Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

AUTENTICAÇÃO FIDO2 EM DISPOSITIVOS ANDROID

ÉLTON CARPINTEIRO PASTILHA

Leiria, Março de 2022

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

AUTENTICAÇÃO FIDO2 EM DISPOSITIVOS ANDROID

ÉLTON CARPINTEIRO PASTILHA
Número: 2192651

Projeto realizado sob orientação do Professor Doutor Miguel Monteiro de Sousa Frade e do Professor Doutor Patrício Rodrigues Domingues.

Leiria, Março de 2022

AGRADECIMENTOS

Agradeço a minha família e amigos que poderão apoiar-me no desenvolvimento do projecto em diversas formas. Agradeço aos orientadores pelo apoio e orientação no desenrolar do projecto.

RESUMO

Nos sistemas informáticos a autenticação de utilizadores é um elemento essencial na segurança da informação e também uma das principais linhas de defesa. A autenticação de utilizadores é base da qual são construídos a maioria dos mecanismos de controlo de acesso e de responsabilidade dos utilizadores, bem como da contabilização da utilização dos serviços. Para o utilizador poder aceder a informação, tem de comprovar que é quem diz ser, passando pelo processo de duas etapas, identificação e verificação. O sistema de autenticação tradicional que faz uso do *Username* e da *Password* tem apresentado vários problemas de segurança. Esses problemas tanto são da parte do utilizador como da parte do sistema. O principal problema é o sistema de autenticação tradicional fazer uso das *Passwords*, em que essas podem ser descobertas caso haja Violação de dados (*Data breach*) e também pelas limitações próprias do ser humano que a escolhe *Passwords* simples e comuns, e que frequentemente as reutiliza. Neste trabalho pretende-se apresentar uma variedade de métodos de autenticação e também descrever como é feita a autenticação com vários fatores. Contudo, o foco deste projecto é a autenticação através do Fast IDentity Online versão 2 (FIDO2). Este protocolo traz grandes vantagens ao nível da segurança, através do uso de chaves assimétricas e uso do conceito *PasswordLess*, não requerendo uso de uma *Password* no processo de autenticação. Apresenta-se uma descrição do FIDO2, demonstra-se o seu uso para autenticação de dois fatores, como método de autenticação *PasswordLess*. Por fim, descreve-se a implementação do FIDO2 na aplicação Bitwarden (gestor de *passwords*) para Android.

Palavras-chave: Autenticação, FIDO, FIDO2, Yubikey, PasswordLess, Android, Bitwarden

ABSTRACT

In computer systems, user authentication is an essential element in information security and also one of the main defense lines. User authentication is the basis on which most access control and user responsibility mechanisms are built, as well as accounting for the use of services. For the user to be able to access the information, he must prove that he/she is who he/she says he/she is, going through the two-step process, identification and verification. The traditional authentication system makes use of *Username* and *Password* has presented several security problems. These problems are both on the part of the user and on the part of the system. The main problem is that the traditional authentication system makes use of *Passwords*, in which these can be discovered if there is a data breach and also because the user is human and he/she has limitations that lead to choose repeated, simple and common *Passwords*. In this work we intend to present a variety of authentication methods and also describe how multifactor authentication is performed. However, the focus of this project is authentication via FIDO2. This protocol brings great advantages in terms of security, through the use of asymmetric keys and the *PasswordLess* concept (without requiring the use of a *Password* in the authentication process). A description of FIDO2 is presented, its use for two-factor authentication is demonstrated, and as a *PasswordLess* authentication method. Finally, the implementation of FIDO2 in the Bitwarden application (*passwords* manager) for Android is described.

Keywords: Authentication, FIDO, FIDO2, Yubikey, PasswordLess, Android, Bitwarden

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

2FA	Two-Factor Authentication.
API	Application Programming Interface.
CTAP	Client To Authenticator Protocol.
CTAP1	Client To Authenticator Protocol versão 1.
CTAP2	Client To Authenticator Protocol versão 2.
DNS	Domain Name System.
ECC	Elliptic-curve cryptography.
FIDO	Fast IDentity Online.
FIDO1	Fast IDentity Online versão 1.
FIDO2	Fast IDentity Online versão 2.
HMAC	Hash-based Message Authentication Code.
HOTP	Hash-based One-Time Password.
HTTP	Hyper Text Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
IDN	Internationalized Domain Names.
IoT	Internet of Things.
JSON	JavaScript Object Notation.

MFA	Multi-Factor Authentication.
MITM	Man-In-The-Middle.
NFC	Near Field Communication.
OTP	One-Time Password.
PIN	Personal Identification Number.
PRNG	Pseudorandom Number Generator.
QR Code	Quick Response Code.
RSA	Rivest-Shamir-Adleman.
SMS	Short Message Service.
SQL	Structured Query Language.
SQRL	Secure Quick Reliable Login.
SSL	Secure Sockets Layer.
SSO	Single Sign On.
TLS	Transport Layer Security.
TOTP	Time-based One-Time Password.
TPM	Trusted Platform Module.
U2F	Universal 2^{nd} Factor.
UAF	Universal Authentication Framework.
URL	Uniform Resource Locator.
USB	Universal Serial Bus.
W3C	World Wide Web Consortium.

WebAuthn Web Authentication.

1

INTRODUÇÃO

Nos sistemas informáticos a autenticação de utilizadores é um elemento essencial na segurança da informação e também uma das principais linhas de defesa. A autenticação de utilizadores é base da qual são construídos a maioria dos mecanismos de controlo de acesso e de responsabilidade dos utilizadores (*accountability*) (Stallings, 2018). A definição de autenticação de utilizadores, segundo a RFC 4949 (Internet Security Glossary), é o processo de verificação da identidade reivindicada por uma entidade. Este processo divide-se em duas etapas:

- **identificação** – consiste na apresentação do valor do atributo reivindicado (por exemplo, nome de utilizador) ao subsistema de autenticação;
- **verificação** – apresentação, ou geração, de informação de autenticação (*e. g.* um valor assinado com uma chave privada) que prova a ligação entre o atributo apresentado e a identidade reivindicada;

A forma mais comum de autenticação é baseada em nomes de utilizador e uma palavra-passe. Por exemplo, a utilizadora Maria dos Prazeres pode ter como identificador o nome de utilizador `mprazeres`. O nome de utilizador tem de ser armazenado em todos os computadores que a Maria deseje usar e pode ser revelado aos administradores ou outros utilizadores, ou seja, o nome de utilizador tipicamente não é secreto. Como forma de validar a sua identidade, a Maria tem de fornecer uma palavra-passe que deve ser mantida secreta, sendo apenas conhecida pela Maria e pelo sistema informático. Assim, desde que mais ninguém conheça ou adivinhe a palavra chave da Maria, a combinação entre o identificador `mprazeres` e o verificador (palavra-passe) permitem autenticá-la e desta forma obter acesso ao sistema informático, bem como auditar as suas atividades.

O uso de nome de utilizador e respetiva palavra-passe (*password*) continua a ser o método de autenticação mais utilizado devido à sua simplicidade e baixo custo de implementação (Creese et al., 2013). Esta forma de autenticação está presente no dia-a-dia da maioria das pessoas, por exemplo para a gestão bancária, compras *online*, controlo de dispositivos Internet of Things (IoT), até às atividades de lazer e entretenimento. No entanto, este método de autenticação apresenta vários problemas que podem colocar em causa a sua segurança. Um desses problemas é o sistema

presumir que o utilizador é totalmente fiável. No entanto, como se tem observado ao longo dos anos (Taneski et al., 2019), isso não se verifica. Os utilizadores têm dificuldades em memorizar credenciais e por essa razão fazem uso de palavras-passe fracas e repetidas em diversas plataformas. Além disso, podem ser manipulados para partilhar as sua credenciais, por exemplo através de técnicas de engenharia social, *e. g. phishing* (Heartfield e Loukas, 2015). Mas os utilizadores não são o único ponto de falha, pois as plataformas e as aplicações também apresentam falhas e nem sempre conseguem proteger as credenciais dos utilizadores. Quando uma violação de dados acontece, entidades maliciosas tentam descobrir as palavras-passes através de ataques de dicionário (D. Wang e P. Wang, 2015), tabelas *rainbow* (Ah Kioon et al., 2013).

Taneski et al. (2019) apresenta um levantamento dos trabalhos científicos desde 1979 até 2019 sobre os problemas associados à autenticação através de palavras-passe. Apesar destes problemas serem estudados há várias décadas, a maioria das soluções propostas apresentam o utilizador como o problema e o elo mais fraco na segurança dos sistemas de informação (Taneski et al., 2019). As recomendações mais comuns para resolver o problema das palavras-passe são: uso de senhas que não façam parte de um dicionário; uso de caracteres especiais para aumentar a segurança da senha; criar estratagemas de memorização de senhas; e por fim usar sistemas autenticação centralizados designados Single Sign On (SSO). No entanto, as três primeiras recomendações sobrecarregam o utilizador devido à limitada capacidade de memorização dos humanos. Por outro lado, o uso de sistemas de autenticação centralizados implica que os utilizadores depositem elevada confiança nas plataformas interligadas para autenticação. Além disso, levanta ainda preocupações relacionadas com a privacidade, devido ao ponto central de acesso ficar a conhecer todos os serviços onde é efetuada a autenticação (Sun et al., 2011). Por estas razões a autenticação baseada em palavras-passe é considerada fraca do ponto de vista da segurança (Bonneau et al., 2015).

A autenticação de utilizadores pode ser realizada com base em vários factores que podem ser usados de forma isolada, ou combinada:

- factor de conhecimento – algo que utilizador sabe;
- factor de posse – algo que utilizador possui;
- factores de inerência – algo que utilizador tem nas suas características biológicas;
- factor cronológico – fazer uso da data e hora;

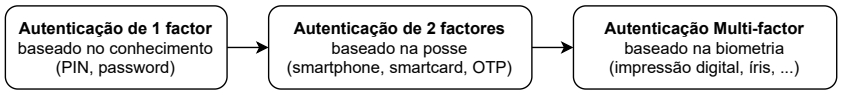

Figura 1: Evolução da autenticação de um factor até multi-factor (Petsas et al., 2015)

- factor de localização – localização geográfica do utilizador.

Assim, para remediar os problemas inerentes ao uso de senhas, muitos serviços começaram a implementar outros métodos de autenticação. Esses métodos alternativos podem ser usados isoladamente (*passwordless*) ou de forma complementar ao método tradicional (*username* e *password*) e tipicamente envolvem o uso de algo que esteja na posse no utilizador, *e. g.* o telemóvel. Esta última abordagem de autenticação é designada autenticação de dois fatores, ou Two-Factor Authentication (2FA). Posteriormente, foi proposto a autenticação multi-factor, Multi-Factor Authentication (MFA), para fornecer um grau de autenticação superior através do uso de mais do que dois factores de autenticação, *e. g.* através da leitura de impressão digital (Petsas et al., 2015). A Figura 1 ilustra a evolução da autenticação de utilizadores desde a utilização de um só factor (PIN, *password*, *etc*), passando por 2FA (smartphone, tokens USB, *etc*) até ao MFA com dados biométricos.

A integração de diversas formas de autenticação de utilizadores nas aplicações de software aumenta a sua complexidade, como também são difíceis de implementar corretamente e por isso apresentam muitos desafios aos programadores. Numa tentativa de dar resposta a estes problemas foi criado o FIDO. Criado pelo grupo FIDO Alliance, o FIDO tem como objetivos garantir a segurança, a privacidade, a usabilidade e a facilidade de integração no software (FIDO Alliance, 2020). O FIDO apresenta um conjunto de normas abertas de comunicação, escaláveis e inter-operáveis para implementação de mecanismos de autenticação forte. Atualmente o FIDO tem duas versões, a primeira versão apresenta especificações para a implementação de autenticação de dois factores. A segunda versão acrescenta a autenticação forte sem recurso a palavras-passe *passwordless*, baseada em criptografia de chaves públicas. O FIDO2, foi também padronizado pelo consórcio World Wide Web Consortium (W3C) sob o nome Web Authentication (WebAuthn) (W3C, 2021) .

1.1 OBJETIVOS DO PROJETO

A norma WebAuthn é bastante recente e por esta razão os exemplos de implementação são escassos e relativamente complexos e muitas vezes de difícil compreensão para um programador com pouca experiência nesta área. Além disso, os exemplos

existentes tendem a focar-se nos métodos de autenticação mais simples suportado pelo FIDO e não abordam a autenticação com chaves públicas armazenadas em tokens criptográficos. Assim, os objetivos deste projeto são:

- estudar e compreender as diversas formas de autenticação suportadas pelo FIDO2;
- documentar a implementação FIDO2 em várias plataformas, nomeadamente Web e Android;
- criar código de referência para programadores como exemplo de implementação de autenticação forte através do FIDO2 e recurso a *tokens* físicos;

1.2 CONTRIBUTOS

Deste projeto resultaram os seguintes contributos:

- este relatório que documenta o FIDO2, as diversas formas de autenticação suportadas, bem como as características criptográficas que garantem a segurança do mesmo;
- foi criado e publicada em *open source* uma prova de conceitos para servir de referência para os programadores que desejem implementar FIDO2 nas suas aplicações, nomeadamente para Web[1] e Android[2];
- foi partilhado a implementação do suporte de FIDO2 para o gestor de senhas *open source* Bitwarden [3], com suporte para as chaves criptográficas Yubikey[4]:
  - na infra-estrutura do servidor[5] (*backend*);
  - no serviço de web[6] (*frontend*);
  - aplicação móvel para Android[7] com suporte para leitura de chaves Yubikey com Near Field Communication (NFC);
- foi identificado e reportado um problema na biblioteca *open source* "abergs/fido2-net-lib"que realiza funções criptográficas do FIDO2, tambem foi criado uma possivel solução[8].

1 https://github.com/DestruidorPT/FIDO2-WebServer-Test
2 https://github.com/DestruidorPT/FIDO2-Android-Test
3 https://github.com/bitwarden
4 https://www.yubico.com/products/security-key/
5 https://github.com/DestruidorPT/server
6 https://github.com/DestruidorPT/web e https://github.com/DestruidorPT/jslib
7 https://github.com/DestruidorPT/mobile
8 https://github.com/DestruidorPT/fido2-net-lib

1.3 ESTRUTURA DO RELATÓRIO

Este relatório está organizado da seguinte forma, no Capítulo 2 é apresentado o estado da arte onde são abordadas as diversas formas de autenticação e autenticação multi-factores, a norma FIDO2 e ainda uma explicação sumária do funcionamento dos *tokens* Yubikey. Segue-se o Capítulo 3 com a metodologia usada para o desenvolvimento deste projeto e a descrição de cada fase do projeto. No Capítulo 4 são apresentadas as tecnologias usadas para a criação e desenvolvimento da prova de conceito para testar a implementação do FIDO2 com ferramentas que já apresentam suporte nativo para esta norma. O Capítulo 5 descreve o desenvolvimento e teste de código para ser adicionado ao gestor de senhas *open source* Bitwarden no servidor (*back end* e *front end*) e na aplicação Android. Por fim, as conclusões são apresentadas no Capítulo 6 onde se resume os resultados deste projeto, incluindo as dificuldades encontradas e trabalho futuro.

2

ESTADO DA ARTE

Tipicamente a autenticação ocorre quando um utilizador tenta aceder a um recurso informático (um computador, uma aplicação, ou serviço). O recurso exige que o utilizador forneça a identidade pela qual é conhecido no sistema, juntamente com uma prova da identidade reivindicada. A autenticação simples requer apenas uma prova, ou factor, normalmente uma palavra-passe (*password*). Para garantir segurança adicional, os sistemas podem exigir mais de um factor de autenticação, autenticação multifactor, e nesse caso terão de ser fornecidos múltiplas provas de verificação de identidade, uma por cada factor de autenticação. O uso de múltiplos factores de autenticação para provar a identidade de alguém é baseado na premissa de que uma pessoa não autorizada provavelmente não será capaz de fornecer todos os factores necessários para o acesso. Assim, se numa tentativa de autenticação pelo menos um dos componentes estiver em falta, ou se for incorreto, a identidade do utilizador não é estabelecida com um grau de certeza suficiente e o acesso ao recurso informático permanece bloqueado. Um sistema de autenticação multi-factor pode incluir um, ou mais dos seguintes factores:

- factor de conhecimento – algo que utilizador sabe, *e. g.* uma palavra-passe, um Personal Identification Number (PIN), as respostas a um conjunto pré-determinado de perguntas, *etc*. Esta é a forma mais comum de autenticação porque é simples de implementar e de baixo custo;
- factor de posse – algo que utilizador possui, *e. g. smartcards*, cartões eletrónicos, chaves físicas, dispositivos criptográficos. Este tipo de autenticador é habitualmente designado por *token*;
- factores de inerência – algo que utilizador tem nas suas características biológicas e que pode ser subdividido em:
  - estática – autenticação baseada em algo que o utilizador é (biometria estática), *e. g.* através do reconhecimento de impressões digitais, retina, face, ADN, *etc*;

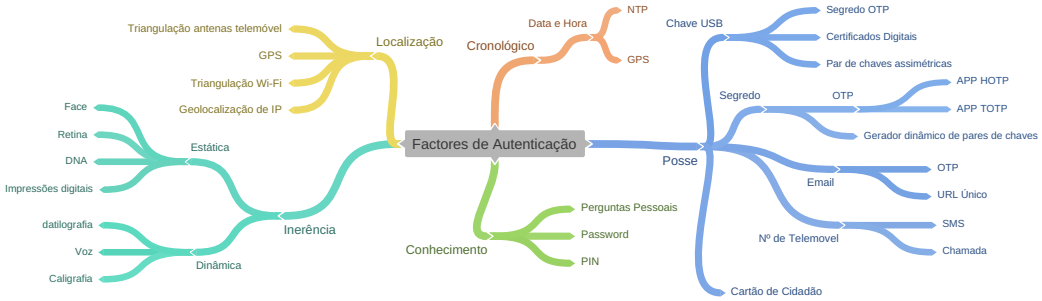

Figura 2: Factores de Autenticação.

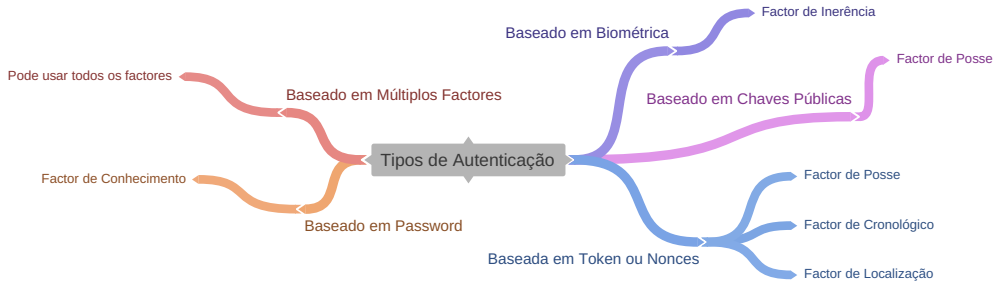

Figura 3: Tipos de Autenticação.

  - dinâmica – autenticação baseada em algo que o utilizador faz (biometria dinâmica), *e. g.* através do reconhecimento de padrões de voz, características da caligrafia, ritmo de datilografia, *etc*;

- factor de localização – localização geográfica do utilizador que pode ser de elevada precisão, por exemplo através de coordenadas GPS, ou aproximada, por exemplo através do endereço IP. Tipicamente este factor é usado em conjugação com outros, por exemplo se um utilizador estiver dentro das instalações duma empresa poderá ser-lhe solicitado apenas uma palavra-passe, ou PIN, mas se estiver fora poderá ser-lhe exigido factores de autenticação adicionais, ou até ser impedido de se autenticar se por exemplo estiver fora do país.
- factor cronológico – faz uso da data e hora, como exemplo horário atual.

