de segurança, foram estudadas várias estratégias que um atacante poderia utilizar para quebrar o licenciamento produzido pelo *license.js*, e retiradas conclusões sobre qual a proteção que a ferramenta oferece para resistir a essas estratégias, e quais as suas falhas e o custo necessário para um atacante chegar aos seus objetivos. Na análise do desempenho, foram utilizados dois casos de uso, um de *stress* ao sistema, através da execução de uma série de algoritmos, e um de uma aplicação real, um jogo. Estes dois casos de uso permitem que se perceba mais objetivamente do que pode criar *bottleneck*, e se algo a causar mau desempenho em *stress* causa também um decréscimo notável num caso de uso de um Consumidor.

Um atacante pode ter várias estratégias de ataque para quebrar o licenciamento, na análise de segurança que o *license.js* proporciona foram concluídas algumas vulnerabilidades, mas que são normais numa aplicação deste tipo, com as restrições que as plataformas impõem, e pela arquitetura da solução. A vulnerabilidade mais comum é o facto de poder-se desofuscar o código e como tal ter uma compreensão completa do funcionamento do *license.js*, tanto localmente como remotamente. No entanto, sabe-se também que o custo é muito alto [CTL97], ainda para mais quando ferramentas automáticas de desofuscação não conseguem cumprir o seu objetivo contra as transformações que o *JScrambler* aplica. No geral, a análise à segurança concluiu que o sistema é robusto, implementando uma série de mecanismos para resistir aos ataques mais comuns ou importantes.

Os resultados dos testes de desempenho mostram algo que já era esperado, o uso de ofuscação nos ficheiros da aplicação, juntamente com o módulo de licenciamento do *license.js*, produz uma redução muito elevada num teste de *stress*, pois o código é transformado em diversas formas, que reordena, modifica e até introduz elementos neste. No entanto, os dados obtidos na execução do jogo não concluem uma quebra tão drástica de desempenho com ofuscação, sendo bastante aceitáveis os valores observados. É importante também referir que o módulo de licenciamento em funcionamento, sem ofuscação, produz poucas diferenças em relação a uma aplicação não modificada, tal como era o objetivo do *license.js*. Embora a ofuscação seja um componente importante para a segurança do *license.js*, um produtor pode personalizar as escolhas de quais os ficheiros, ou pedaços de código, através de anotações, que quer transformar com injeções de licenciamento e ofuscação, e assim ter um maior controlo sobre a versão final da sua aplicação e seu desempenho.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo é feito um resumo do trabalho implementado pelo *license.js*, e são retiradas e discutidas conclusões sobre este. Há também uma secção de trabalho futuro que indica vários pontos que podem ser melhorados em futuras iterações do *license.js*, e como pode ser feito esse melhoramento.

5.1 Conclusões

Os objetivos do trabalho eram criar uma ferramenta que reunisse condições para que licenciamento fosse aplicado a aplicações *JavaScript*, com funcionalidades de gestão da proteção para quem usasse a ferramenta, e com um equilíbrio entre o custo da proteção, performance e usabilidade.

Para concretizar os objetivos, foi realizado um estudo de opções que proporcionassem a aplicação de direitos de utilização, não só para *software*, mas para todos os tipos de interesse, a nível informático. Esse estudo foi focado em DRM, que engloba grande parte da área de licenciamento, sendo apontadas várias técnicas interessantes de proteção de conteúdo e arquiteturas. Foi feita também uma pesquisa sobre quais as plataformas *JavaScript* onde possa haver um potencial interesse em aplicar licenciamento.

Através do estudo das tecnologias e plataformas, foi retirada informação para especificar-se a ferramenta. Nasceu assim o *license.js*, que integra código de licenciamento em aplicações *JavaScript* para *Node.js* e *Cordova*, e assegura que uma aplicação é executada só se os direitos de utilização, especificados por quem usa o *license.js* para submeter aplicações, estiverem a ser cumpridos. O *license.js* permite também que haja uma gestão das aplicações protegidas, através de uma aplicação *Web*, assim como das licenças e subscrições emitidas, havendo assim a possibilidade de um controlo bastante restrito de quem e onde usa uma aplicação protegida.