As formas mais comuns de autenticação que serão referidas (Maayan, s.d.) estão representadas na Figura 2, ilustrando os principais factores de autenticação.

De seguida apresentam-se uma análise a algumas das formas mais comuns de autenticação, que fazem uso dos factores referidos anteriormente.

2.1 AUTENTICAÇÃO BASEADA EM *PASSWORD*

O método de autenticação tradicional e o mais conhecido (Maayan, s.d.), tem como base o uso uma palavra secreta (*password*), em que supostamente só o utilizador conhece. A *password* pode ter números, letras ou caracteres especiais, em grande maioria das autenticações com *password* também fazem uso do *username*, que pode ser o e-mail, número de telemóvel ou um nome de utilizador (*username*), isto permite identificar o utilizador. São poucos os casos que não usam o *username*, exemplo o PIN do cartão SIM, o PIN de desbloqueio do telemóvel e entre outros, o que se pode tirar partido dessa análise, é que não é comum utilizar o *username* em autenticações locais e que não exista mais do que um utilizador no sistema ou aplicação.

Neste método de autenticação é seguro desde que o utilizador não use *password* pequenas e simples, que não repita as mesmas credenciais em diferentes plataformas e que esses utilizadores não se deixem enganar pelos métodos de ataque como *phishing*. Mas como o ser humano tem limitações, esse tipo de situações acontece por terem dificuldade em decorar diferentes *passwords* para todas as plataformas que usa, que podem ser muitas. Além disso, também por serem facilmente manipuláveis em dar informações sensíveis a identidades maliciosas. Estas situações podem acontecer por falta de conhecimentos informáticos ou por propostas atrativas com falsas intenções.

Neste tipo de autenticação os problemas não se devem apenas às limitações dos utilizadores. Por vezes existem fugas de informação, ou violação de dados, nos servidores através de ataques cibernéticos ou por acessos ilegais dos administradores. Este tipo de ataques facilita a descoberta das *passwords* porque os servidores têm de as armazenar, tipicamente de forma criptográfica. Se as *passwords* forem roubadas, mesmo criptografadas, devem ser consideradas comprometidas, pois identidades maliciosas pode descobrir-las através de ataques de dicionário, ou de força bruta. Nestes casos ficam em risco também as contas noutros serviços onde tenham sido reutilizadas a mesma *password* que a usada no servidor comprometido.

Na Figura 4 pode-se visualizar o processo de autenticação a uma plataforma online. O Bob envia as credenciais dele para o servidor, o Servidor verifica se o utilizador existe e se a *password* está correta. Ao verificar que os dados estão corretos o servidor envia uma autorização de autenticação ao Bob.

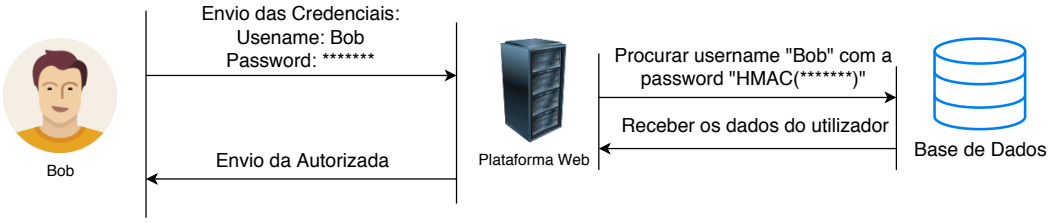

Figura 4: Processo simplificado de autenticação com *username* e *password*.

2.2 AUTENTICAÇÃO BASEADA EM MÚLTIPLOS FACTORES

A autenticação baseada em múltiplos factores não é nada mais que um conjunto de autenticações num só, ou seja, é uma autenticação que faz uso de dois ou mais factores para autenticar. Esta autenticação é efetuada linear, para que o utilizador possa autenticar tem que fazer autenticação em cada factor com sucesso, caso um factor esteja errado o utilizador não conseguira autenticar.

Esta técnica é conhecida como autenticação com múltiplos factores, também conhecida como MFA, é usada para reforçar verificação de identidade, para assim estabelecer com um grau aceitável de certeza do acesso ao recurso.

Quem define o grau de certeza aceitável é o administrador do sistema, em que o administrador decide se é necessário mais ou menos factores de autenticação, tendo em mente quanto maior o grau de certeza (maior o número de factores), menor será aceitação desse sistema de autenticação, ou seja, vai existir muitos utilizadores que se vão recusar usar esse sistema e continuar a usar um sistema de autenticação mais fraco, porque esse sistema é menos amigável e mais demorado.

Neste tipo de sistemas, normalmente um dos métodos de autenticação é o tradicional, *username* e *password*, mas não é obrigatório ter esse. Para uma boa autenticação multi-factores factores é preciso ter algo que sabemos (factores de conhecimento, exemplo *username* e *password*), algo que temos (factor de posse, exemplo chave de segurança física) e algo que somos (factores de inerência, ou biométricos, por exemplo impressão digital). Assim, mesmo que descubram o *username* e a *password* por métodos de *Phishing*, as entidades maliciosas ainda teriam de obter os factores de posse e de inerência.

Na autenticação MFA, também existe a autenticação de dois factores, conhecida como Universal 2^{nd} Factor (U2F) que só faz uso unicamente de dois factores.

O U2F trabalha da seguinte forma, primeiro utilizador coloca as credenciais na página de *login*, o servidor valida, e caso seja válido, o servidor verifica se há necessidade de solicitar o segundo método de autenticação.

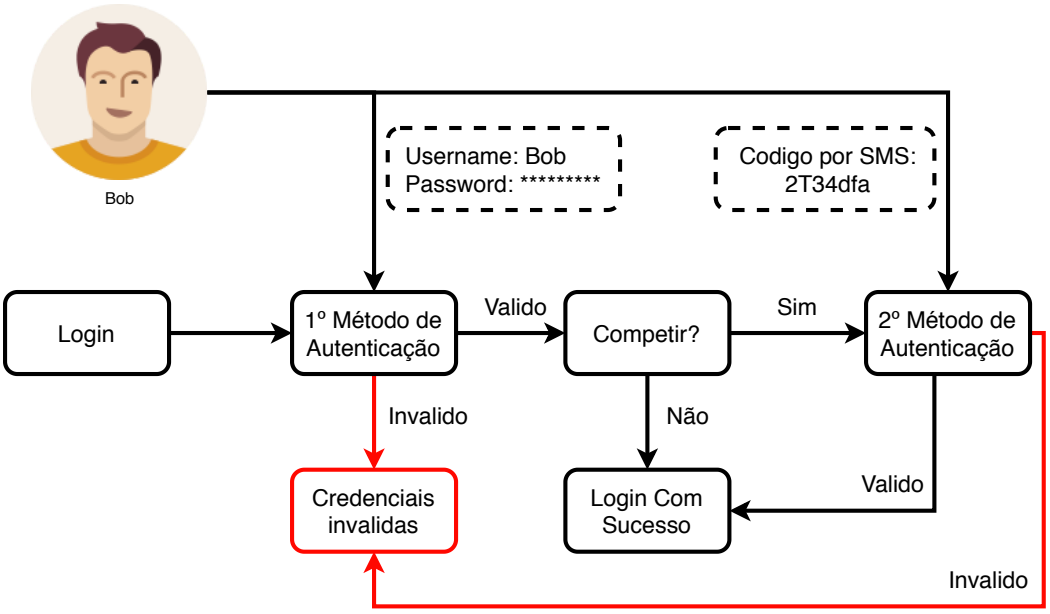

Figura 5: Processo de autenticação de dois factores.

Na necessidade de segundo método de autenticação, o utilizador providencia as credenciais do segundo método de autenticação. O processo é ilustrado na Figura 5.

2.3 AUTENTICAÇÃO BASEADA EM CHAVES PÚBLICAS

A autenticação baseada em chaves públicas permite a identificação de utilizadores, de máquinas ou de dispositivos, através do uso de criptografia assimétrica. Estas chaves são criadas através de algoritmos de criptografia assimétricos, que geram uma chave privada, que nunca deve ser partilhada e uma chave pública, que é partilhada para ajudar na identificação. Nas autenticações baseadas em chaves públicas uma parte delas faz uso das Autoridades de Certificação (CA), que têm a função de garantir que um certificado digital (chaves públicas) pertence ao titular. Esta forma de autenticação é usada por exemplo no protocolo Hyper Text Transfer Protocol Secure (HTTPS) em que os servidores web se autenticam perante os clientes.

Quando se faz uso do sistema de criptografia assimétrica é preciso ter em conta as seguintes informações, quando um conteúdo é encriptado ou assinado por uma chave privada, a chave pública pode verificar assinatura ou desencriptar o conteúdo, mas quando é a chave pública a encriptar ou assinar o conteúdo, só a chave privada pode validar assinatura e desencriptar o conteúdo.

A assinatura do conteúdo serve para garantir que aquela pessoa assinou aquele conteúdo, ou seja, validar a identidade de um cliente. É necessário usar a chave

privada para assinar, pois essa não é partilhada e chaves públicas não conseguem replicar assinatura, apenas validar.

No caso da encriptação do conteúdo, para garantir que uma só pessoa consegue aceder ao conteúdo, ou seja, criar confidencialidade de informação. É necessário encriptar o conteúdo com a chave pública, pois só a chave privada permite desencriptar a informação.

Tendo isso em mente, grande parte das autenticações baseado em chaves públicas, trabalham da seguinte forma.

Primeiro é criado pelo dispositivo do cliente um conjunto de chaves (privada e pública), e de seguida é efetuado o registo da chave pública (enviado para o servidor) e esse é guardado junto com a conta do cliente.

Segundo é efetuado o login, em que o servidor envia um conteúdo para ser assinado, o cliente assina o conteúdo com a chave privada e envia para o servidor, ao receber a chave no lado do servidor, o servidor valida assinatura usando a chave pública previamente registada. Se assinatura for valida então autenticação é bem sucedida.

Existem também tecnologias de autenticação que fazem uso de chaves públicas e privadas, como por exemplo o Secure Quick Reliable Login (SQRL)[1] e o Fast IDentity Online (FIDO). De seguida é detalhado o funcionamento do Secure Quick Reliable Login (SQRL). O FIDO será abordado com mais detalhe no capítulo 3.

2.3.1 *Secure Quick Reliable Login (SQRL)*

O SQRL vem propor o uso de chaves assimétricas sem necessitar de uma chave física. Primeiro é necessário instalar o SQRL num sistema, por exemplo num *smartphone*, ou *browser*. No processo de instalação é gerado um par de chaves – uma pública e uma privada. Depois a aplicação SQRL permite gerir as identidades armazenadas e realizar a autenticação num servidor que suporta o protocolo do SQRL.

O funcionamento do SQRL é relativamente simples e pode ser visualizado nas Figura 6 e 7. Um utilizador faz um pedido ao servidor, a solicitar a página de *login*. O servidor gera um Uniform Resource Locator (URL) e um *token* (*nonce*), e envia para o cliente. Ao ter a página de *login*, o utilizador clica no URL (exemplo na Figura 6) ou lê o Quick Response Code (QR Code) com smartphone (exemplo na Figura 7). De seguida o cliente SQRL envia o *nonce* assinado com a sua chave privada e o servidor valida a assinatura com a chave pública correspondente (previamente

1 https://www.grc.com/sqrl/sqrl.htm

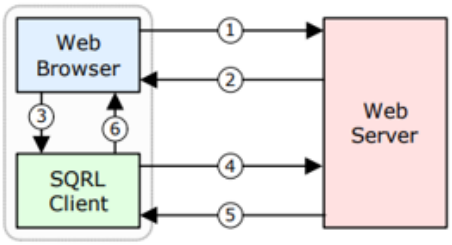

Figura 6: Interação entre *Browser*, o SQRL e o servidor (Gibson, 2019).

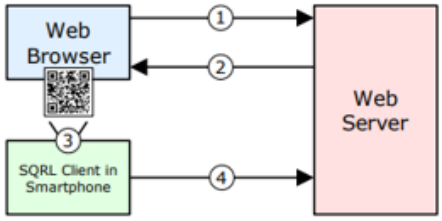

Figura 7: Interação entre *Browser*, o SQRL no Smartphone e o servidor (Gibson, 2019).

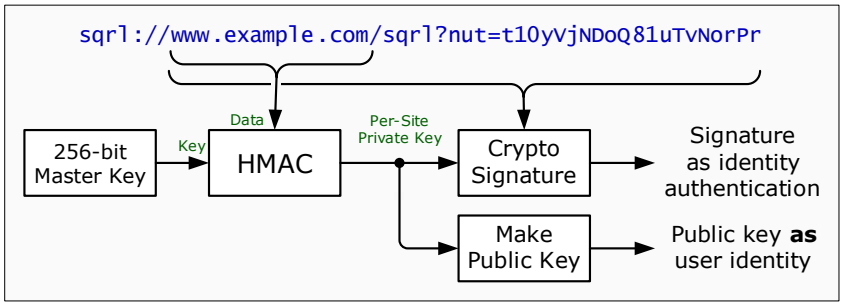

Figura 8: Diagrama criptográfico do SQRL (Gibson, 2019)

armazenada no servidor aquando do registo do cliente). Após a verificação com sucesso da assinatura, o *browser* do cliente é redirecionado para a página que quer aceder. A principal vantagem do SQRL é o facto do servidor não precisar de armazenar nenhum segredo dos clientes, apenas armazena as suas chaves públicas. Desta forma, caso ocorra alguma violação do servidor, a autenticação dos clientes não fica comprometida porque não há *passwords* envolvidas. Além disso, o par de chaves é criado de forma dinâmica e dependente do domínio onde se pretende autenticar (ver Figura 8). Desta forma garante-se que existe um par de chaves único por cada domínio (Gibson, 2019).

2.4 AUTENTICAÇÃO BIOMÉTRICA

A autenticação biométrica faz uso das características biológicas do utilizador, ou seja, faz uso de algo físico do utilizador, exemplo impressão digital, ADN, voz, face ou íris. Alguns dos tipos de sensores biométricos que existem (Babich, 2019):

- Leitores de impressão digital - este tipo de autenticação é muito popular;
- Reconhecimento facial - este tipo não é consistente e tem dificuldades em diferenciar pessoas parecidas de cara;

- Identificação de voz - utilizado pela Alexa (Amazon) e o Google Assistance para identificar o utilizador que está a falar;
- Scanners do olho - um sistema robusto, em que analisa o padrão único da íris do olho.

Esta forma de autenticação tornou-se comum com o surgimento de sensores de impressões digitais nos *smartphones* e posteriormente com a adição de sensores para reconhecimento facial.

2.5 AUTENTICAÇÃO BASEADA EM *TOKENS* OU *NONCE*

A autenticação baseada em *tokens* e autenticação baseada em *nonces*, ambos são usados para um único espaço de tempo, os *nonces* são empregues por curtos períodos temporais, relativo em segundos ou minutos. Por sua vez, os *tokens* apresentam validade temporal maior, na ordem de dias ou mesmo meses.

O *token* é um conjunto de informação codificada ou encriptada, ao contrário do *nonce* que é um valor aleatório. Ambos tendem a gerar informação única, ou seja, no caso dos *tokens* tende a ser único na base de dados do servidor e no *nonce* tende a ser único de forma a não se repetir, pelo menos num futuro próximo.

A autenticação baseada em *tokens* tem como objetivo facilitar o utilizador a manter sessões abertas, sem necessitar de estar a fazer *login* constantemente em sistemas *stateless* (como por exemplo no protocolo Hyper Text Transfer Protocol (HTTP) e HTTPS). Quando o utilizador faz *login* com sucesso, o servidor devolve um *token* de sessão ao cliente que por sua vez é guardado nos *cookies* por um tempo limite. Para cada pedido que o cliente efetuar no mesmo domínio é enviado o *token* e assim provar que a sessão já foi autenticada. Assim, desde que a sessão ainda seja válida, os pedidos serão aceites. Por essa razão, se uma entidade maliciosa aceder ao sistema e obter os *cookies* pode conseguir aceder à conta do utilizador através do *token* de sessão lá guardado.

Existe algumas proteções no caso do roubo dos cookies. Uma das proteções é autenticação baseada no factor localização, em que é verificado pelo sistema que caso utilizador esteja numa localização desconhecida ou fora do normal uso dele, esse *token* é rejeitado. Neste caso, o utilizador deve voltar a autenticar-se.

A autenticação baseada em *nonce* é utilizada nos casos de One-Time Password (OTP), ou seja, uma *password* de utilização única (por exemplo para uma única transação ou para uma sessão de *login*), em que pode conter só números ou números

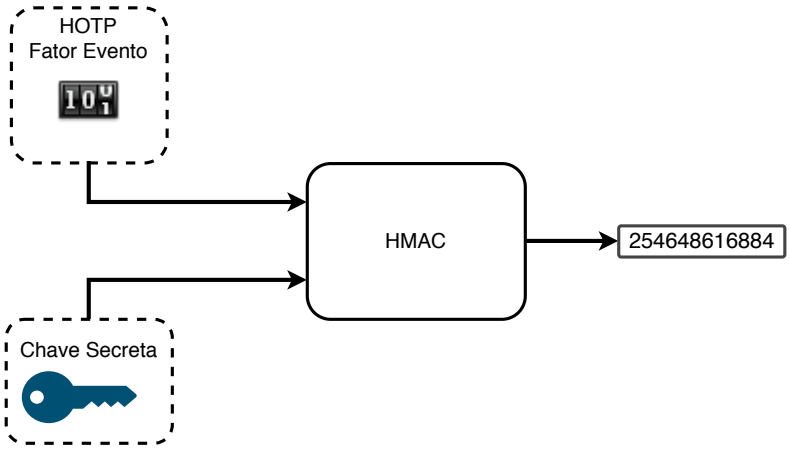

Figura 9: Processo de gerar *password* de utilização única por HOTP.

e letras (Eldefrawy et al., 2011). Essas *passwords* podem ser obtidas por Short Message Service (SMS), por email ou por uma aplicação. As *passwords* de utilização única são geradas através de uma chave secreta e por um factor em movimento (exemplo factor tempo ou através de um factor eventos). Deste modo, uma password OTP tem uma validade temporal reduzida, da ordem dos segundos.

Existem dois tipos de OTP (OneLogin, s.d.):

- Hash-based One-Time Password (HOTP) – Este OTP baseia se na ocorrência de um evento, por exemplo um pedido de autenticação, uma chave secreta é processada por um Hash-based Message Authentication Code (HMAC) para devolver uma *password* de utilização única (ver Figura 9).
- Time-based One-Time Password (TOTP) – Este OTP funciona da mesma forma que Hash-based One-Time Password (HOTP), a diferença é que acrescenta um parâmetro cronológico, exemplo a hora atual. Utilizado de preferência na autenticação, pelo motivo de colocar a *password* (de utilização única) sincronizada entre aplicação e o servidor em causa, desde que a chave secreta na aplicação seja igual à do servidor, a *password* fica sincronizada por utilizarem o mesmo factor cronológico e um segredo (ver Figura 10).

O *nonce* como OTP pode ser utilizado em operações de autenticação por SMS, por email e por aplicativo (LoginRadius, 2019), tais como Google Authenticator para Android.

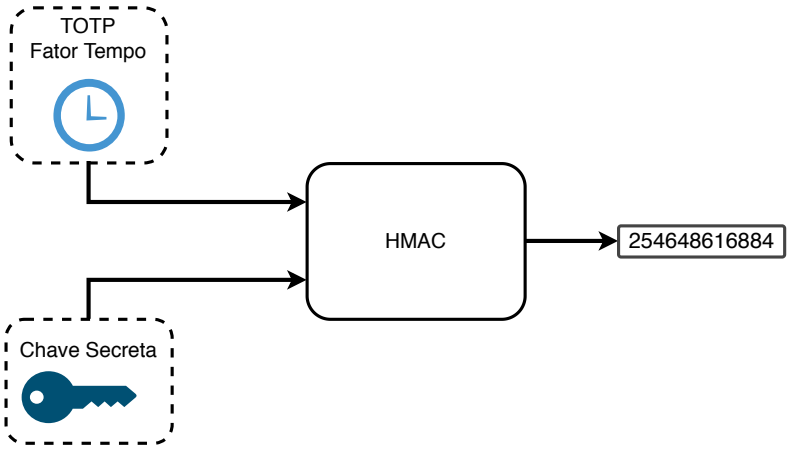

Figura 10: Processo de gerar *password* de utilização única por TOTP.

2.5.1 *Autenticação por SMS*

O meio mais comum para autenticação de dois factores, em que é necessário algo que usamos no dia-a-dia, é um telemóvel e um número de telefone registado numa operadora de telecomunicações. O seu funcionamento é simples, após a autenticação com *username* e *password* é enviado um *nonce* (tipicamente um número de 6 dígitos) por SMS. De seguida o utilizador escreve o *nonce* no sistema no qual está a tentar autenticar-se. Esta forma de autenticação, por SMS, também pode ser usada como autenticação de um só factor, mas não é comum. Além disso, não é o método mais seguro, pois as mensagens SMS não são encriptadas e é possível aceder ilegitimamente ao conteúdo de SMS explorando vulnerabilidades nas redes de telecomunicações móveis (Whittaker, 2018).

2.5.2 *Autenticação por email*

O email é o meio mais usado nos pedidos de recuperação de *password*, mas também pode ser usado para autenticação de dois factores. O método é semelhante ao uso de SMS, a única diferença é que em vez de receber por SMS recebe por email um autenticador único, tipicamente um URL.

2.5.3 *Autenticadores por aplicativo*

O seu funcionamento requer uma aplicação num sistema que esteja sempre acessível, habitualmente um *smartphone*. A aplicação guarda as chaves secretas de cada uma

das plataformas, para assim poderem ser gerados códigos em intervalos de tempo regulares a partir da hora e da chave secreta. Este tipo de sistema é vantajoso pela facilidade de configuração do método Time-based One-Time Password (TOTP) de cada plataforma, em que possibilita a configuração através do QR Code, e ainda possibilitar a eliminação ou criação de uma nova chave secreta quando esse for comprometido. Como exemplos, o Google Authenticator e o Microsoft Authenticator, ambos para dispositivos móveis.

2.5.4 *Autenticadores por hardware*

A autenticação por hardware requer um dispositivo físico. O mais simples é o *hardware nonce*, um dispositivo sem Internet, que mostra *nonces* para um única plataforma. Existe ainda a série dos YubiKey 4 e 5 em que permite colocar vários TOTP ou HOTP de diferentes plataformas no dispositivo, através da aplicação móvel *"Yubico Authenticator"* ou para Windows *"Yubikey Personalization Tool"*.

No caso de *hardware nonce*, o dispositivo permite ao utilizador ter uma proteção adicional através do TOTP. No entanto, trouxe outras complicações, como a bateria, em que obriga o utilizador a ter sempre uma pilha extra por perto, para as situações em que o dispositivo fica sem bateria. Além disso, obriga o utilizador a trazer um dispositivo, nalguns casos de grandes dimensões, consigo. Em alguns casos os dispositivos só permitem o acesso a uma única plataforma. Isto obriga a ter um dispositivo diferente para cada plataforma. Em caso de perda do *hardware nonce* os dados ficam disponíveis a todos que conseguirem ter acesso ao ecrã do dispositivo.

No caso do Yubikey com OTP permite ter mais que uma plataforma configurada e pode ser tanto HOTP como TOTP, a desvantagem deste método no dispositivo é que obriga a ter uma aplicação para ter acesso aos *nonces*.

Em ambos os tipos de dispositivo, é usado o TOTP para gerar *nonce* de x em x de tempo, para que seja criado um *nonce* diferente em todas as vezes, por levar em consideração a data e hora atual, que está constantemente a mudar e leva em consideração uma chave secreta partilhada entre o dispositivo e a plataforma em causa.

Tabela 1: YubiKeys disponíveis e respetivos protocolos suportados.

Nome	Protocolos
YubiKey 5 Series	FIDO2/WebAuthn, U2F, Smart card, OpenPGP, OTP
Security Key Series	FIDO2/WebAuthn, U2F
YubiKey FIPS Series	smart card, OTP, U2F

Tabela 2: YubiKeys disponíveis e preços indicativos.

Nome	Interface	Preço Sem IVA
YubiKey 5 Series	USB-A, USB-C, NFC	45 a 60 euros
Security Key Series	USB-A, NFC	20 a 27 euros
YubiKey FIPS Series	USB-A, USB-C	46 a 69 euros

2.6 YUBICO

O Yubico foi criado em 2007 com a missão de tornar a autenticação simples e disponível para todos. A empresa criou chaves de segurança física, chamadas YubiKey, para que os utilizadores pudessem autenticar através do método de autenticação de dois factores, ou múltiplos factores, ou ainda autenticar sem *password*.

As YubiKeys contêm FIDO U2F e FIDO2 que serão explicados no capítulo 3, tem também suporte para OTP que tem várias aplicações e entre outras funcionalidades (ver Tabelas 1 e 2), que são (Kelly, 2020):

- *Password* Estática – A funcionalidade desta aplicação é adicionar a *password* o tal conhecido sal (*Salt*), tornar as *passwords* mais fortes por adicionar um conjunto de caracteres aleatoriamente às *passwords*. O sal é conhecido mais no armazenamento do lado do servidor, para evitar *passwords* simples e pequenas, de forma a combater ataques por dicionário ou por *Brute-force*. O que esta funcionalidade quer ajudar é adicionar sal às *passwords* do utilizador, antes de enviar a *password* para o servidor, para assim garantir uma *password* mais forte. Para criar o sal é preciso o programa *"YubiKey Personalization Tool"* em que é possível adicionar a funcionalidade sal ao YubiKey (até 32 caracteres), esses caracteres podem ser numéricos, letras ou caracteres especiais. Para usar a funcionalidade sal no campo de *password* da página, coloca-se a YubiKey no dispositivo e clica-se no botão do YubiKey, esse vai simular um teclado e escrever o sal previamente definido, depois de inserir o sal o utilizador pode colocar a sua *password* e clicar *login*. A vantagem desta funcionalidade é que

permite ao utilizador guardar na memória uma *password* simples sem símbolos, pois o sal adiciona símbolos e números à *password*. A desvantagem é entrar em sistemas que têm o teclado com formato diferente do escolhido no YubiKey, o que faz com que símbolos sejam trocados, como por exemplo, no YubiKey com teclado US e o sistema operativo do cliente com o teclado PT;

- OATH – A funcionalidade contém o OTP normal, ou seja, contém a funcionalidade HOTP que leva em consideração um contador e ainda contém a funcionalidade TOTP que é o HOTP só que em vez de ser considerado um contador, leva em consideração o tempo do dispositivo que está a ler o YubiKey, o YubiKey permite até 32 OTPs;
- Yubico OTP – A sua funcionalidade é semelhante à do OTP a diferença é que faz uso de chaves assimétricas para autenticar o OTP gerado, usado um servidor externo para fazer essas verificações;
- Challenge-Response – É um conjunto da funcionalidade Yubico OTP e OATH-HOTP, em que autentica o utilizador através da assinatura de um *nonce* usando o Yubico OTP ou OATH-HOTP em que o *nonce* é considerado como contador.

A YubiKey também tem o PIV (*Personal Identity Verification*) como smart card em que possibilita autenticação com certificados, para ganhar acesso a computadores, a edifícios ou a serviços, desde que o certificado (chave pública) seja registado na plataforma desejada à sua conta de utilizador. O YubiKey PIV suporta os algoritmos Rivest-Shamir-Adleman (RSA) 2048, Elliptic-curve cryptography (ECC) P256 e ECC P384. Também é possível o uso do OpenPGP no YubiKey, essa funcionalidade ativa as operações de assinatura da criptografia RSA (RSA 2048, RSA 3072 e RSA 4096) ou ECC usando uma chave privada armazenada em um cartão inteligente (como YubiKeys), tendo como uso encriptar informações, nomeadamente emails.

3

FAST IDENTITY ONLINE

O FIDO tem como significado em latim como “fiel”, ou seja, que é digno de confiança (Priberam Informática, 2021). O FIDO surgiu em 2012, apoiado pelas empresas como Paypal, Lenovo, Google, Visa e Microsoft, conhecido coletivamente como o FIDO Alliance, em que o seu principal objetivo é a diminuição do uso das *passwords*. Os problemas que a FIDO Alliance menciona é que não existe tanta adoção dos métodos de autenticação através de chaves assimétricas ou outros métodos de igual forma resistentes por estes não serem *“user-friendly”* e o custo de desenvolver esses serviços nas plataformas é alto. Dessa forma, o grupo quis criar técnicas sólidas para serem reconhecidas como “standard” pelas organizações e poder assim certificar as plataformas que usam as técnicas, garantido assim que essas estão corretamente implementadas (FIDO Alliance, s.d.; FidoAlliance, s.d.).

O primeiro lançamento do grupo FIDO Alliance foi em dezembro de 2014, em que foi publicado o FIDO U2F (Universal 2^{nd} Factor) e o FIDO UAF (Universal Authentication Framework), chamado atualmente Fast IDentity Online versão 1 (FIDO1). O FIDO UAF consiste na implementação de autenticação sem *password* através do uso da impressão digital e o FIDO U2F desenvolvido pela Google e a Yubico como um método de autenticação de dois factores. O FIDO U2F inclui o seu próprio protocolo do lado do cliente, que é o Client To Authenticator Protocol (CTAP), que permite autenticação através de Universal Serial Bus (USB), NFC ou Bluetooth.

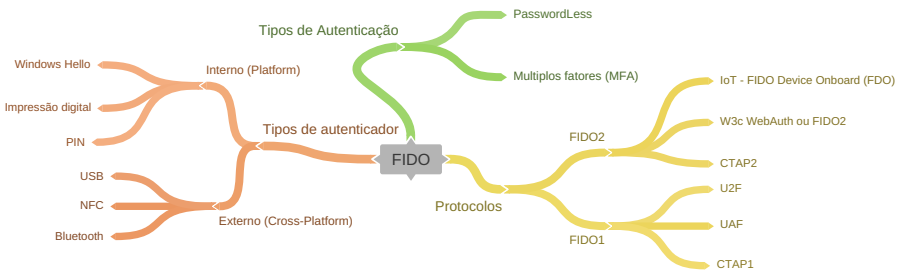

Figura 11: Protocolos, tipos de autenticação e tipos de autenticador do FIDO.

O FIDO1 estava dividido em dois protocolos FIDO Universal Authentication Framework (UAF) apoiado pela Paypal e FIDO U2F apoiado pela Google, mas ambos com a vantagem de estarem embutidos nos dispositivos. No caso do UAF, embutida a autenticação biométrica nos dispositivos móveis e no caso U2F embutido nativamente no *browser* Chrome. Desta forma o utilizador não necessita de instalar ou habilitar esses protocolos, pois estes ficam disponíveis quando o cliente precisar (*user-friendly*).

O segundo lançamento foi em fevereiro de 2016, que é a continuação do FIDO U2F e o seu protocolo CTAP (Client To Authenticator Protocol), em que passa a existir a versão 2 do protocolo CTAP, chamado Client To Authenticator Protocol versão 2 (CTAP2). O CTAP2 agora permite autenticação através do dispositivo local como FIDO UAF, só que neste caso também permite, além do uso de impressão digital, como exemplo a *password* do sistema, ou Windows Hello no caso do Windows. O FIDO2, comparado com FIDO1, agora tem como possibilidade não só implementar autenticação de dois factores, mas também como autenticação sem *password* ou autenticação única (*PasswordLess*). Para além disso o FIDO2 tornou-se numa norma de W3C (identificada pelo o W3C como *WebAuthn*), também juntou o FIDO U2F e o FIDO UAF em um só e trouxe o novo CTAP2 que possibilita comunicar com autenticadores do tipo interno, USB, NFC e Bluetooth. O FIDO2 foi desenvolvido em conjunto com W3C e é compatível com as versões anteriores, como FIDO U2F e FIDO UAF.

O FIDO2 atualmente (29/06/2020) é suportado em quase todos os *browsers*, com exceção de alguns *browsers* em dispositivo móveis que ainda estão a ser desenvolvidos. Para mais detalhe pode-se observar na Figura 12 os *browsers* que disponibilizam FIDO2 (chamado na figura *WebAuthn* Application Programming Interface (API) e CTAP2) e FIDO U2F.

Nos subtópicos seguintes será abordado o funcionamento do FIDO2, em que será explicado como é efetuado a comunicação e os parâmetros que são trocados entre o cliente e o servidor. Antes de explicar o seu funcionamento é de relembrar que o FIDO foi classificado nos capítulos anteriores como autenticação baseada em chaves públicas, o que quer dizer que o FIDO é um protocolo que faz uso das técnicas de criptografia assimétrica. Esta consiste em criar um par de chaves para cada servidor, nesse par de chaves existe a chave privada que só deve existir no lado do cliente (nunca partilhado), e a chave pública que é partilhada com o servidor na altura do registo. Relembrando também que autenticação FIDO2 pode ser tanto "autenticação sem password" como também "autenticação de múltiplos factores". Neste caso serão explicados ambos, mas na "autenticação de múltiplos factores"

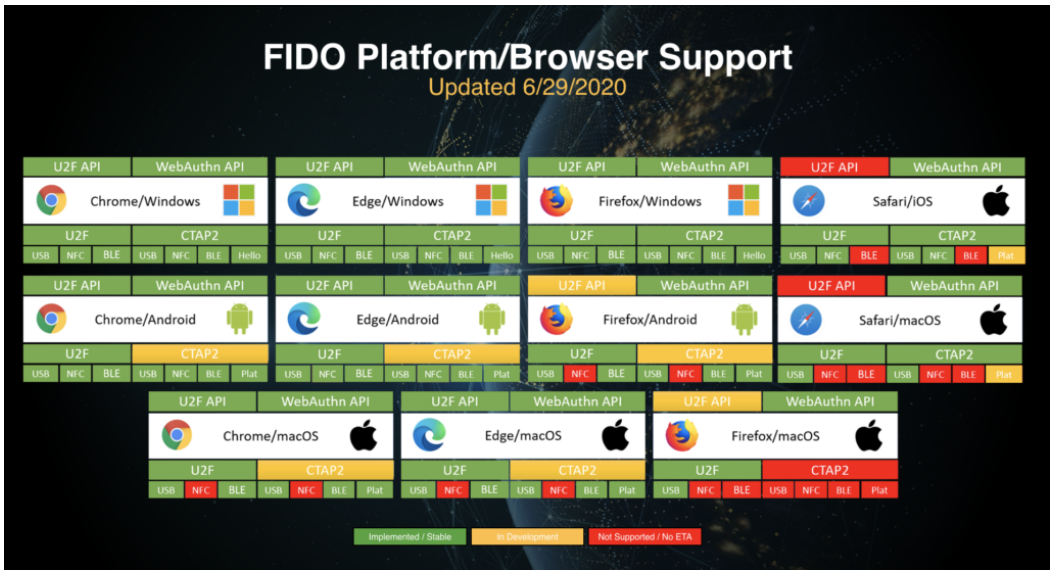

Figura 12: *browsers* que suportam o FIDO2 (WebAuthn e CTAP2) e o FIDO U2F (FidoAlliance, s.d.).

será considerado como uma autenticação de dois factores em que um dos factores é autenticação com *password*, e será também falado sobre registo das chaves públicas usando o FIDO2.

3.1 AUTENTICAÇÃO E REGISTO USANDO O FIDO2

Para poder efetuar uma autenticação com o FIDO2, é necessário que o servidor tenha conhecimento da chave pública do dispositivo que o cliente pretenda usar para autenticar. O FIDO2 classifica as chaves públicas em duas categorias, a categoria *"platform"* em que faz uso das técnicas de criptografia do sistema (Windows Hello ou impressão de dedo no caso dispositivos móveis) e a categoria *"cross-platform"* que faz uso das técnicas de criptografia de um dispositivo externo que pode ser por USB, NFC ou Bluetooth.

Como o FIDO2 comunica por pedidos HTTP, este segue a regra de pedido e resposta. Então, supondo que o utilizador já está autenticado e está na página para registo de chaves FIDO2, o utilizador terá de escolher que categoria de chave pretende utilizar, ou seja, escolher autenticador interno ou externo. Ao selecionar o tipo de autenticador é submetido a opção, e enviado um pedido a pedir o desafio ao servidor avisando que pretende utilizar categoria de autenticador selecionada (*"platform"* ou *"cross-platform"*). De seguida o servidor cria o desafio que é enviado para o cliente, a categoria de autenticador escolhida e chaves públicas já registadas,

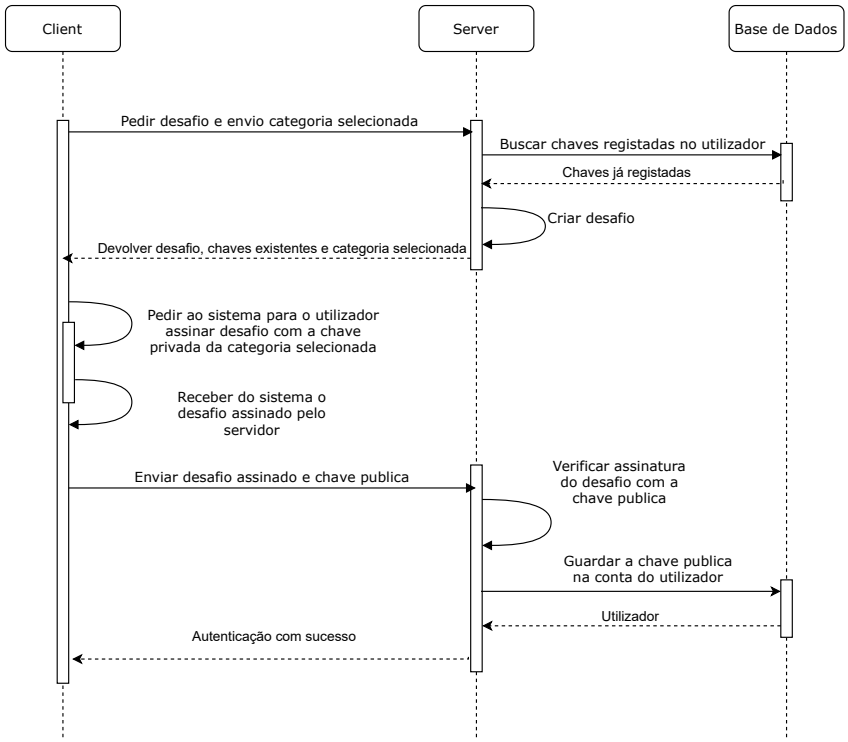

Figura 13: Processo de registar uma chave pública usando FIDO2.

para assim impedir registar com uma já existente. Depois, o cliente chama o sistema de FIDO2 do cliente para assinar o desafio, no caso de *"platform"* o uso do Windows Hello (no caso do Windows), ou a impressão digital (no caso do Android). No caso de ser *"cross-platform"* o uso do Yubikey ou outro tipo de dispositivo externo em que faz uso Bluetooth, ou USB ou NFC. Depois de assinar o desafio com o autenticador selecionado, é enviado o desafio assinado juntamente com a chave pública. O servidor ao receber esta informação verifica se o desafio está corretamente assinado com a chave pública enviada e depois guarda a chave pública na base de dados registado na conta do utilizador e envia uma mensagem de sucesso, como se pode se visualizar na Figura 13.