Após a especificação e implementação do *license.js*, foram estipulados uma série de testes para verificar a integridade e viabilidade da ferramenta. Primeiramente foi feito uma análise de

segurança, onde foram apontadas várias estratégias que um atacante poderia tomar para quebrar o licenciamento, e como o *license.js* as resiste. Depois, foram completados alguns testes de performance, com algoritmos e uma aplicação real, para testar o impacto do *license.js* em comparação às versões desprotegidas. Ambos os testes revelaram-se com resultados positivos, se por um lado existem algumas vulnerabilidades a nível da segurança, sabe-se que o custo para realizar um ataque utilizando essas vulnerabilidades é bastante elevado, e verifica-se que a performance num sistema em *stress* é afetada, pela utilização de ofuscação, mas que no entanto, numa aplicação real, o decréscimo é muito menor, estando compreendido em valores aceitáveis para uso pelo Consumidor.

Com efeito, os objetivos do trabalho foram cumpridos, com a implementação do *license.js*, criando uma solução atualmente inexistente, pela proteção de licenciamento em código *JavaScript*, reunindo uma série de técnicas e mecanismos que produzem uma ferramenta robusta e usável.

5.2 Trabalho Futuro

Embora o *license.js* tenha cumprido todos os objetivos do trabalho, existem uma série de melhoramentos que podem aumentar a eficácia ou casos de uso da ferramenta.

Um dos pontos fulcrais da segurança do *license.js* é a ofuscação, em que caso sejam derrotadas todas as técnicas de ofuscação, um atacante tem acesso a um código mais próximo do original, embora não igual, visto que algumas das transformações são irreversíveis [CTL97]. Podem ser aplicadas ao código uma série de transformações que do ponto de vista da performance são muito pesadas, como o *self-defending* do *JScrambler*, mas que elevam a fasquia da proteção. Além disto, embora tendo anotações de código, que podem ajudar a perceber e reduzir os pontos onde a performance pode ser afetada, estas têm que ser definidas pelo produtor, o que pode não ser suficiente para garantir a estabilidade de uma aplicação. Para colmatar em parte estes fatores, poder-se-á realizar uma análise do código, percebendo o *control flow* da aplicação, e exercitando o código para obter-se uma análise mais profunda, de forma a perceber quais os pontos do código com mais impacto no sistema. Esta compreensão do *control flow* pode ser concretizada, por exemplo, com o estudo da complexidade ciclomática do código [McC76], ficando-se com o conhecimento sobre quais os pontos do código mais complexos, que podem causar problemas de desempenho. Deste modo pode-se minimizar ou personalizar as transformações, tanto de licenciamento como de ofuscação, nesses pontos. Além disto, pode-se integrar uma das capacidades do *JScrambler* de anotações de código [JSc15], que analogamente ao *license.js*, possibilita a anotação de blocos de código para controlar qual o nível de ofuscação aplicado, podendo ser especificadas quais as transformações desejadas, e cobrindo casos que uma análise do *control flow* não detete.

Neste momento, o *license.js* aplica licenciamento após uma aplicação estar implementada, criando uma camada por cima do funcionamento normal desta. Para melhorar as funcionalidades do *license.js*, como a limitação de utilizadores que usam uma aplicação protegida, no mesmo dispositivo, é útil a criação de uma interface que ligasse a aplicação ao *license.js*. Esta interface seria implementada pelo *license.js*, e possibilitaria a que um produtor possa usar métodos dessa

interface para enviar informação pertinente ao servidor do *license.js*, que caso contrário não teria acesso.

A versão corrente possibilita o acesso a aplicações de forma *online* e *offline*. Esta arquitetura é boa do ponto de vista de um consumidor, que tem acesso ao conteúdo mesmo sem uma ligação à *Web*, todavia, a nível de proteção, faz com que o licenciamento tenha que ficar mais complexo, e mais vulnerável até. A ideia seria então criar uma versão do *license.js* para aplicações puramente online, ou seja que tivesse um acesso continuo a recursos remotos. Com esta hipótese poder-se-ia adotar técnicas de DRM como agentes de código, onde partes do código da aplicação não estariam presentes no dispositivo, sendo enviadas *on-the-fly* quando fossem necessárias, ou então a aplicação poderia estar sempre a reportar ao servidor vários dados, e caso não obtivesse comunicação, então quebrava.