Nos seguintes casos vamos considerar que o utilizador não está autenticado e que o *browser* (Cliente) já pediu a página de *login*. Desta forma a explicação será a mesma tanto num *browser* como numa aplicação móvel, pois no caso das aplicações móveis essas já contém as páginas.

No caso da autenticação sem *password* (*PasswordLess*), o utilizador insere o *username* e enviar para o servidor. O servidor verifica se existe e depois cria um desafio para o cliente que é um conjunto letras e números criados de forma aleatória. O cliente terá que assinar o desafio usando a sua chave privada que está no sistema ou num dispositivo externo (exemplo Yubikey) e envia-o ao servidor. De seguida, o

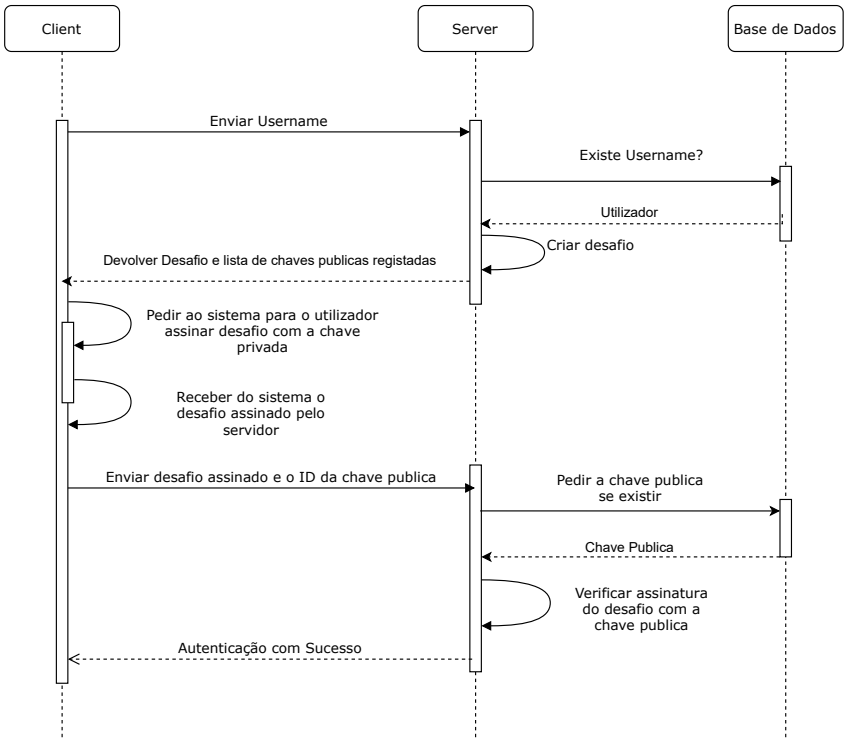

Figura 14: Processo de autenticação sem *password* com o FIDO2.

servidor verifica a assinatura usando a chave pública que foi guardada com a conta do utilizador. Depois de verificado a assinatura, o servidor devolve a resposta de autenticação com sucesso e o *token* de sessão ou de autenticação, este processo pode se visualizar na Figura 14.

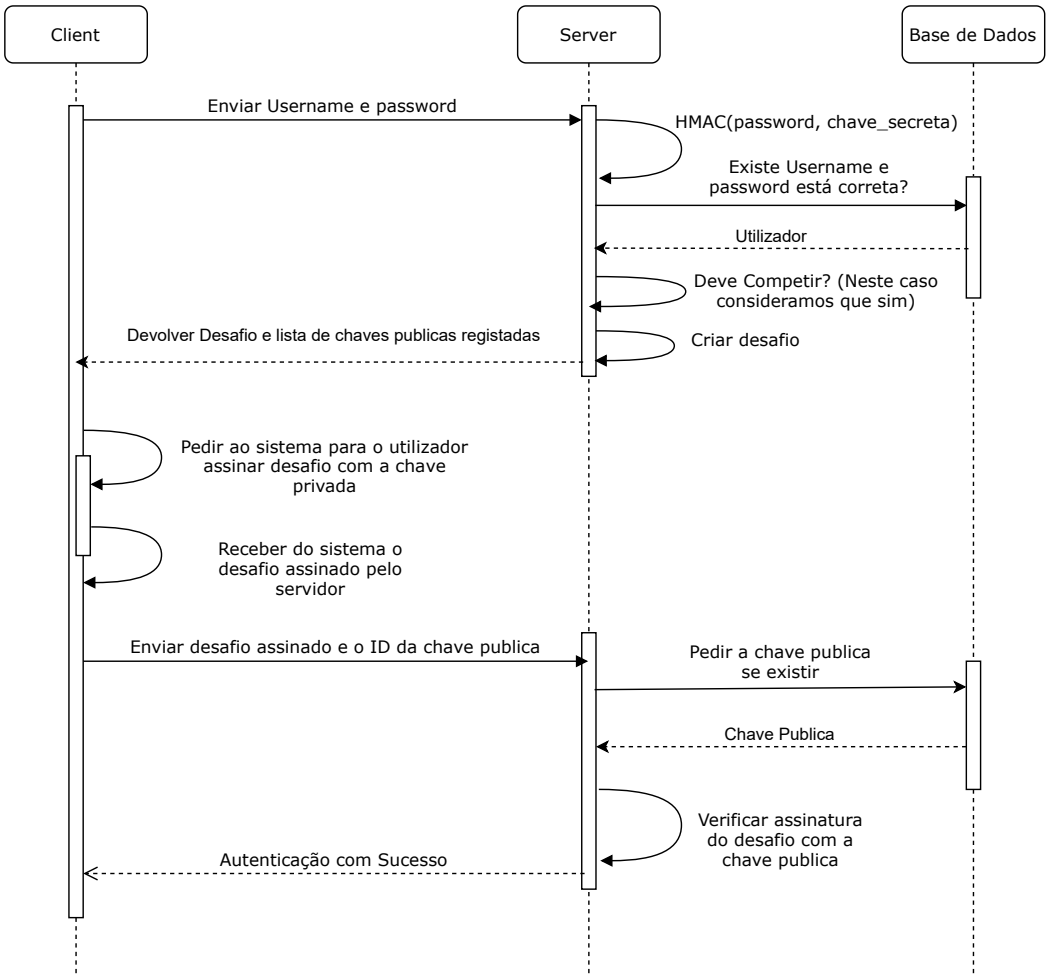

Figura 15: Processo de autenticação de 2 factores com o FIDO2.

No caso da autenticação de dois factores (supondo que um dos factores é *password*), como em grande maioria das aplicações o utilizador insere o *username* e a *password* e envia para o servidor. Por sua vez o servidor verifica se a *password* está correta, depois de verificar com sucesso o servidor vai avaliar se o cliente terá que fazer o segundo factor de autenticação. Neste caso consideramos que sim e então o servidor cria um desafio para o cliente que é um conjunto letras e números criados de forma aleatória em que o cliente tem que assinar o desafio usando a sua chave privada que está no sistema ou num dispositivo externo (exemplo Yubikey) e é enviado ao servidor. O servidor verifica a assinatura usando a chave pública que foi guardada com a conta do utilizador, depois de verificado o servidor devolve a resposta com autenticação com sucesso e o *token* de sessão ou de autenticação. Este processo está representado na Figura 15.

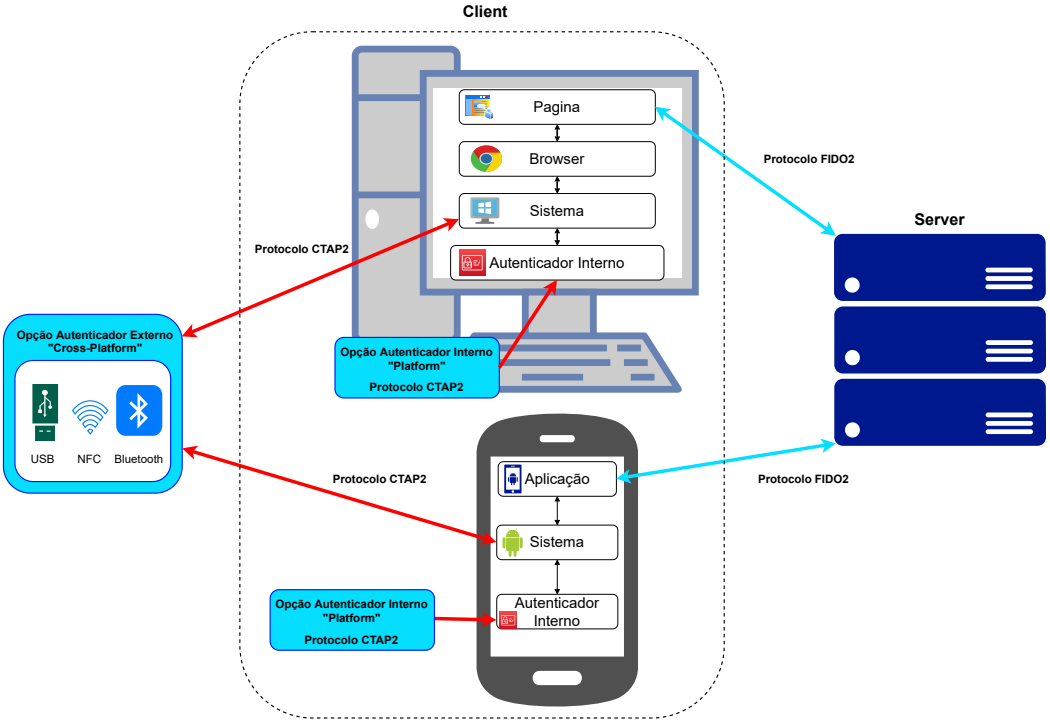

Figura 16: Funcionamento geral do FIDO2.

3.2 PROTOCOLOS DO FIDO2

No sub-capítulo anterior foi falado de forma geral como o utilizador autentica usando o FIDO2. Neste sub-capítulo será detalhado o seu funcionamento, incluindo os parâmetros que são passados no protocolo FIDO2.

Primeiro de tudo existem dois protocolos, o protocolo CTAP2 em que o sistema operativo usa para assinar e recolher chaves publicas dos autenticadores, autenticador externo (*"cross-platform"*) e interno (*"platform"*), e o protocolo FIDO2 em que é usado na comunicação entre aplicação e o servidor, está representado na Figura 16 onde cada um dos protocolos é usado.

Agora será abordado de forma mais detalhada como o protocolo FIDO2 funciona. O protocolo funciona em 4 passos, primeiro pedir ao servidor o desafio, segundo receber o desafio, terceiro enviar desafio assinado e quarto receber mensagem de sucesso (Figura 17). Tanto no registo de uma chave publica como na autenticação ambos tem esses 4 passos, a diferença entre eles é os dados adicionais que vão em conjunto com o desafio. No caso do registo (Figura 13) é trocado com o servidor o desafio, as informações para criar uma chave publica, para assinar o desafio e a troca também da chave publica. No caso da autenticação (Figura 14) é trocado o

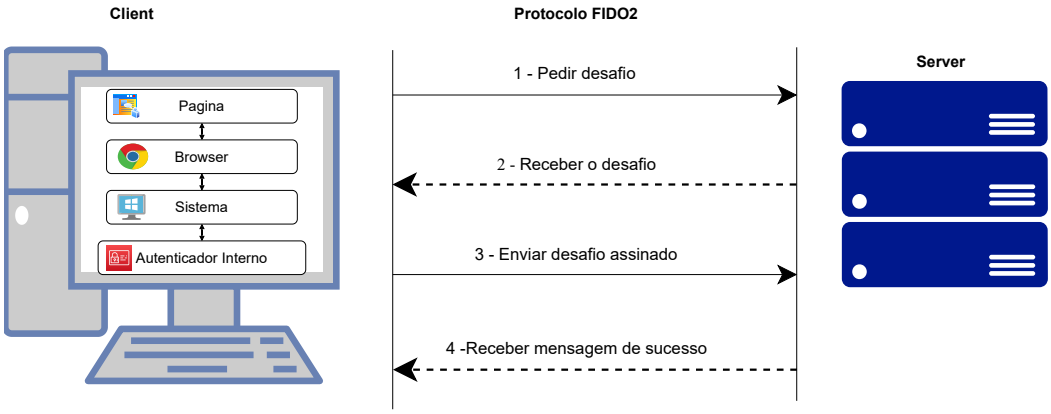

Figura 17: Protocolo FIDO2.

desafio e os IDs das chaves publicas que utilizador tem registado, para evitar fazer pedidos com chaves não registadas, estes são os principais mas há mais.

No caso de registar uma chave pública seguem-se os seguintes passos e parâmetros:

1. Pedir ao servidor o desafio

   O cliente terá que enviar uma forma de identificação do utilizador, por exemplo o *username* ou os tokens de autenticação. Também deve escolher como pretende usar o autenticador, se pretende usar *platform* que faz uso do autenticador interno do sistema ou *cross-platform* que faz uso do dispositivo externo como USB, NFC, Bluetooth. Esta é a opção que pode ficar disponível ao utilizador para escolher, o resto das opções da autenticação do cliente devem ser escolhidas pelo administrador ou o sistema. De seguida está o nome do objeto *authenticatorSelection* onde deve conter as opções que cliente pretende usar:

   - *authenticatorSelection* – informações que o cliente tenciona usar na autenticação, (do tipo *AuthenticatorSelectionCriteria*) contém:
     - *authenticatorAttachment* – o tipo de autenticação selecionado, possíveis *platform* que é o uso autenticador interno do sistema ou *cross-platform* que é uso dispositivo externo como USB, NFC, Bluetooth;
     - *requireResidentKey* – é um *boolean*, se for verdadeiro obriga que a chave seja guardada no dispositivo, por exemplo no yubikey. Caso seja falso, o autenticador guarda numa forma diferente desde que não seja na memória do autenticador, em que pode ser a guardar no lado do servidor, isto será visto no capítulo 3.3;

- *userVerification* – decide se o utilizador necessita de verificar a ação antes de executar, com um PIN, um toque num botão ou colocar impressão digital.

2. Receber o desafio

   O servidor verifica o pedido do cliente e verifica se no *authenticatorSelection* não existe alguma opção que não esteja dentro das restrições definidas pelo administrador. Por exemplo o servidor ou administrador pode não autorizar o uso de *platform* que faz uso do autenticador interno do sistema. Ao validar todos os dados, o servidor cria o desafio e envia junto todas as informações necessárias para o cliente criar a chave pública e assinar o desafio. De seguida está o objeto *PublicKeyCredentialCreationOptions* onde deve conter as opções que o servidor envia para o cliente usar:

   Tipo *PublicKeyCredentialCreationOptions*

   - *rp* (obrigatório) – é a identificação do servidor, (tipo *PublicKeyCredentialRpEntity*) contém:
     - *name* (obrigatório) – serve para mostrar ao utilizador o nome do serviço ou servidor no qual está autenticar, desta forma o cliente poder confirmar, exemplo "Serviço de Autenticação IPLeiria" ou só "IPLeiria";
     - *id* (obrigatório) – é a forma de identificação do servidor ao qual o cliente está a tentar autenticar, por exemplo *estg.ipleiria.pt* ou *ipleiria.pt*, isto é o servidor ID.
   - *user* (obrigatorio) – contem a informações do utilizador, (tipo *PublicKeyCredentialUserEntity*) contém:
     - *id* (obrigatório) – em que é a identificação do utilizador no servidor;
     - *displayName* (obrigatório) – nome do utilizador que é mostrado na janela de confirmação.
   - *challenge* (obrigatório) – é um conjunto de caracteres gerados pelo servidor de forma aleatória para ser assinado;
   - *pubKeyCredParams* (obrigatório) – contém a lista dos algoritmos aceites pelo servidor para gerar as chaves assimétricas, (tipo lista de *PublicKeyCredentialParameters*) contém:
     - *type* (obrigatório) – é o tipo de algoritmo a ser utilizado, de momento só existe o *public-key* (chave assimétrica);

  - *alg* (obrigatório) – identificação do algoritmo registado no Registo de Algoritmos IANA COSE [1].

- *timeout* – tempo em milissegundos que utilizador tem para poder se autenticar a nova chave pública que vai enviar;
- *excludeCredentials* – lista de IDs das chaves públicas já registadas para evitar o utilizador fazer pedidos de registo com chaves públicas já registadas, (lista de *PublicKeyCredentialDescriptor*) contém:
  - *id* (obrigatorio) – identificação da chave pública registada;
  - *type* (obrigatório) — é o tipo de algoritmo a ser utilizado, de momento só existe o *public-key* (chave assimétrica);
  - *transports* – é onde informa por onde a chave pública foi registada, pode ser *nfc*, *usb*, *ble* (Bluetooth Low Energy), ou *internal*.
- *authenticatorSelection* – informações que o cliente tenciona usar na autenticação, (tipo *AuthenticatorSelectionCriteria*) contém:
  - *authenticatorAttachment* – o tipo de autenticação selecionado, possíveis *platform* que é o uso autenticador interno do sistema, ou *cross-platform* que é uso dispositivo externo como USB, NFC, Bluetooth;
  - *requireResidentKey* – é um *boolean*, se for verdadeiro obriga que a chave seja guardada no dispositivo, por exemplo no yubikey, caso seja falso o autenticador guarda numa forma diferente desde que não seja na memória do autenticador, em que pode ser a guardar no lado do servidor, isto será visto no capítulo 3.3;
  - *userVerification* – decide se o utilizador necessita de verificar a ação antes de executar, com um PIN, um toque num botão ou colocar impressão digital.
- *attestation* – é a forma como é dado a assinatura, de forma anónima ou direta.
- *extensions* – este membro contém parâmetros adicionais que solicitam processamento adicional pelo cliente e autenticador. Por exemplo, o cliente pode solicitar que apenas autenticadores com determinados recursos sejam usados para criar a credencial ou que informações específicas sejam retornadas.

1 https://www.iana.org/assignments/cose/cose.xhtml

3. Enviar desafio assinado

   O cliente recebe os parâmetros para criar a chave pública, e envia os parâmetros para o autenticador selecionado, em que este executa o serviço de autenticação em que pede ao utilizador para colocar a *password* no caso ser *platform* e no caso *cross-platform* inserir o dispositivo. Neste passo é criado a chave privada e a chave pública que é usada para assinar o desafio, que depois é enviado ao servidor a assinatura, o desafio, o ID da chave pública, a chave pública e as informações da chave pública como o algoritmo usado. De seguida está o objeto *PublicKeyCredential* onde deve conter as opções que o cliente envia para o servidor usar:

   Tipo *PublicKeyCredential*

   - *id* – é o ID da chave que está a ser utilizada, mas em base 64 para URL;
   - *rawid* – é o ID da chave que está a ser utilizada, mas em *ArrayBuffer*;
   - *type* – é o tipo de algoritmo a ser utilizado, de momento só existe o *public-key* (chave assimétrica);
   - *getClientExtensionResults()* – contém os resultados do processamento dos parâmetros adicionais postos no *extensions* no pedido anterior;
   - *response* – devolve os valores em resposta ao pedido feito, (tipo *AuthenticatorAttestationResponse*) contém:
     - *attestationObject* – Contém os seguintes valores:
       - *authData* – os dados do autenticador em uma matriz de bytes que contém metadados sobre o evento de registo (contador) e bem como a chave pública, e o identificador único do autenticador;
       - *fmt* – é o formato da maneira pela qual a assinatura (*attStmt*) está representada, os autenticadores podem fornecer a assinatura de várias maneiras, este campo indica como o servidor deve analisar e validar os dados (Consortium, 2021b).
       - *attStmt* – contém o desafio assinado pela chave privada.
     - *clientDataJSON* – Contém os seguintes valores:
       - *challenge* – é o mesmo valor de desafio enviado pelo servidor;
       - *origin* – origem do cliente que fez o pedido, o servidor deve verifica se está correto e se permite;

        * *type* – é o valor do operador que está a ser executado, neste caso *webauthn.create*, que significa adicionar ou criar uma chave pública.

4. Receber mensagem de sucesso

   Nesta fase envia-se uma mensagem de sucesso de registo da chave pública ao utilizador.

No caso de autenticar com uma chave pública seguem se os seguintes passos e parâmetros:

1. Pedir ao servidor o desafio

   O cliente terá que enviar uma forma de identificação do utilizador, por exemplo o *username* ou os tokens de temporária autenticação, e também pode enviar opções adicionais do FIDO2 para o servidor. De seguida está o nome do objeto (*authenticatorSelection*) onde deve conter as opções adicionais que o cliente pode usar:

   - *authenticatorSelection* – informações que o cliente tenciona usar na autenticação, (tipo *AuthenticatorSelectionCriteria*) contém:
     - *userVerification* – decide se o utilizador necessita de verificar a ação antes de executar, com um PIN, um toque num botão ou colocar impressão digital.

2. Receber o desafio

   O servidor verifica o pedido do cliente e verifica se no *authenticatorSelection* não existe nenhuma opção que não esteja dentro das restrições. Por exemplo, o servidor ou o administrador do sistema pode não levar em consideração as opções selecionadas pelo cliente. Ao validar todos os dados o servidor cria o desafio e envia junto com todas as informações necessárias para o cliente criar a chave pública. De seguida está o objeto *PublicKeyCredentialCreationOptions* onde deve conter as opções que o servidor envia para o cliente usar:

   Tipo *PublicKeyCredentialCreationOptions*

   - *challenge* (obrigatório) – é um conjunto de caracteres gerados pelo servidor de forma aleatória para ser assinado;
   - *rpId* – é a forma de identificação do servidor ao qual o cliente está autenticado, exemplo *estg.ipleiria.pt* ou *ipleiria.pt*, que é o servidor ID.

- *userVerification* – decide se o utilizador necessita de verificar a ação antes de executar, com um PIN, um toque num botão ou colocar impressão digital.
- *timeout* – tempo em milissegundos que utilizador tem para poder se autenticar;
- *allowCredentials* – lista de IDs das chaves públicas já registadas, para evitar o utilizador fazer pedidos de *login* com chave públicas não registadas no utilizador, (tipo lista de *PublicKeyCredentialDescriptor*) contém:
  - *id* (obrigatório) – identificação da chave pública registada;
  - *type* (obrigatório) – é o tipo de algoritmo a ser utilizado, de momento só existe o *public-key*, do tipo chave assimétrica;
  - *transports* – é onde informa por onde a chave pública foi registada, pode ser *nfc*, *usb*, *ble* (Bluetooth), ou *internal*.
- *extensions* – este membro contém parâmetros adicionais que solicitam processamento adicional pelo cliente e autenticador. Por exemplo, o chamador pode solicitar que apenas autenticadores com determinados recursos sejam usados para criar a credencial ou que informações específicas sejam retornadas.

3. Enviar desafio assinado

   O cliente recebe as informações, envia os parâmetros para autenticador selecionado e este executa o serviço de autenticação em que pede ao utilizador para colocar a *password* no caso ser *platform* e no caso *cross-platform* inserir o dispositivo. Neste passo é recuperada a chave privada da memória do autenticador e usada para assinar o desafio, que depois é enviado ao servidor o desafio assinado e as informações adicionais relativo à assinatura. De seguida está o objeto *PublicKeyCredential* onde deve conter as opções que o cliente envia para o servidor usar:

   Tipo *PublicKeyCredential*

   - *id* – é o ID da chave que está a ser utilizada, mas em base 64 para URL;
   - *rawid* – é o ID da chave que está a ser utilizada, mas em *ArrayBuffer*;
   - *type* – é o tipo de algoritmo a ser utilizado, de momento só existe o *public-key*, o tipo chave assimétrica;
   - *getClientExtensionResults()* – contém os resultados do processamento dos parâmetros adicionais postos no *extensions* no pedido anterior;

- *response* – devolve os valores em resposta ao pedido feito, (tipo *AuthenticatorAttestationResponse*) contém:
  - **"authenticatorData"** – os dados do autenticador numa matriz de bytes que contém metadados sobre o evento de registo (contador) e bem como o endereço do servidor;
  - *clientDataJSON* – contém os seguintes valores:
    - *challenge* – é o mesmo valor de desafio enviado pelo servidor;
    - *origin* – origem do cliente que fez o pedido, o servidor deve verificar se está correto e se permite;
    - *type* – é o valor do operador que está a ser executado, neste caso *webauthn.get*, que significa que está a pedir para autenticar.
  - *signature* – a assinatura criada do desafio através da chave privada.
  - *userHandle* – este campo é fornecido opcionalmente pelo autenticador e representa o *user.id* que foi fornecido durante o pedido desafio.

4. Receber mensagem de sucesso

   Nesta fase envia-se uma mensagem de sucesso de autenticação do utilizador e os *token* de autenticação caso o servidor faça uso desse método de autenticação para manter o cliente autenticado.

Agora que já descrito o protocolo FIDO2, será então explicado o protocolo CTAP2. Este protocolo consiste em tratar a comunicação entre o sistema e o dispositivo de autenticação (por exemplo Yubikey ou Windows hello). Para quem está aplicar o método de autenticação FIDO2 na sua aplicação, para o efeito, só necessita de saber como funciona este protocolo CTAP2 no modo geral, pois no caso do sistema operativo e do navegador (exemplo Chrome, Firefox e etc) estes já têm o CTAP2 aplicado. Em alguns casos ainda não tem o CTAP2 instalado na sua raiz, em que será necessário saber de forma detalhada, para assim poder aplicar e usar para comunicar com o dispositivo de autenticação (para pedir chaves públicas e para pedir os desafios assinados).

Como o FIDO2 tem compatibilidade com as versões anteriores (FIDO1), o protocolo CTAP2 poderá ser usado como compatibilidade com autenticadores que fazem uso Client To Authenticator Protocol versão 1 (CTAP1), mas não é obrigatório usar. Uma das diferenças entre eles é que o CTAP1 usa *raw message* enquanto o CTAP2 faz uso da forma de codificação tipo *CBOR* (Fido Alliance, 2021c).

O protocoloCTAP2 está dividido em três partes específicas:

- **Autenticador API** – Que é o operador de autenticação que simula uma API, em que aceita parâmetros como entrada e retorna parâmetros ou um código de erro, o autenticador API é o dispositivo de autenticação, por exemplo Yubikey;
- **Codificação de Mensagem** – É uma mensagem enviar ou recebida do autenticador API, em que é codificada em *raw message* ou em *CBOR Message*;
- **Ligação específica para transporte** – É a ligação ou canal de transmissão entre a aplicação e o autenticador API, em que os pedidos e respostas são transportados para os autenticadores por meio de transportes específicos (por exemplo, USB, NFC, Bluetooth). Para cada tecnologia de transporte, as ligações de mensagem são especificadas para este protocolo.

No protocolo FIDO2 foram vistos os parâmetros que o servidor envia e recebe, esses parâmetros são de certa forma iguais aos que o CTAP2 usa, pois os dados que são passados pelo protocolo FIDO2 são os mesmo que passam pelo CTAP2. A diferença é que o tratamento desses dados é de forma diferente entre os dois: o FIDO2 faz uso de JSON e o CTAP2 faz uso de *raw message* ou *CBOR Message*. Também são disponibilizados outro tipos de informações, que podem ser obtidas executando os seguintes comandos:

- *authenticatorMakeCredential* (Código: *0x01*) – Este comando serve para pedir ao autenticador para gerar uma nova credencial (chave privada e chave pública);
- *authenticatorGetAssertion* (Código: *0x02*) – Neste comando tem como uso autenticar ou comprovar de forma criptográfica a autenticidade do utilizador, usando a credencial previamente registada ou gerada;
- *authenticatorGetNextAssertion* (Código: *0x01*) – O comando é usado com o anterior, porque o comando anterior (*authenticatorGetAssertion*) só envia de volta uma chave publica e uma assinatura de cada vez. Este método serve para receber as outras assinaturas e credenciais, só quando o comando anterior avisa que existe mais que uma credencial registada para o mesmo servidor ID do mesmo autenticador;
- *authenticatorGetInfo* (Codigo: *0x04*) – Este comando disponibliza informações como a versão de formatos que estão disponíveis (“FIDO_2_1”, “FIDO_2_0” e “U2F_V2”), o ID do autenticador (*AAGUID* – Authenticator Attestation Globally Unique ID), algoritmos disponíveis, tipo de transporte disponível

(USB, NFC e Bluetooth) e entre outros. Mostra informações do autenticador e o que ele é capaz de fazer;

- *authenticatorClientPIN* (Código: *0x06*) – Este comando serve para colocar um PIN no autenticador, para assim poder ser verificado cada operação através desse mesmo PIN, que é definido pelo utilizador;
- *authenticatorReset* (Codigo: *0x07*) – Este comando é usado pelo utilizador, para redefinir o autenticador de volta aos valores pré-definidos de fábrica, invalidando todas as credenciais criadas por ele e reiniciar o PIN;
- *authenticatorBioEnrollment* (Código: *0x09*) – Este comando é usado para provisionar, enumerar e excluir impressões digitais registadas no autenticador, que tem o mesmo propósito que o PIN;
- *authenticatorCredentialManagement* (Código: *0x0A*) – Comando usado para descobrir as credenciais criadas por este autenticador, usando a credencial ID quando disponível ou o servidor ID (quando a credencial ID não está disponível), isto funciona se as credenciais estiverem registadas no autenticador (por exemplo Yubikey). Se for uma credencial guardada do lado do servidor, essa não é possível descobrir por este método, será explicado mais à frente no capítulo 3.3;
- *authenticatorSelection* (Código: *0x0B*) – Este comando é para o utilizador escolher qual autenticador quer usar (para autenticar ou registar), através de um *click* ou a impressão digital no autenticador. Quando o utilizador fizer isso, esse autenticador fica selecionado para a operação que deseja fazer, como por exemplo autenticar;
- *authenticatorLargeBlobs* (Código: *0x0C*) – Este comando é usado para guardar informação adicional e sensível em conjunto com a credencial, em que é permitido até 1024 octetos (*Bytes*);
- *authenticatorConfig* (Codigo: *0x0D*) – Serve para definir as configurações do autenticador, como exemplo definir o mínimo de caracteres do PIN ou definir se deve pedir ao utilizador para verificar em cada operação, através de um clique no autenticador;

Se queremos gerar uma credencial usando o protocolo CTAP2, primeiro terá de se executar o comando *authenticatorGetInfo*, para assim poder verificar que tipo de formatos de comunicação estão disponíveis. Depois de selecionado o formato de comunicação compatível entre os dois (autenticador e a plataforma), é iniciado então o comando *authenticatorMakeCredential* em que este retorna a chave pública,

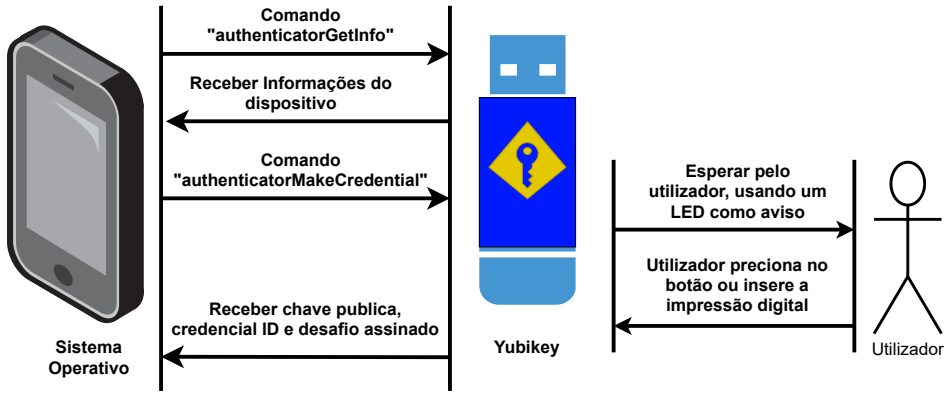

Figura 18: Protocolo CTAP2 para gerar nova credencial usado Yubikey.

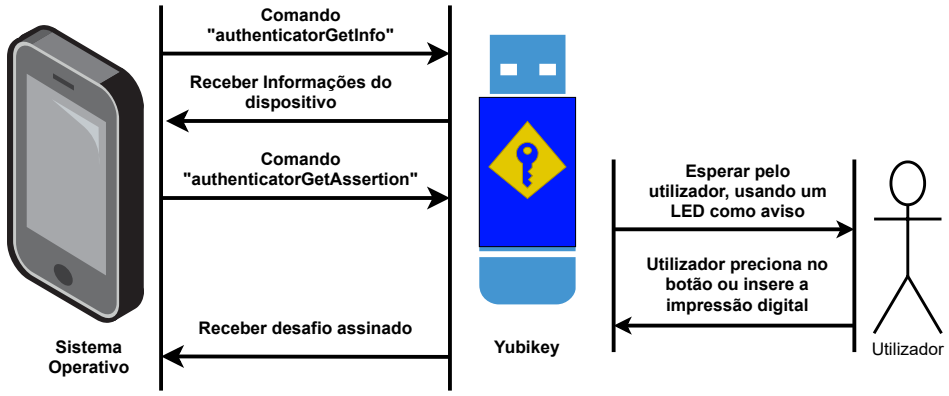

Figura 19: Protocolo CTAP2 para autenticar usado Yubikey.

credencial ID e o desafio assinado. Seguidamente, o utilizador autentica a operação através de um *click* ou com a impressão digital no autenticador. Desta forma, ficaram criadas as credenciais (chave privada e chave pública) para a plataforma selecionada (ver Figura 18).

No caso da autenticação acontece o mesmo, primeiro é feita a verificação de compatibilidade de formatos através do comando *authenticatorGetInfo* e de seguida é executado o comando *authenticatorGetAssertion* que devolve o desafio assinado. Em caso de mais que uma credencial registada no mesmo servidor ID, será necessário executar o comando *authenticatorGetNextAssertion* até que este deixe de devolver desafios assinados com diferentes credenciais, ou seja, é só devolvido um desafio assinado de cada vez. Isto pode acontecer quando é registado uma nova credencial sem eliminar a anterior para o mesmo servidor ID (ver Figura 19).

3.3 ASSINATURA E CHAVES FIDO2

Antes de começar a explicar o funcionamento das chaves FIDO2 e a assinatura, é necessário saber que existem dois tipos de chaves:

- **Chave residente (*"Resident Key"*)** – Este tipo de chave é guardado no próprio autenticador, ou seja, a chave privada é guardada, por exemplo, na memória do Yubikey identificada por o servidor ID ou por ID da credencial gerada. Quando é pedido para assinar um desafio o Yubikey vai à lista de chaves privadas e identificada por aquela que tem o servidor ID ou tem o ID da credencial;
- **Chave do lado do servidor (*"Server-Side key"*)** – Neste tipo de chaves o funcionamento é diferente, pois a chave privada não é guardada no autenticador, é sim guardada no servidor, em que essa está encriptada pela chave mestre do autenticador, para assim só esse autenticador poder desencriptar a chave privada e ter acesso à mesma.

As chaves residentes são mais seguras porque desta forma as chaves privadas nunca saem do autenticador. O problema com este tipo de processo é que limita o utilizador ao número de chaves que pode guardar e assim limitar o número de plataformas ou aplicações que pode usar para autenticar. Por esta razão passaram a armazenar as chaves do lado do servidor (*"Server-Side key"*). Assim, pode-se ter um número ilimitado de chaves geradas pelo autenticador, mas com isto volta o problema como nas *passwords* – em caso de roubo da base de dados do lado do servidor, estas chaves privadas ficariam expostas. Então para resolver este problema, os autenticadores usam uma chave mestre única (que nunca sai do autenticador) para encriptar a chave privada que pertence a essa plataforma e assim sem a chave mestre para desencriptar tornar-se inútil a informação guardada na base de dados.

Para o autenticador criar chaves tem de seguir algumas regras, que são (Fido Alliance, 2019e):

- Utilizar o tipo de geração de chaves que o FIDO2 permite. Até ao momento só existe o tipo *public-key*. Ou seja, a criação de chaves assimétricas em que recomendam seguir o Pseudorandom Number Generator (PRNG) na criação e usar uma boa entropia;
- Cada chave gerada tem de ser identificada para cada servidor ID, para assim não permitir usar uma chave que não pertence ao servidor ID identificado;

- Se usar o tipo de chave *Server-Side key*, a chave privada antes de ser enviada para o servidor tem de ser guardada num modo seguro (encriptada) em que só o autenticador tem acesso;
- Não deve ser possível identificar a quem pertence a chave *"Server-Side key"*, ou seja, quando a chave privada encriptada na base de dados do servidor for consultada, não deve ser possível saber se pertence a este servidor ID ou a outro servidor ID, para casos em que a base de dados é partilhada.

Quando se fala que a chave privada do tipo *Server-Side key* é guardada no servidor, não quer dizer que seja mesmo a chave privada (criptográfica) gerada para o servidor ID, também pode ser informação que a chave segurança usa para gerar outra vez a mesma chave privada (criptográfica) usada no registo. Quando se faz uso de uma autenticação com chaves assimétricas é possível usar se o servidor ID para gerar as chaves, mas também é possível em conjunto com servidor ID usar um conjunto de caracteres *random*, para em caso o utilizador necessitar que seja recriada uma nova chave privada, para o mesmo servidor ID, essa seja diferente todas as vezes. Assim sendo, quando se fala na chave privada do tipo *Server-Side key* guardada no servidor, pode ser tanto a chave privada (criptográfica) como pode ser informação usada para gerar chave privada (criptográfica), isso está dependente do fabricante de chaves de segurança (autenticador), em que esse tem de escolher uma opção e proteger ambas na melhor forma possível.

Sabendo essas regras, será então demonstrado num gráfico que cumpre com essas regras sobre a forma como a Yubikey faz para criar as chaves, para encriptar e desencriptar a chave privada e ainda verifica se essa chave lhe pertence. Assim, começando pela criação, pode se visualizar na Figura 20, neste caso é usado o algoritmo ECC (o algoritmo é escolhido através da lista que o servidor fornece), em que essa recebe uma entropia e gera a chave privada. A partir da chave privada é possível então descobrir ou calcular a chave pública. Ainda na mesma Figura 20, é ainda possível visualizar o uso de criptografia autenticada na forma de *Encrypt-then-MAC* (EtM), usando a chave pública mestre do autenticador para encriptar a chave privada e usando o resultado do SHA-256 (servidor ID + User ID) para assinar a informação encriptada.