Por fim, o *license.js* pode usufruir de uma procura mais extensa de formas de identificar dispositivos ou sistemas operativos, por exemplo associar estruturas de sistemas de ficheiros a dispositivos, ou testar chamadas ao sistema de forma a evitar que um simples *name spoofing* do sistema operativo possa esconder a verdadeira informação deste.

Referências

[AAVEVE14] Axel Arnbak, Hadi Asghari, Michel Van Eeten e Nico Van Eijk. Security collapse in the https market. *Communications of the ACM*, 57(10):47–55, 2014.

[Ado15a] Adobe. Adobe air developer center. Disponível em http://www.adobe.com/devnet/air.html, Janeiro 2015.

[Ado15b] Adobe. Adobe reader download. Disponível em http://get.adobe.com/reader/, Janeiro 2015.

[Ado15c] Adobe. Javascript for acrobat | adobe developer connection. Disponível em http://www.adobe.com/devnet/acrobat/javascript.html, Janeiro 2015.

[Ado15d] Adobe. Phonegap | home. Disponível em http://phonegap.com/, Janeiro 2015.

[Ado15e] Adobe. Scripting developer center | adobe developer connection. Disponível em http://www.adobe.com/devnet/scripting.html, Janeiro 2015.

[AHH08] Abdullah Al Hasib e Abul Ahsan Md Mahmudul Haque. A comparative study of the performance and security issues of aes and rsa cryptography. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 2, pages 505–510. IEEE, 2008.

[And06] Byron Anderson. A Primer on Copyright Law and the DMCA, 2006. URL: http://csaweb106v.csa.com/ids70/view_record.php?id=2&recnum=62&log=from_res&SID=fmcbnttokdhjuocc1gm30an8t2$\delimiter"026E30F$nhttp://www.tandfonline.com/doi/abs/10.1300/J120v45n93_10, doi:10.1300/J120v45n93_05.

[Apa15] Apache Software Foundation. Apache cordova. Disponível em http://cordova.apache.org/, Janeiro 2015.

[App15] Appcelerator Inc. Titanium mobile application development | appcelerator inc. Disponível em http://www.appcelerator.com/titanium/, Janeiro 2015.

[Aud15] AuditMark. Protect your javascript | jscrambler. Disponível em https://jscrambler.com/pt/, Janeiro 2015.

[BM07] Willms Buhse e Jan Van Der Meer. The Open Mobile Alliance Digital Rights Management. (January):140–143, 2007.

[CTL97] Christian Collberg, Clark Thomborson e Douglas Low. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand, 1997.

[DJ11] Qin Dong e Peng Yu Ji. Research on effect of digital entertainment in the digital era. *Proceedings - 2011 4th International Symposium on Knowledge Acquisition and Modeling, KAM 2011*, pages 566–567, 2011. doi:10.1109/KAM.2011.152.

[DR98] Joan Daemen e Vincent Rijmen. Aes proposal: Rijndael. 1998.

[Eri03] JS Erickson. Fair use, DRM, and trusted computing. *Communications of the ACM*, 46(4):34, April 2003. URL: http://portal.acm.org/citation.cfm?doid=641205.641228, doi:10.1145/641205.641228.

[Esp15] Esprima. Esprima. Disponível em http://esprima.org, Junho 2015.

[est15] estools. estools/escodegen. Disponível em https://github.com/estools/escodegen, Junho 2015.

[fee15] feedic. htmlparser2. Disponível em https://www.npmjs.com/package/htmlparser2, Junho 2015.

[for15] forkem. Spuds in space. Disponível em https://github.com/forkem/spuds-in-space, Junho 2015.

[FPV98] Alfonso Fuggetta, Gian Pietro Picco e Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=685258, doi:10.1109/32.685258.

[FZ08] Min Feng e Bin Zhu. A DRM system protecting consumer privacy. In *2008 5th IEEE Consumer Communications and Networking Conference, CCNC 2008*, pages 1075–1079. Ieee, 2008. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4446542, doi:10.1109/ccnc08.2007.244.

[Goo15] Google. Google apps script — google developers. Disponível em https://developers.google.com/apps-script/, Janeiro 2015.