Seguindo o que está representado na Figura 20, será possível então cumprir as regras anteriormente mencionadas. Cumpre-se a utilização de chaves assimétricas usado ECC; cumpre-se a segurança de guardar a chave privada gerada usando o AES-256 em conjunto com as chave assimétrica mestre do autenticador para proteger; cumpre-se a identificação de cada chave a cada servidor ID, usando o

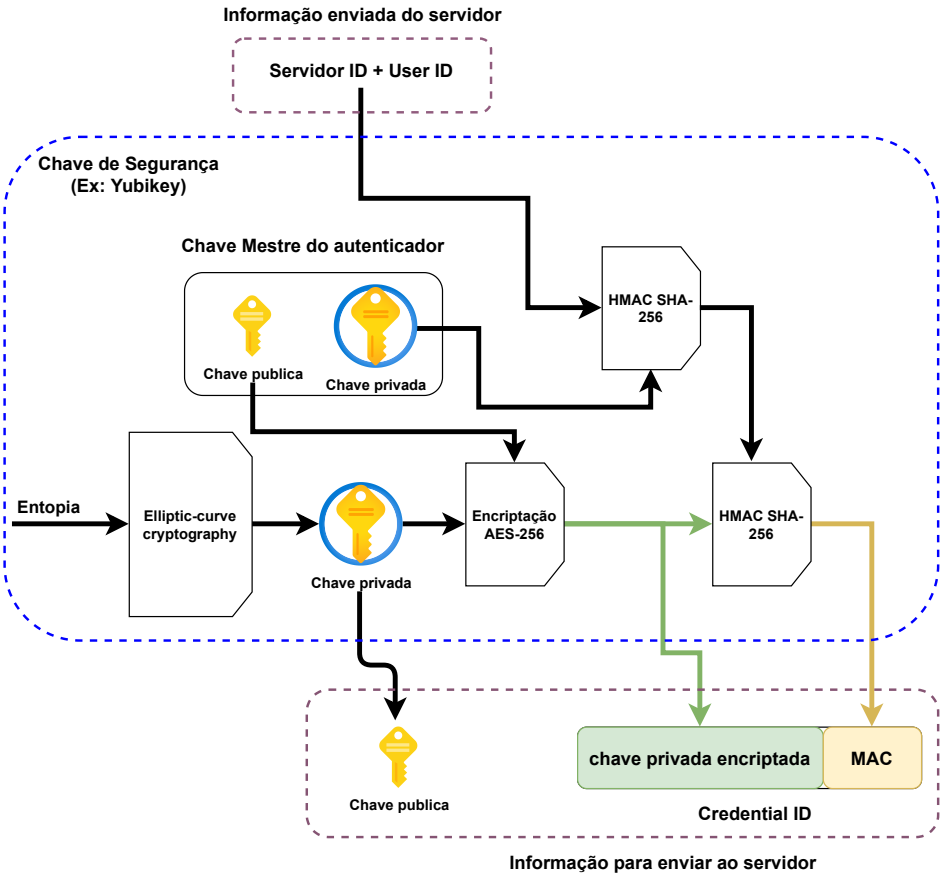

Figura 20: Criação de chave FIDO2 e a preparação da chave privada.

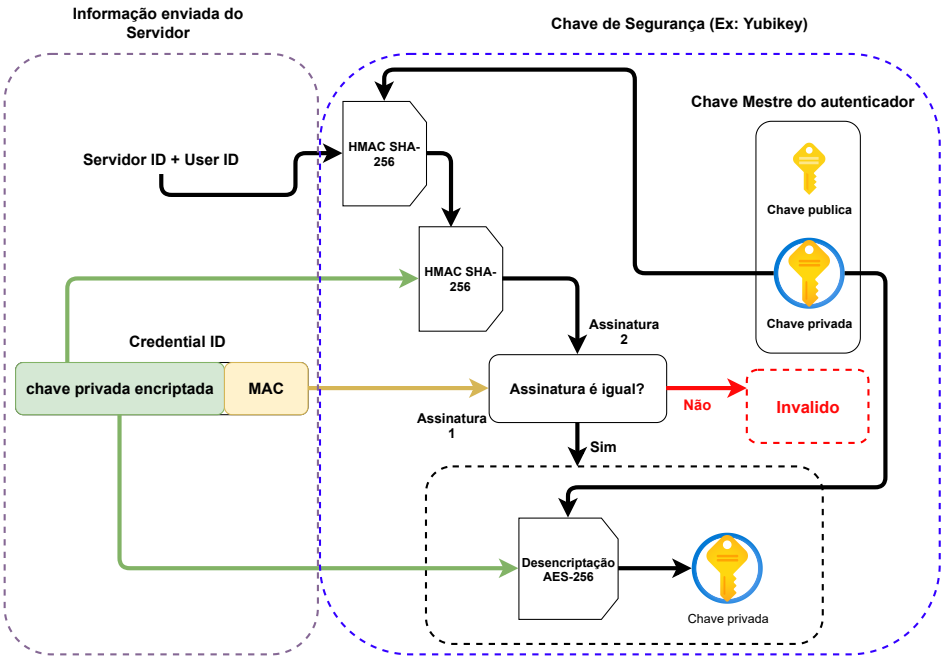

Figura 21: Validar e recuperar chave privada.

servidor ID e o do utilizador ID para assinar a chave privada que está encriptada; e por último, a regra em que não possibilita a identificação a quem pertence a chave privada, pois a chave privada mestre é usado como chave no SHA-256 (servidor ID + User ID), assim não possibilita no lado do servidor a identificação pois esses não conseguem criar o resultado do SHA-256 sem chave privada mestre.

Agora resta mostrar como é feita a recuperação da chave privada, mas primeiro é preciso saber uma regra: antes de desencriptar a chave privada, o autenticador tem de verificar que a chave privada foi ele que gerou e que pertence ao servidor ID que está a pedir para autenticar. Passando à demonstração, que se pode visualizar na Figura 21, é feito SHA-256 (servidor ID + User ID) com a chave privada mestre à chave privada que está encriptada, será assim possível verificar se as assinaturas são iguais, e assim também poder validar que foi o autenticador que criou essa chave privada e que pertence aquele Servidor ID. Ao saber que é válido, procede-se à desencriptação da chave privada usando chave privada mestre do autenticador, caso não seja válido, impede o processo de continuar e retorna um erro.

Relativamente à assinatura do desafio, para poder autenticar ou registar uma nova chave FIDO2, existem dois tipos de assinatura (Consortium, 2021a):

- **assinatura "attestation signature"** – Esta assinatura é produzida quando uma nova chave FIDO2 é pedida, em que fornece prova criptográfica de certas propriedades do autenticador e da credencial;

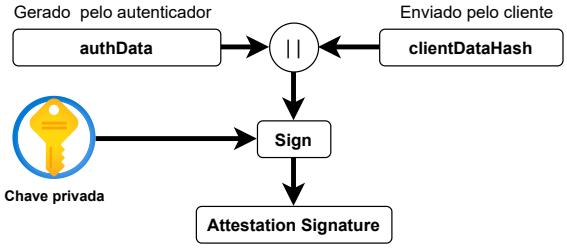

Figura 22: Processo de assinatura do "attestation signature".

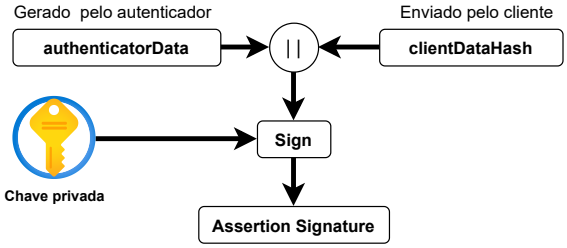

Figura 23: Processo de assinatura do "assertion signature".

- **assinatura "assertion signature"** – Esta assinatura é para casos de autenticação, em que representa que o utilizador deu o seu consentimento para uma transação específica, como autenticação ou autorizar uma transferência bancária.

No caso da assinatura "attestation signature" são usados dois dados, o *authData* (que contém o *hash* do servidor ID, contador, chave pública, credencial ID, Authenticator Attestation Globally Unique ID do autenticador, entre outros dados) e o *clientDataHash* (que contém o desafio, o tipo de ação a fazer e a origem do cliente), estes são concatenados e depois assinados com a chave privada selecionada pelo o utilizador, pode se visualizar na Figura 22.

No caso da assinatura *assertion signature* são usados dois dados, o *authenticatorData* (que contém *hash* do servidor ID, o contador, entre outros dados) e o *clientDataHash* (que contém o desafio, o tipo de ação a fazer e a origem do cliente). Estes dados são concatenados e depois assinados com a chave privada selecionada pelo o utilizador, pode se visualizar na Figura 23.

3.4 VANTAGENS DE USAR FIDO2

Existem diferentes formas para autenticar um utilizador, mas a melhor maneira de saber quais as tecnologias que tem melhor resultado é através do estudo da tecnologia e estatísticas. A Google efetuou um estudo em que demonstra alguns

Figura 24: Estatísticas dos métodos de autenticação da Google e na prevenção de *Bots* automaticos, *Phishing* e Ataques Focados (Thomas e Moscicki, 2019)

dos métodos de autenticação que a plataforma usa e indica quais têm tido melhor proveito na proteção contra *Bots*, *Phishing* e ataques focados ao utilizador (Thomas e Moscicki, 2019).

Na Figura 24 pode-se visualizar os métodos de autenticação, que são divididos em dois tipos: o desafio baseado em dispositivo e o desafio baseado em conhecimento. Comparando entre os dois tipos, o que tem mais proveito é o desafio baseado em dispositivo, mas o desafio baseado em conhecimento consegue prevenir os *Bots* automáticos tão eficaz como o outro tipo, só é pouco eficaz na prevenção dos *Phishing* e nos ataques focados, porque é possível enganar o utilizador de maneira a ele fornecer os dados que tem conhecimento. De um modo geral, o desafio baseado em dispositivo com uso de uma chave de segurança física tem o melhor sucesso nestas estatísticas a 100% em tudo, o que talvez queira dizer que até ao momento, a melhor tecnologia é que requer o uso de uma chave de segurança física, como exemplo Yubikey, e uma tecnologia que faça uso dessas chaves de segurança fisica, como exemplo o FIDO2.

O FIDO2 tem a capacidade de prevenir ataques como o *Phishing* e o uso de credenciais comprometidas. Estes tipos de ataques causam grandes problemas aos utilizadores, por não terem conhecimento ou por o servidor não ter a segurança devida. O FIDO2 previne esses ataques da seguinte forma:

- **Phishing** – Nos ataques de *Phishing* as entidades maliciosas fazem uso de caracteres especiais ou de nomes semelhantes. Usando o Paypal como exemplo, as entidades maliciosas criam um servidor web igual ao do Paypal, mas com o URL "`https://www.päypal.com`". Os utilizadores não diferenciam entre URL

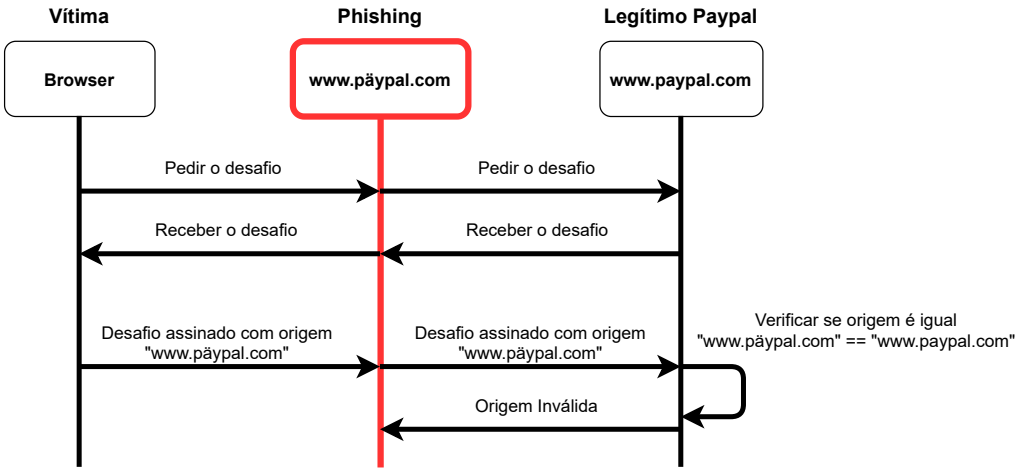

Figura 25: Simular um ataque *Phishing*.

malicioso do URL fidedigno, pois a diferença é apenas no caracter "ä", mas com esta diferença é redirecionado para um site malicioso. Este tipo de ataque designa-se por ataque Internationalized Domain Names (IDN) homográfico. O FIDO2 protege contra este tipo de ataques na autenticação uma vez que faz uso do URL para assinar, em que é colocado o URL no campo origem. Na Figura 25 pode-se visualizar que o site *Phishing* faz de intermediário entre a vitima e o Paypal legítimo, em que o objetivo da entidade maliciosa é conseguir obter o *token* de autenticação que é dado quando autenticação é concluída com sucesso. No caso do FIDO não é possível roubar a credencial na forma de ataque *Phishing* porque quando a vítima receber o desafio para assinar, é assinado o desafio em conjunto com a origem (a origem é a informação de onde o cliente está aceder, que neste caso é "`https://www.päypal.com`") e outros dados, e depois enviado, em que a identidade maliciosa reenvia o desafio assinado para o Paypal e este verifica que a origem não corresponde a uma origem legítima do Paypal, e o Paypal retorna uma mensagem de erro. Desta forma a entidade maliciosa não consegue obter o *token* de autenticação, tornando o FIDO uma boa solução contra o *Phishing*.

- **Credenciais comprometidas** – No caso das *passwords*, quando uma identidade maliciosa consegue obter as *passwords* da base de dados de um servidor ou usar *passwords* que são comuns, pode se considerá-las comprometidas pois a identidade maliciosa vai fazer uso de *brute-force* para descobrir as *passwords* ou uso de um dicionário de *passwords*. Neste caso o FIDO2 também protege contra base de dados comprometidas, pois mesmo que a chave pública do utilizador seja obtida, essa só tem como uso para verificar as assinaturas do

desafio, pois só a chave privada é que assina o desafio, então torna-se inútil a chave pública para a identidade maliciosa.

Muitos métodos de autenticação requerem o uso obrigatório do HTTPS, porque os dados que são transmitidos entre o cliente e o servidor são sensíveis, em que podem comprometer autenticação ou as credenciais usadas caso seja em HTTP. O mesmo acontece no caso da autenticação tradicional. A diferença entre o HTTP e o HTTPS é que no caso do HTTPS os dados entre o servidor e o cliente são encriptados usando Transport Layer Security (TLS) para proteger contra Man-In-The-Middle (MITM). Neste caso são usados certificados digitais (chaves assimétricas) e chaves simétricas. As chaves assimétricas são usadas para garantir que só o servidor recebe a chave simétrica e assim ter uma comunicação segura através da encriptação com chave simétrica. Desta forma, o FIDO também requer o uso do HTTPS, pois o FIDO é vulnerável aos ataques Man-In-The-Middle (MITM). Se por alguma razão os certificados digitais ou chave simétrica estiverem comprometidos, a autenticação também fica comprometida. A gravidade deste tipo de ataques difere em duas situações:

- **Ataque no acto do registo** – esta situação é a mais grave de todas, que é a situação quando o utilizador está a fazer o registo de uma chave FIDO. Neste caso a entidade maliciosa pode comprometer a autenticação da conta do utilizador. Como se pode visualizar na Figura 26, a vítima pede o desafio como normalmente. Esse pedido passa pela entidade maliciosa sem qualquer alteração, de seguida é recebido o desafio em que a entidade maliciosa reencaminha para a vítima. Para que essa não peça um novo desafio enquanto a entidade maliciosa regista com a sua própria chave FIDO e recebe a mensagem com sucesso, comprometendo assim a conta. Caso a vítima envie o desafio assinado, com a sua própria chave FIDO, é impedido ou interrompido para que não seja comprometida a tentativa da entidade maliciosa, pois o desafio só dá para uma vez, quem enviar primeiro ao servidor é quem fica com a chave FIDO registada. A segurança da conta para esta situação fica comprometida, principalmente se a vítima não reparar que existe uma chave FIDO que não lhe pertence.
- **Ataque no acto da autenticação** – Esta situação é menos grave que anterior, que é a situação quando o utilizador está a fazer o *login*, pois neste caso a entidade maliciosa pode comprometer a autenticação, só uma vez com os dados do utilizador. Neste caso, a entidade maliciosa tem duas opções: a opção de ficar só a fazer *sniffing* ou a opção *Replay Attack*. No caso de uso de autenticação por token basta o *sniffing* (pois no final da autenticação é

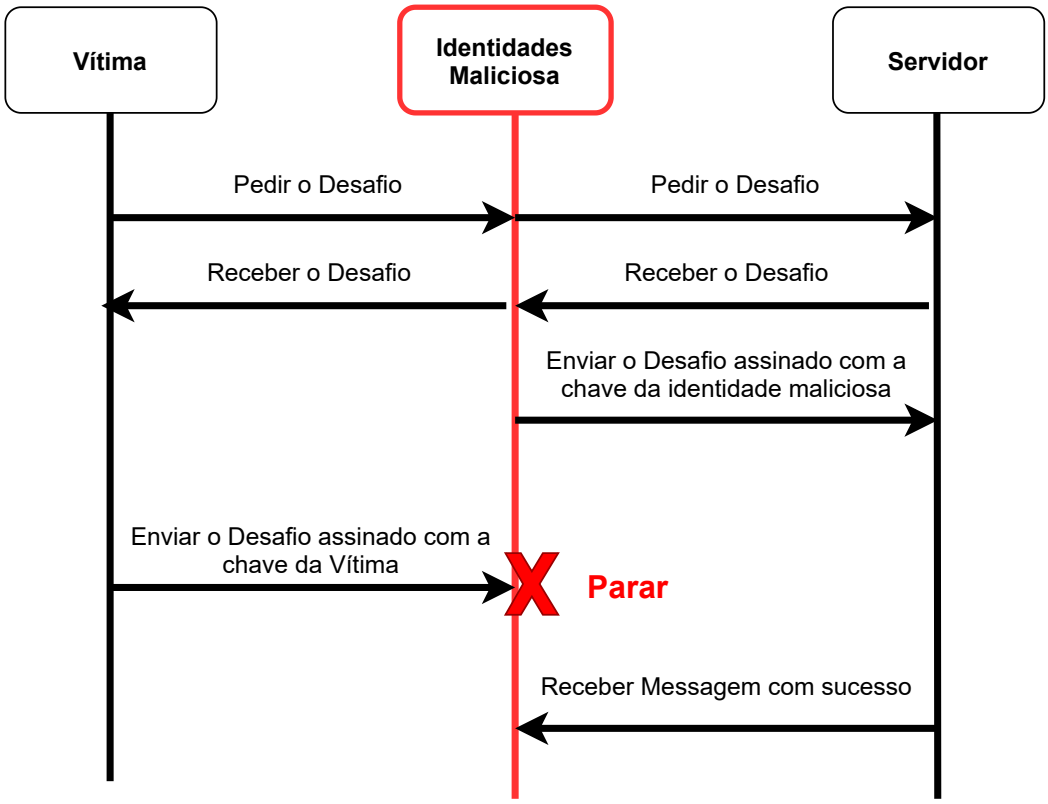

Figura 26: Simular um ataque MITM no ato do registo.

enviado o Token), já no caso do *Replay Attack* em que autenticação por token não exista ou se pretenda impedir a vítima de autenticar para que a entidade maliciosa possa fazer estragos na conta do utilizador enquanto a vítima não tem acesso, como exemplo remover todas as chaves FIDO. Pode-se visualizar a opção *Replay Attack* na Figura 27.

Outras grandes vantagens de usar o FIDO é a existência de certificações, é de grande relevância, pois assim todos os sistemas e dispositivos que fazem uso do FIDO estão conforme a documentação o diz. De acordo com Frank Dickson (Fido Alliance, 2021a) existe uma falta de certificações no uso de autenticação biométrica e com essa falta tem havido dificuldades para os profissionais de segurança, pois não existe referências de atributos importantes, nem um padrão de desenvolvimento. O FIDO vem acabar com essa situação de falta de certificação no uso de biometria, certificando a impressão digital, reconhecimento de voz, reconhecimento facial e reconhecimento da íris do olho.

O FIDO ainda tem mais uma proteção para ajudar a impedir o uso de autenticadores de fontes não confiáveis e de certificar que o autenticador tem o certificado que diz ter. Esta proteção é chamada de *FIDO Alliance Metadata Service* que

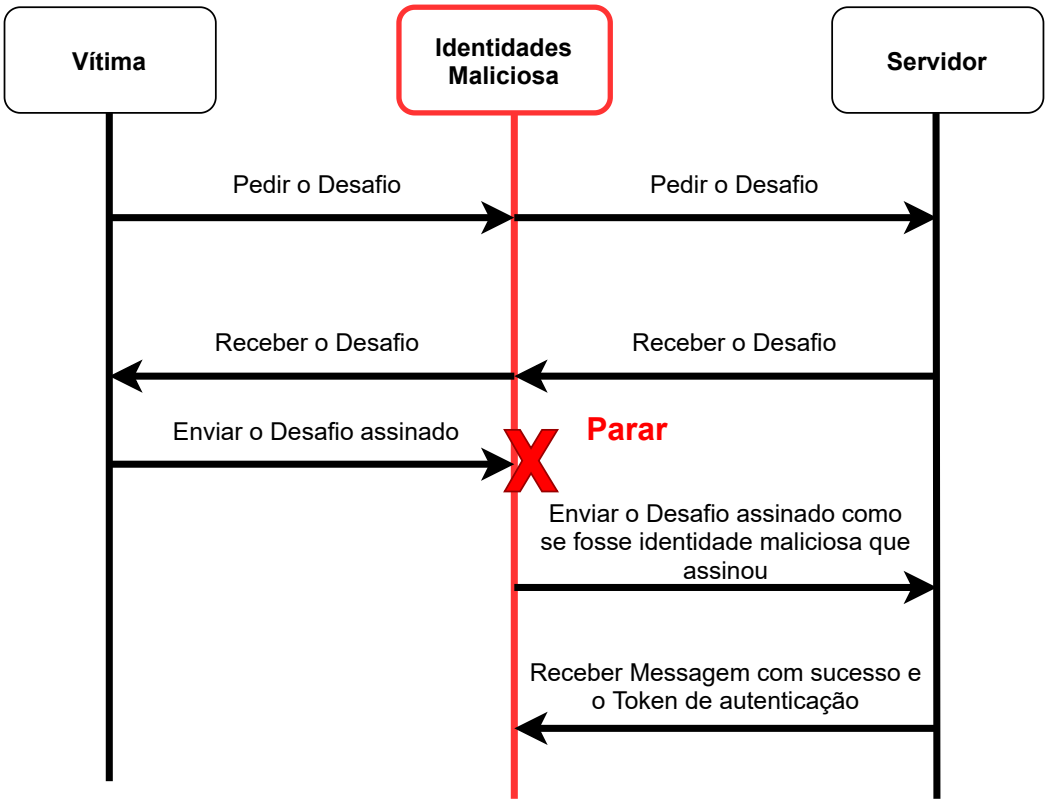

Figura 27: Simular um ataque MITM no ato do *login*.

contém a lista de modelos de autenticadores, o seu tipo de certificado e problemas de segurança conhecidos nesse autenticador, permitindo assim as organizações usar essas informações para especificar que tipo de nível de certificação que autorização e poder fazer uso das notificações de segurança para garantir uma resposta eficaz a incidentes (Fido Alliance, 2021d).

3.5 CERTIFICAÇÕES DO FIDO

O FIDO é uma nova forma de autenticação usando chaves assimétricas, para que este tenha a segurança que é capaz de fornecer, o desenvolvedor de software deve implementar o FIDO como é definido na documentação. Para que a implementação seja de forma segura, a FIDO Alliance fornece certificação para demonstrar que a implementação do FIDO está dentro dos parâmetros da documentação.

A FIDO Alliance fornece 3 tipos de certificação, a certificação funcional, a certificação de autenticação por níveis e a certificação de componente biométrica (Fido Alliance, 2019d).

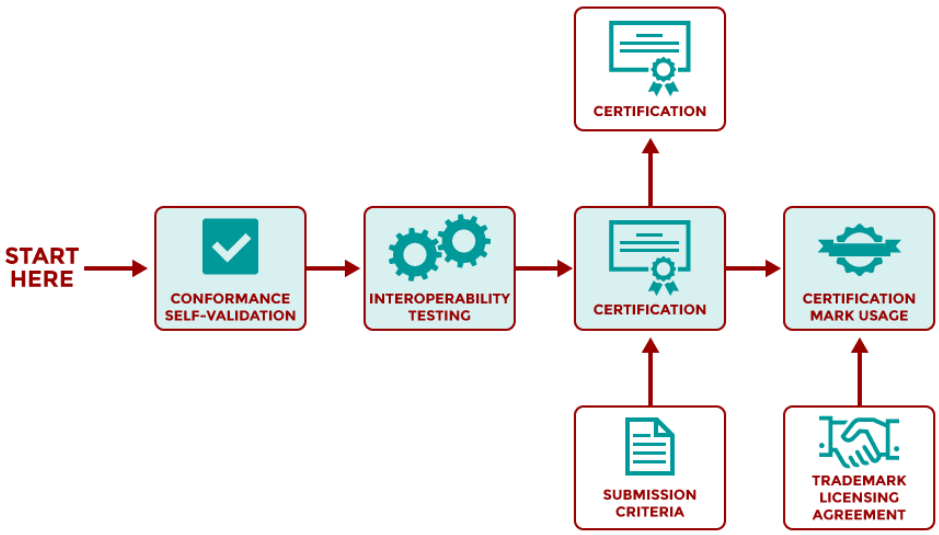

Figura 28: Processo para obter o certificação funcional (Fido Alliance, 2021e).

A certificação funcional tem como objetivo certificar serviços ou plataformas que façam uso do FIDO. Desta forma garante tanto à empresa como aos consumidores, que a plataforma está a cumprir as normas e que funciona corretamente. A certificação funcional segue o processo representado na Figura 28 (Fido Alliance, 2021e). Este processo permite que meçam a conformidade e garantam a interoperabilidade entre produtos e serviços que suportam as especificações FIDO, permitindo também as plataformas ou aos serviços de exibir o logotipo "FIDO Certified" para demonstrar aos utilizadores, consumidores e parceiros que contém uma implementação FIDO de alta qualidade.

A certificação de autenticação por níveis é uma certificação para autenticadores FIDO (chave de segurança). Existem 3 níveis de certificação (ou 6 níveis de certificação a contar com os níveis extras) e cada nível acima tem o que o nível anterior tem e mais algumas funcionalidades de segurança, os níveis são os seguintes (Fido Alliance, 2021b) (ver Figura 29):

- **Nível L1** – Este é o certificado mais básico, este nível avalia a proteção contra ataques básicos (exemplo phishing, MITM e violações à base de dados) (Fido Alliance, 2019a).
- **Nível L2** – Neste nível tem de conter medidas adicionais para impedir ataques mais avançados às chaves de segurança, ou seja, tem de ser resiliente a *malwares* que queiram extrair informações da chave de segurança do sistema operativo. Este nível tem de estar em conforme com as soluções como *FIDO Allowed Restricted Operating Environment* e *Allowed Cryptography lists*. Estas soluções

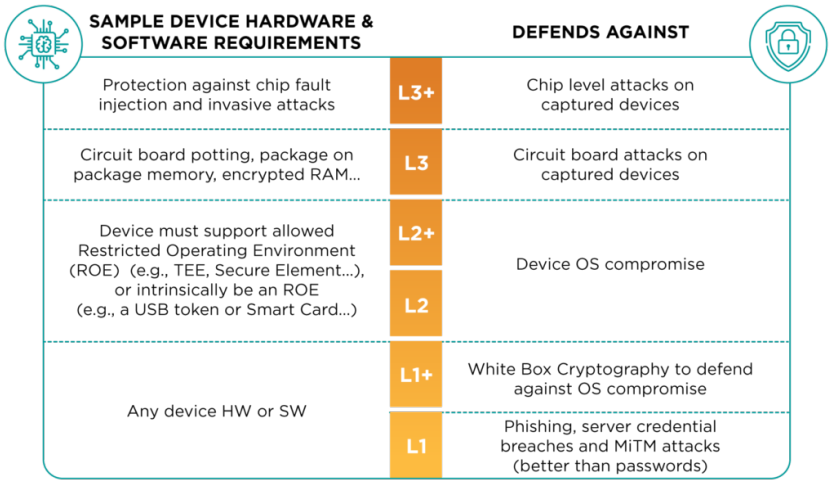

Figura 29: Certificação de autenticação por niveis (Tzur-David, 2020).

tentam restringir os acessos a secções importantes ou não existentes (Fido Alliance, 2019b).

- **Nível L3** – Este nível tenta prevenir ataques físicos básicos ao circuito elétrico, como exemplo ser usado um CPU com RAM encriptada e com um verificador de integridade, a chave de segurança deve prevenir sabotagem física e mostrar sinais claros de que a chave de segurança foi manipulada por uma entidade maliciosa (Fido Alliance, 2018).
  - **Nível L3+** – Este é um sub-nível do L3, e é o nível mais seguro de um autenticador FIDO, este certifica que não é manipulado o *chip* e que contém uma funcionalidade de segurança extra, que é o Trusted Platform Module (TPM). O TPM ajuda a proteger as chaves assimétricas criadas pelo autenticador (Fido Alliance, 2019c).

A certificação de componentes biométricas tem como objetivo certificar os autenticadores que fazem uso de um componente biométrico, em que este certifica o uso da impressão digital, do reconhecimento de voz, do reconhecimento facial e do reconhecimento da íris do olho (ver Figura 30). O certificado permite verificar a sua segurança e a confiabilidade das tecnologias de autenticação biométrica, esta é necessária para qualquer dispositivo com componente biométrico (Fido Alliance, 2021a). Para obter o certificado tem de ser testado a componente biométrica, por um laboratório com profissionais que criam uma lista de exemplos biométricos para testar o dispositivo e que criam um relatório das descobertas [2].

2 lista de laboratórios autorizados pelo FIDO Alliance em `fidoalliance.org/certification/biometric-component-certification/fido-accredited-biometric-laboratories/`

Figura 30: Processo para obter o certificação de componente biométrico (Fido Alliance, 2021a).

4

METODOLOGIAS DE DESENVOLVIMENTO

Foram adotados diferentes tipos de metodologia de forma a que pudesse ajudar a desenvolver este projeto de uma maneira fluída, neste caso uma adaptação da metodologia ágil com a metodologia em cascata. Desta forma o desenvolvimento do projeto foi realizado com base nos seguintes princípios fundamentais das duas metodologias:

- Metodologia em cascata:
  - Concluir uma etapa de cada vez, só avança para a seguinte quando anterior estiver concluída;
  - Escrever documentação ao longo do processo.
- Metodologia ágil:
  - Entregas rápidas e contínuas de software funcional;
  - Flexibilidade para mudanças tardias no projeto;
  - Entregas frequentes de software funcional de 15 em 15 dias;
  - Entrega de relatórios 15-5 de 15 em 15 dias;
  - Reuniões a cada 15 dias;
  - Uso da metodologia Scrum.

Na metodologia ágil são usados os princípios da metodologia ágil Scrum, que é conhecido pelos seus ciclos ou etapas de desenvolvimento chamados *sprints*, em que é definida uma duração fixa, que neste caso é de 15 dias. No início destas *sprints*, existe uma reunião para entregar o que foi feito na *sprint* anterior e para planear o que irá ser desenvolvido na *sprint* seguinte, dando assim início à nova *sprint*. No fim de cada *sprint*, ou seja, em cada reunião é entregue um relatório 15-5, que é entregue a cada 15 dias e tem cinco pontos para serem preenchidos, em que esses pontos são:

- Trabalho realizado - Em que é discutido o conteúdo em que foi realizado na *sprint* que terminou e que pode ser um pedaço de software ou uma pesquisa de informação;

- Trabalho a realizar - É o trabalho que se pretende efetuar na *sprint* que vai dar início e que pode ser um pedaço de software ou uma pesquisa de informação;
- Atrasos registados - Este ponto tem como objetivo deixar marcado as partes que ficaram atrasadas na *sprint*, isto pode acontecer quando o objetivo se tornou mais complicado ou mais trabalhoso do que se esperava;
- Maiores dificuldades - Descreve os problemas e as dificuldades que surgiram;
- Observações - Serve para deixar notas de informações ou deixar nota de algo que tem de se fazer nas *sprints* futuras.

Falando mais especificamente em que situações foi usada cada metodologia, segue-se a Figura 31 em que se pode visualizar a metodologia em cascata e o desenvolvimento de todo o trabalho de modo geral, em que se começou pela fase do "Estudo Teórico", de seguida a fase de "Levantamento de Requisitos", depois a fase de "Prova de Conceito" e por último a fase de "Implementação num Projeto Real". Em todas essas fases era realizada uma integração com o "Relatório" para assim não se perder informação e para melhor descrever o que foi feito.

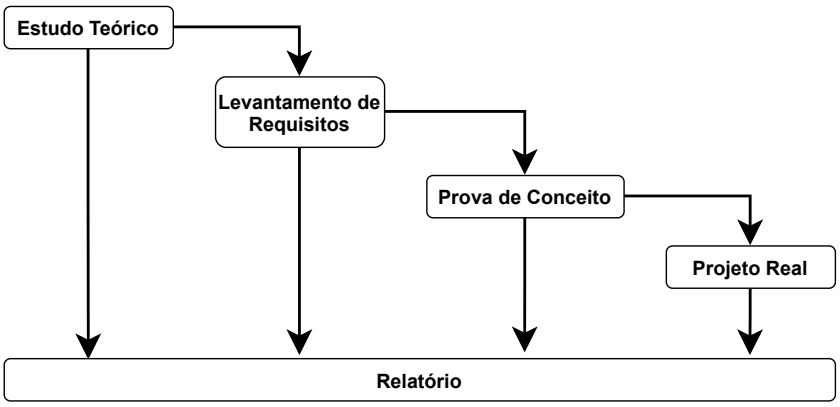

Figura 31: Metodologia geral.

Em todas estas fases da metodologia foi usada a metodologia ágil, ou seja, começando pela fase do "Estudo Teórico" até à fase de "Implementação num Projeto Real" e a fase do "Relatório", foram usados *sprints* de 15 em 15 dias. Até mesmo para situações em que não era desenvolvimento de software, pois esta forma permitiu documentar o que foi feito e definir prioridades tanto no estudo da informação, como no desenvolvimento do software. Em cada *sprint* foi seguido o processo que se encontra na Figura 32, em que começa na fase de "Levantamento de Requisitos", depois a fase de "Implementação ou Estudo" e por último "Testes Manuais ou Testes do Estudo", e depois o processo é ciclico, em que são usadas as reuniões bi-semanais como referência de fim e início de *sprint*.

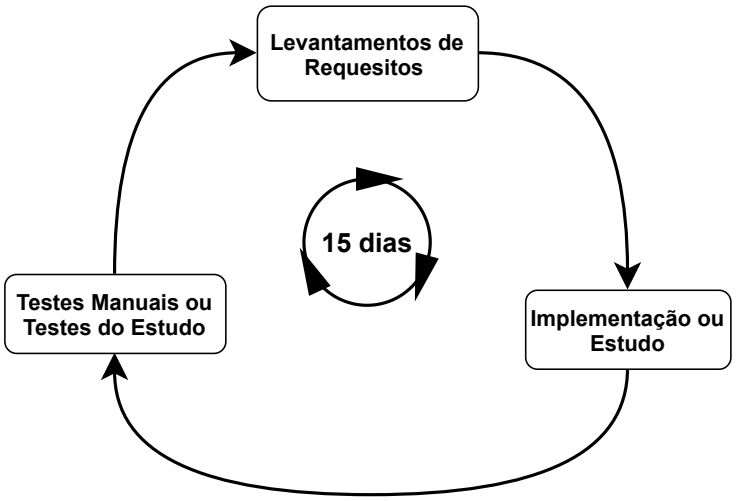

Figura 32: Metodologia scrum.

Indo em mais detalhe do processo da metodologia usada, este foi dividido em duas secções, o processo geral que está representado na Figura 31 e o processo Scrum que está representado na Figura 32.

4.1 PROCESSO GERAL

De seguida descrevem-se cada umas das fases de desenvolvimento deste projeto.

4.1.1 *Fase do Estudo Teórico*

Nesta fase foi reunida toda a informação que foi encontrada na documentação do FIDO2 e em publicações científicas feitas sobre o FIDO2. Este trabalho de pesquisa possibilitou saber como é que o FIDO2 funciona, como implementá-lo em dispositivos Android e em servidores Web. Nesta fase a informação recolhida foi colocada no capítulo Estudo Teórico, onde é possível verificar as informações encontradas sobre como o FIDO2 trabalha tanto ao nível dos seus protocolos como na assinatura e na criação de chaves.

4.1.2 *Fase de Levantamento de Requisitos*

O levantamento de requisitos feito inicialmente, foi uma parte bastante importante na organização do trabalho. Foi nesta fase em que foram definidos os requisitos mais

relevantes e com maior prioridade, o que possibilitou o planeamento do projeto de forma a garantir a sua fluidez de desenvolvimento. Os requisitos definidos foram os seguintes, ordenados primeiro pelos mais importantes:

- Compreender como usar a API do FIDO2 nas linguagens de programação selecionadas;
- Colocar o FIDO2 a funcionar na Web e no Servidor;
- Colocar o FIDO2 a funcionar no Android em conjunto com o Servidor;
- Procurar um projeto *open-source* para colocar o FIDO2 em funcionamento;
- Entender o funcionamento do projeto *open-source*;
- Implementar o FIDO2 no projeto *open-source*;
- Partilhar o projeto e a ideia do FIDO2 no projeto *open-source* selecionado.

4.1.3 *Fase de Prova de Conceito*

A fase de prova de conceito foi criado para comprovar o uso do FIDO2 e testar as suas funcionalidades, usado linguagens de programação que tinham suporte nativo para o FIDO2. Desta maneira pretendia-se aprender de forma eficaz a implementar o FIDO2 nas diferentes aplicações (Android, Web e Servidor), e poder assim fazê-lo num ambiente controlado.

4.1.4 *Fase de Implementação num Projeto Real*

Para esta fase foi selecionado o gestor de palavras passe *open-source* designado Bitwarden [1]. Para esta fase foram abordados 3 sub-projetos do Bitwarden: o servidor (*back end*), o serviço Web (*front end*) e aplicação Android. Foram realizadas alterações nestes sub-projetos de forma a colocar o FIDO2 a funcionar na plataforma Bitwarden. Por fim, estes projetos com as modificações realizadas, foram colocados em produção num ambiente controlado, para testes. Mais tarde, as alterações realizadas foram submetidas para apreciação do projeto Bitwarden com o objetivo de aumentar a segurança dessa plataforma na autenticação dos utilizadores.

1 https://bitwarden.com/

4.1.5 *Fase do Relatório*

A fase de documentação, em que são relatadas as informações encontradas e o trabalho realizado. O relatório foi desenvolvido em simultâneo com o desenvolvimento dos sub-projetos e das informações encontradas nas pesquisas.

4.2 PROCESSO SCRUM

4.2.1 *Fase de Levantamento de Requisitos*

Nesta fase são decididos que requisitos são necessários para completar a *sprint*. Os requisitos pode ser desde selecionar programas, até selecionar informações a utilizar. Também neste fase são definidos os objetivos para atingir no final da *sprint*. Alguns exemplos de objetivos são o estudo da API do FIDO2 no Android, ou implementar o FIDO2 em Android.

4.2.2 *Fase de Implementação ou Estudo*

Esta fase foi dividida em duas partes: pesquisa e implementação. Na fase de pesquisa foi realizado o estudo do FIDO2 ou de outra informação relevante para a sua compreensão e posterior documentação. Na fase da implementação foi criado código que permitisse implementar no software os requisitos e objetivos da *sprint*.