[GPB12] Stefan Vladimir Ghiţ, Victor Valeriu Patriciu e Ion Bica. A new DRM architecture based on mobile code and white-box encryption. In *2012 9th International Conference on Communications, COMM 2012 - Conference Proceedings*, pages 303–306. IEEE, June 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6262567, doi:10.1109/ICComm.2012.6262567.

[GVS13] Richa Garg, Ravi Sankar Veerubhotla e Ashutosh Saxena. AtDRM. In *Proceedings of the 6th ACM India Computing Convention on - Compute '13*, pages 1–6, 2013. URL: http://dl.acm.org/citation.cfm?id=2522599http://dl.acm.org/citation.cfm?doid=2522548.2522599, doi:10.1145/2522548.2522599.

[Hal94] Neil Haller. The s/key one-time password system. In *In Proceedings of the Internet Society Symposium on Network and Distributed Systems*, pages 151–157, 1994.

[HHM13] Henning Heitkötter, Sebastian Hanschke e Tim A Majchrzak. Comparing cross-platform development approaches for mobile applications. *Web Information Systems and Technologies*, 140:120–138, 2013. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0003904502990311, doi:10.1007/978-3-642-36608-6.

[HLJ+14] Jheng-jia Huang, Pei-chun Lu, Wen-shenq Juang, Chun-i Fan, Zheng-yang Lin e Chun-hung Lin. Secure and efficient digital rights management mechanisms with privacy protection. *Journal of Shanghai Jiaotong University (Science)*, 19(4):443–447, August 2014. URL: http://link.springer.com/10.1007/s12204-014-1523-5, doi:10.1007/s12204-014-1523-5.

[IDC15] IDC. Smartphone os market share, q1 2015. Disponível em http://www.idc.com/prodserv/smartphone-os-market-share.jsp, Junho 2015.

[Joy15] Joyent. Node.js. Disponível em http://nodejs.org/, Janeiro 2015.

[JSc15] JScrambler. Ignore code blocks - jscrambler help | jscrambler. Disponível em https://jscrambler.com/pt/help/javascript_obfuscation/ignore_code_blocks, Junho 2015.

[JU12] Jeff Jarmoc e Dell SecureWorks Counter Threat Unit. Ssl/tls interception proxies and transitive trust. *Black Hat Europe*, 2012.

[LHJ+03] Junseok Lee, Seong Oun Hwang, Sang-Won Jeong, Ki Song Yoon, Chang Soon Park e Jae-Cheol Ryou. A DRM Framework for Distributing Digital Contents through the Internet. *ETRI Journal*, 25(6):423–436, December 2003. URL: http://etrij.etri.re.kr/Cyber/BrowseAbstract.jsp?vol=25&num=6&pg=423, doi:10.4218/etrij.03.0103.0024.

[Lin15] Linux Foundation. Tizen | an open source, standards-based software platform for multiple device categories. Disponível em https://www.tizen.org/, Janeiro 2015.

[LLJ10] Lei Li, Quan Liu e Xuemei Jiang. USB key-based dual-factor dynamic authentication scheme. In *Proceedings - 2010 International Conference on Computational Intelligence and Security, CIS 2010*, pages 446–449, 2010. doi:10.1109/CIS.2010.103.

[LLZL08] Ping Li, Zhengding Lu, Fuhao Zou e Hefei Ling. A DRM system based on mobile agent for digital rights redistribution. *Wuhan University Journal of Natural Sciences*, 13(4):475–480, August 2008. URL: http://link.springer.com/10.1007/s11859-008-0419-3, doi:10.1007/s11859-008-0419-3.

[LMS05] Paul J Leach, Michael Mealling e Rich Salz. A universally unique identifier (uuid) urn namespace. 2005.

[LSnS03] Qiong Liu, Reihaneh Safavi-naini e Nicholas Paul Sheppard. Digital Rights Management for Content Distribution. 21, 2003.

[LZ13] Zhi-Chun Li e Chunxiao Zhang. Digital Rights Management System Based on PKCS#12. In *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 634–637. Ieee, October 2013.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6846719, doi:10.1109/IIH-MSP.2013.163.

[Mat15] Mathias Bynens and John-David Dalton. Benchmark.js. Disponível em http://benchmarkjs.com/, Junho 2015.

[MBH+05] D M'Raihi, M Bellare, F Hoornaert, D Naccache e O Ranen. Hotp: An hmac-based one-time password algorithm. *The Internet Society, Network Working Group. RFC4226*, 2005.

[McC76] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.

[Mic15a] Microsoft. Microsoft playready - home. Disponível em http://www.microsoft.com/playready/, Fevereiro 2015.

[Mic15b] Microsoft. Windows phone. Disponível em http://www.windowsphone.com/, Janeiro 2015.

[MMPR11] David M'Raihi, S Machani, M Pei e J Rydell. Totp: Time-based one-time password algorithm. *Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC*, 6238, 2011.

[MnMnPP01] Antonio Maña, Antonio Maña, Ernesto Pimentel e Ernesto Pimentel. An Efficient Software Protection Scheme. In *In Proceedings of the 16th International Conference on Information Security: Trusted Information*, pages 200–1, 2001. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.154, doi:10.1.1.16.154.

[Mon15] MongoDB Inc. Mongodb. Disponível em http://www.mongodb.org/, Janeiro 2015.

[MSAM14] Ranjeet Masram, Vivek Shahare, Jibi Abraham e Rajni Moona. Analysis and comparison of symmetric key cryptographic algorithms based on various file features. *International Journal of Network Security & Its Applications (IJNSA)*, 6(4), 2014.

[Mun05] T. Mundt. Location dependent digital rights management. *10th IEEE Symposium on Computers and Communications (ISCC'05)*, (Iscc), 2005. doi:10.1109/ISCC.2005.96.

[Mut15] Mutable Inc. Openipmp. Disponível em http://sourceforge.net/projects/openipmp/, Janeiro 2015.

[NJ05] Aamer Nadeem e M Younus Javed. A performance comparison of data encryption algorithms. In *Information and communication technologies, 2005. ICICT 2005. First international conference on*, pages 84–89. IEEE, 2005.

[Nod15] NodeBots. Nodebots - the rise of js robotics. Disponível em http://nodebots.io/, Janeiro 2015.

[npm15] npm Inc. npm. Disponível em https://www.npmjs.com/, Janeiro 2015.

[Ope15a] Open Mobile Alliance Ltd. Open mobile alliance mobile phone standards specifications | oma. Disponível em http://openmobilealliance.org/, Janeiro 2015.

[Ope15b] OpenSDRM. Opensdrm - open rights management. Disponível em http://www.opensdrm.org/, Janeiro 2015.

[PBTG11] Victor Valeriu Patriciu, Ion Bica, Mihai Togan e Stefan Vladimir Ghita. A generalized DRM architectural framework. *Advances in Electrical and Computer Engineering*, 11(1):43–48, 2011. URL: http://www.aece.ro/abstractplus.php?year=2011&number=1&article=7, doi:10.4316/aece.2011.01007.

[Pur15] Pur3 Ltd. Espruino - javascript for microcontrollers. Disponível em http://www.espruino.com/, Janeiro 2015.

[QYTC13] Cheng Qu, Yinyan Yu, Zhi Tang e Xiaoyu Cui. A DRM Scheme Using File Physical Information. *International Journal of Cyber-Security and ...*, 2(1):56–69, 2013. URL: http://sdiwc.net/digital-library/a-drm-scheme-using-file-physical-information.html.

[RC05] Jason F. Reid e William J. Caelli. Drm, trusted computing and operating system architecture, 2005.

[Res00] Eric Rescorla. Http over tls. 2000.

[S+13] Filipe Manuel Gomes Silva et al. Ferramenta de ofuscação de código javascript. 2013.

[SHC07] Hung-Min Sun Hung-Min Sun, Chi-Fu Hung Chi-Fu Hung e Chien-Ming Chen Chien-Ming Chen. An Improved Digital Rights Management System Based on Smart Cards. *2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, pages 308–313, 2007. doi:10.1109/DEST.2007.371989.

[SLYK09] Ming-Kung Sun Ming-Kung Sun, Chi-Sung Laih Chi-Sung Laih, Hong-Yi Yen Hong-Yi Yen e Jyun-Rong Kuo Jyun-Rong Kuo. A Ticket Based Digital Rights Management Model. *2009 6th IEEE Consumer Communications and Networking Conference*, pages 1–5, January 2009. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4784774, doi:10.1109/CCNC.2009.4784774.

[Son15] Sony DADC. Home - sony dadc securom. Disponível em https://www2.securom.com/, Fevereiro 2015.

[SSSN07] Martin Jan Surminen, Nicholas Paul Sheppard e Reihaneh Safavi-Naini. Location-based DRM using WiFi access points. In *ISCIT 2007 - 2007 International Symposium on Communications and Information Technologies Proceedings*, pages 882–886, 2007. doi:10.1109/ISCIT.2007.4392140.

[Sta15] StarForce Technologies Ltd. Starforce technologies is a protection of software and applications against illegal copying and cracking. cd, dvd, audio and video files copy protection. protection of e-mails from hackers. Disponível em http://www.star-force.com/, Fevereiro 2015.

[Sud13] Hasshi Sudler. Effectiveness of anti-piracy technology: Finding appropriate solutions for evolving online piracy. *Business Horizons*, 56(2):149–157, 2013. URL: http://dx.doi.org/10.1016/j.bushor.2012.11.001, doi:10.1016/j.bushor.2012.11.001.

[SY06] S. R. Subramanya e Byung K. Yi. Digital rights management, 2006. doi:10.1109/MP.2006.1649008.

[Syv94] Paul Syverson. A taxonomy of replay attacks [cryptographic protocols]. *Proceedings The Computer Security Foundations Workshop VII*, pages 187–191, 1994. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=315935, doi:10.1109/CSFW.1994.315935.

[Tec15] Technical Machine. Tessel. Disponível em https://tessel.io/, Janeiro 2015.

[TV10] Stefan Tilkov e Steve Vinoski. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14:80–83, 2010. URL: http://doi.ieeecomputersociety.org/10.1109/MIC.2010.145, doi:10.1109/MIC.2010.145.

[US11] Mohab Usama e Mohamed Sobh. Software Copy Protection and Licensing based on XrML and PKCS#11. In *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, pages 856–861, 2011. doi:10.1109/PACRIM.2011.6033007.

[VS11] Ravi Sankar Veerubhotla e Ashutosh Saxena. A DRM framework towards preventing digital piracy. In *Proceedings of the 2011 7th International Conference on Information Assurance and Security, IAS 2011*, pages 1–6, 2011. doi:10.1109/ISIAS.2011.6122785.

[Wak15] Wakanda SAS. wakanda. Disponível em http://www.wakanda.org/, Janeiro 2015.

[YFL09] Aimin Yu, Dengguo Feng e Ren Liu. TBDRM: A TPM-based secure DRM architecture. In *Proceedings - 12th IEEE International Conference on Computational Science and Engineering, CSE 2009*, volume 2, pages 671–677. Ieee, 2009. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5283799, doi:10.1109/CSE.2009.15.

[YLL14] Chih-ta Yen, Horng-twu Liaw e Nai-wei Lo. Digital rights management system with user privacy, usage transparency, and superdistribution support. *International Journal of Communication Systems*, 27(10):1714–1730, October 2014. URL: http://doi.wiley.com/10.1002/dac.2431, doi:10.1002/dac.2431.

[Zha11] Xiao Zhang. A Survey of Digital Rights Management Technologies. 2011.

[ZL11] Zhiyong Zhang e Tao Huang Lili, Danmei Niu. A Novel DRM Security Scheme and its Prototype System Implementation, 2011. doi:10.4156/jdcta.vol5.issue11.42.

[ZS12] Jianrui Zhang e Mark Stamp. Software Activation Using Multithreading. *International Journal of Computer Network and Information Security*, 4(November):1–17, 2012. URL: http://www.mecs-press.org/ijcnis/ijcnis-v4-n12/v4n12-1.html, doi:10.5815/ijcnis.2012.12.01.

[ZYXZ09] Yi Chun Zhang, Lei Yang, Pin Xu e Yong Song Zhan. A DRM authentication scheme based on smart-card. In *CIS 2009 - 2009 International Conference on Computational Intelligence and Security*, volume 2, pages 202–207, 2009. doi:10.1109/CIS.2009.182.