4.2.3 *Fase de Testes Manuais ou Testes do Estudo*

Esta foi a fase final, que se dividiu em duas partes. Primeiro foram realizados testes de estudo, que tem como motivo testar a informação encontrada, pois muita da documentação encontradas estava desatualizadas, tornando-se necessário confirmar o funcionamento. Depois vieram os testes unitários e de funcionalidades que tinham como objetivo testar se existiam erros na implementação de software ou se estava a ser validado corretamente. Caso fossem detetados erros, voltava-se à fase anterior até que esses problemas fossem corrigidos e as funcionalidades válidadas. A Tabela 3 mostra os diversos dispositivos e respetivos sistemas operativos usados para testar as funcionalidades do projeto durante o seu desenvolvimento.

Tabela 3: Dispositivos usados para desenvolvimento e testes durante o projeto.

Dispositivo	**Sistema Operativo**
PC	Windows 10 Education
PC	Windows 11 Pro
Redmi note 8T	Android 9
Redmi note 8T	Android 10
Samsung Galaxy A52S	Android 11
Samsung Galaxy A40	Android 10

5

PROVA DE CONCEITO

Neste capítulo serão abordados todos os aspetos relacionados com o desenvolvimento, funcionalidades, tecnologias usadas, a arquitetura, as suas aplicações e no final a estrutura da base de dados do projeto.

5.1 FUNCIONALIDADE

A funcionalidade da prova de conceito é autenticar um utilizador através de uma aplicação Android usando o FIDO2 no modo PasswordLess, ou seja, sem o uso de palavra-passe. Esta prova de conceito é composta por um *i)* servidor, que permite aos utilizadores autenticarem-se com FIDO2 através de um *browser* que suporte essa funcionalidade, e *ii)* uma aplicação Android que irá fazer os pedidos ao servidor. Esta parte do trabalho tem como objetivo testar e comprovar o uso da autenticação FIDO2 e não é destinado a ser usado em produção.

5.2 TECNOLOGIAS E PROJETOS USADOS

Nesta parte são descritas as tecnologias usadas para desenvolvimento da prova de conceito, em que é analisada a vantagem de cada tecnologia em comparação com outras. Como o objectivo é comprovar as funcionalidades do FIDO2 esta prova de conceito não foi pensada para ser usada em ambientes reais, para uso pessoal ou empresarial. Por esta razão, algumas das tecnologias que vão ser mencionadas foram escolhidas por motivos de eficácia. Ou seja, que não requeriam demasiado tempo de desenvolvimento pelo facto de já possuírem suporte nativo para o FIDO2 sem necessidade de instalar qualquer biblioteca externa.

As tecnologias usadas na prova de conceito foram:

- **Glitch** [1] – É uma simples ferramenta para criar aplicações Web que disponibiliza um serviço gratuito com um editor em tempo real. Isto significa

1 https://glitch.com/

que qualquer alteração feita no projeto em desenvolvimento será publicada automaticamente sem necessidade de inserir comandos para atualizar o servidor. Além disso, possibilita que outros desenvolvedores possam clonar o projeto. A escolha deste serviço deveu-se aos seguintes factos: ter servidor web incorporado; ser gratuito; por oferecer certificado digital; e por fim, por oferecer um domínio DNS para o projeto;

- **Node.js** [2] – É um ambiente de Javascript que executa código Javascript no lado do servidor, que o torna fácil de programar e de incluir projetos existentes no domínio público para auxiliar o desenvolvimento do servidor;
- **simplewebauthn/server** [3] – É uma biblioteca javascript, para ser executada do lado servidor, em que a estrutura e informação do FIDO2 é enviada para o cliente e depois validada. Isto significa que permite verificar se o desafio está assinado com uma chave privada correta, ou seja, que faz par com uma das chaves públicas registadas no servidor;
- **Json** [4] – Tecnologia usada tanto na parte de transmissão de informação entre o cliente e o servidor, como também na base de dados. Tem a vantagem de ser um modelo fácil de ler para os humanos, o que torna o desenvolvimento fácil de usar e fluído de implementar;
- **Android Studio** [5] - Ambiente de desenvolvimento de aplicações Android e permite ainda emular um dispositivo Android para testar as aplicações. A sua principal vantagem é ter disponível um emulador;
- **Kotlin** [6] - É uma linguagem de programação multi-plataforma, orientada a objetos e funcionalidades, possibilita criar aplicações com menos linhas de código comparando com o Java. Além disso, ainda possibilita integrar Java juntamente com o Kotlin. Para a prova de conceito foi escolhido o Kotlin por ser uma linguagem dos mesmos criadores do Android, o que possibilita ter fácil acesso a biblioteca FIDO2 já existentes nas versões mais recentes do sistema operativo Android.

Os projetos usados:

- **Template de FIDO2 no Glitch** [7] - É o modelo que a Google usa para demonstrar o FIDO2. No entanto, este projeto contém vários erros devido

2 https://nodejs.org/en/
3 https://simplewebauthn.dev/docs/packages/server/
4 https://www.json.org/
5 https://developer.android.com/studio
6 https://developer.android.com/kotlin
7 https://webauthn-codelab.glitch.me/

ao facto de não acompanhar as atualizações que o FIDO2 teve. Ainda assim, demonstrou ser útil por ter uma base do FIDO2 nas versões anteriores e ter uma interface Web base já construída.

5.3 ARQUITETURA

Este é um dos pontos mais críticos na construção de um sistema de software, pois sem uma arquitetura bem definida a complexidade de implementação aumenta e a compreensão do mesmo diminui. A arquitetura de software consiste na definição de interligação entre os componentes do sistema, como interagem e se relacionam entre eles. Desta forma pretende-se garantir benefícios tais como o aumento do desempenho, a simplicidade, a compreensão do sistema e a garantia de escalabilidade. Começando pela arquitetura geral do projeto, que engloba os três principais componentes: o servidor API, a aplicação Web e a aplicação móvel. A comunicação entre eles está representada na Figura 33 em que o canal de comunicação consiste em métodos HTTP (`Get`, `Post`, `Put`, `Delete`), para que as aplicações do cliente possam fazer pedidos de autenticação ao servidor e poderem assim ter acesso às suas contas.

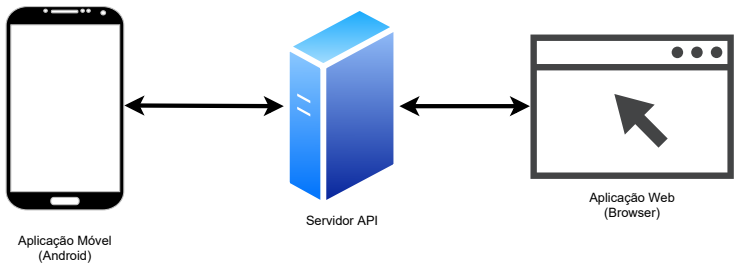

Figura 33: Arquitetura geral da prova de conceito.

De seguida apresenta-se a arquitetura do servidor. Este inclui o serviço Web, para que o *browser* do cliente possa receber as páginas a que quer aceder, e inclui também o serviço de API, que possibilita as aplicações do lado do cliente a fazer recolha de informação e pedidos de autenticação. Na arquitetura todos os pedidos são enviados para `server.js` que depois são redirecionados para o lugar correto em que devem ser tratados. Quando o pedido é uma página Web, este é redirecionado para a página correta em `views`. Quando os pedidos são para API esses são redirecionados para a pasta `api` que contém ficheiros necessários para tratar os pedidos, cujo endereço é `mcif.glitch.me/api`. Neste caso só existe um ficheiro que é o `auth.js`, que trata os pedidos de autenticação que são enviados para o endereço que começa o seguinte URL: `mcif.glitch.me/api/auth`. Também existe o caminho `.data/db.json`, que é a base de dados onde estão armazenados os dados dos utilizadores e os respetivos

tokens para autenticação. Existe ainda o `simplewebauthn/server` que é biblioteca FIDO2 para o lado do servidor. Esta é responsável por criar respostas FIDO2 estruturadas aos pedidos FIDO2 do cliente e verificá-los.

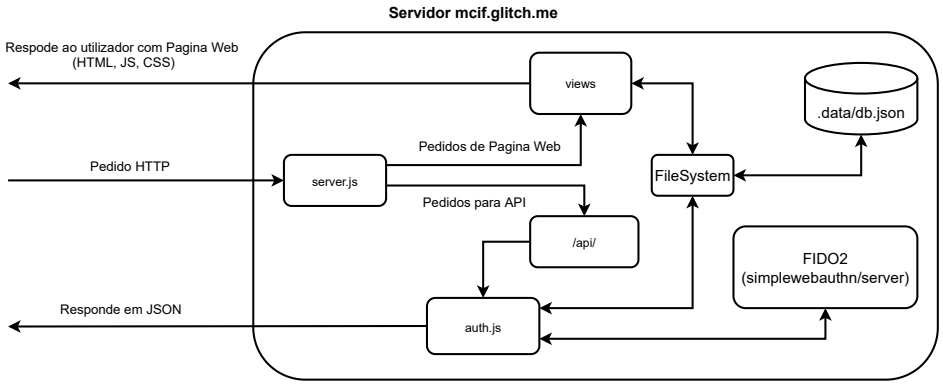

Figura 34: Arquitetura do servidor da prova de conceito.

No caso da arquitetura da aplicação Web supomos que o pedido da página Web ao servidor tenha sido feito com sucesso como está representado na Figura 35. A aplicação tem dois componentes importantes: os ficheiros do servidor e a API do navegador (`navegator`). Os ficheiros enviados para o servidor incluem a página, que contém `html`, `javascript` e o `css` necessários para apresentar a página ao utilizador. Contém ainda o ficheiro `cliente.js` em `javascript`. Este ficheiro serve como intermediário entre as ações do utilizador e a API do servidor, ou seja, onde são criado os pedidos para a API do servidor e tratadas as respostas do pedido. Os *browsers* que são compatíveis com o FIDO2 têm na API um elemento chamado `navigator` que por sua vez contém `credentials`. É neste último elemento que se pode pedir ao *browser* para abrir a janela do FIDO2, para que o utilizador possa assinar o desafio proposto pelo servidor.

De seguida descreve-se a arquitetura na aplicação móvel Android. A aplicação Android tem duas partes principais: a parte da aplicação móvel e a parte `com.google.android.gms.fido.fido2` onde está presente a API do FIDO2. A parte da aplicação móvel foi desenvolvida segundo o padrão MVC (Model–View–Controller). Neste modelo temos o `view` que é a interface da aplicação, temos o `model` que é o modelo dos dados (como exemplo uma classe utilizador, que contém o nome *username* e outros). Existe ainda o `controller` que é a parte responsável pelo tratamento da parte lógica da aplicação, ou seja, onde são criados os pedidos e tratado as respostas dos pedidos efetuadas à API do servidor.

A parte `com.google.android.gms.fido.fido2` contém o componente `Fido2APIClient` que é o principal. Este é responsável por pedir ao sistema operativo Android que

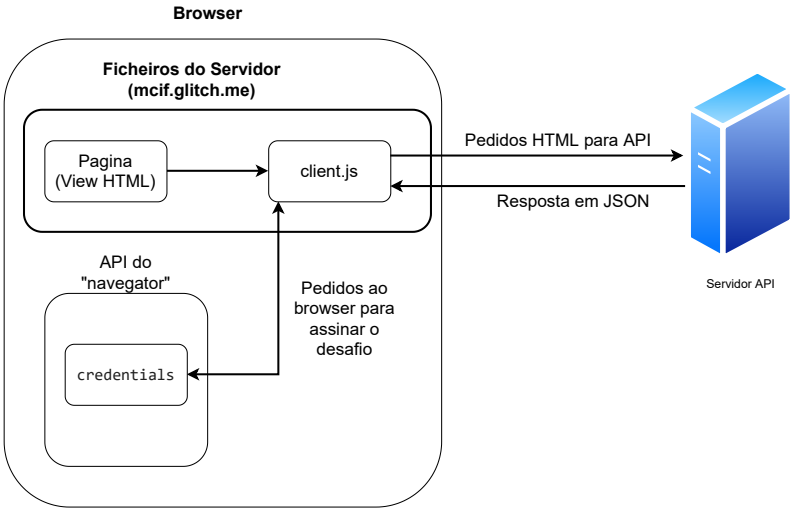

Figura 35: Arquitetura da aplicação Web da prova de conceito.

apresente ao utilizador a indicação para usar o FIDO2 para assinar o desafio. Existe ainda o `com.google.android.gms.fido.fido2.api.common` que contém o `AuthenticatorAssertionResponse`, o `AuthenticatorAttestationResponse` e o `AuthenticatorErrorResponse` que servem para desserializar os dados enviados do servidor.

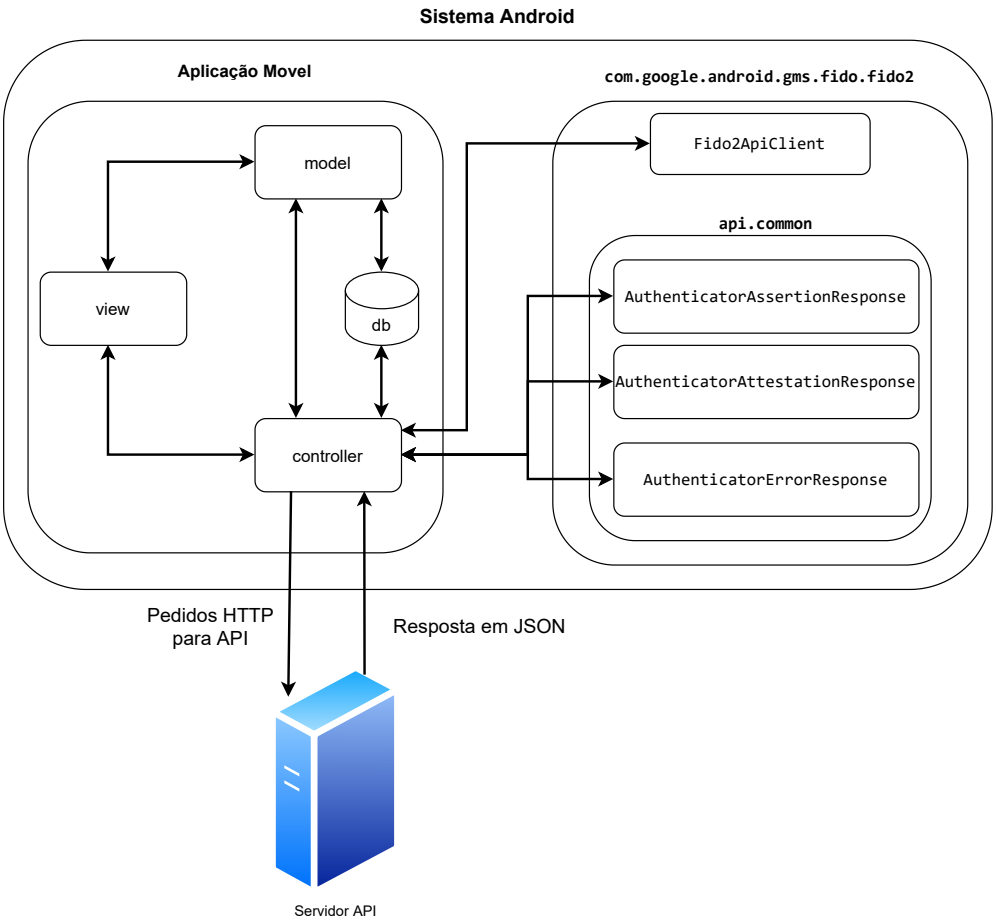

Figura 36: Arquitetura da aplicação móvel.

5.4 BASE DE DADOS

A informação necessária para a aplicação está armazenada em JavaScript Object Notation (JSON) e a sua escolha foi motivada pela simplicidade que apresenta. Foi decidido utilizar o JSON por não ser necessário instalar bibliotecas para gerir ou guardar os dados e porque o JSON não se preocupa que tipo de dado são guardados. Neste caso os dados guardados são o `ID`, `username` e as credenciais, tudo num só ficheiro, como se pode ver nas Tabelas 4 e 5.

Tabela 4: Dados guardados do utilizador.

Nome	**Explicação**
ID	ID de identificação de utilizador
username	username do utilizador para autenticar
credentials	Lista de chaves FIDO2 registadas

Tabela 5: Dados guardados por cada chave FIDO2.

Nome	**Explicação**
credID	ID de identificação da chave FIDO2
publicKey	Chave Pública do tipo FIDO2
prevCounter	Contador da chave
transports	Lista de tipos de transporte permitido

5.5 PRODUTO FINAL

O desenvolvimento da prova de conceito está dividido em duas aplicações: *i*) aplicação Web e *ii*) aplicação móvel. Ambas contêm o mesmo número de ecrãs e em cada ecrã contém o mesmo conteúdo. Por esta razão, antes da demonstração da explicação das funcionalidades, irão ser mostradas as rotas possíveis de ecrã como está representado na Figura 37.

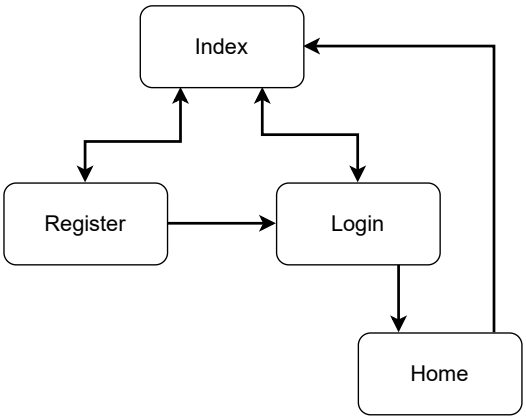

Figura 37: Arquitetura de ecrãs.

Para além destes ecrãs, existem ainda os ecrãs que são do sistema operativo na parte de autenticação FIDO2. Três ecrãs no sistema Windows e dois ecrãs no sistema Android. Entre o Android e o Windows existem dois ecrãs em comum, que é um ecrã para assinar com o tipo `Platform` e o outro ecrã e para assinar com o tipo `Cross-Platform`. O Windows dispõe de um terceiro ecrã que só aparece quando a conta de utilizador tem mais do que um tipo de autenticação FIDO2 (`Platform` ou `Cross-Platform`). No caso do Android não aparece, porque o Android verifica se existe alguma chave pública do tipo `Platform` vindo do *smartphone* onde está a ser executado aplicação. No caso de ser verdadeiro mostra o ecrã `Platform`, caso contrario mostra o ecrã `Cross-Platform`.

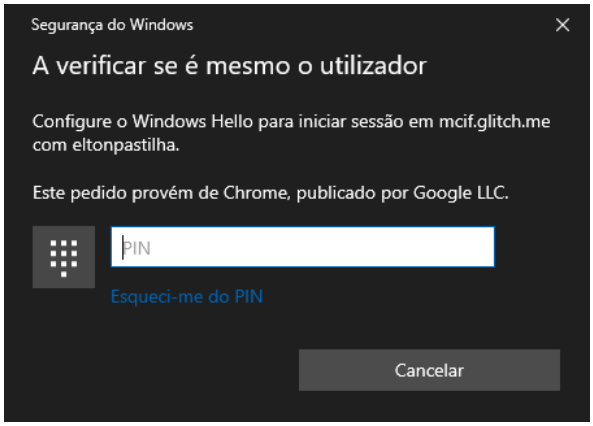

Figura 38: Ecrã "Platform" do sistema Windows 10.

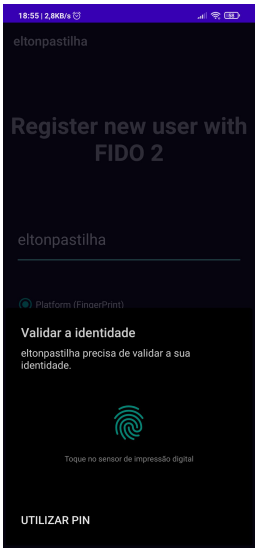

Figura 39: Ecrã "Platform" do sistema Android 9.

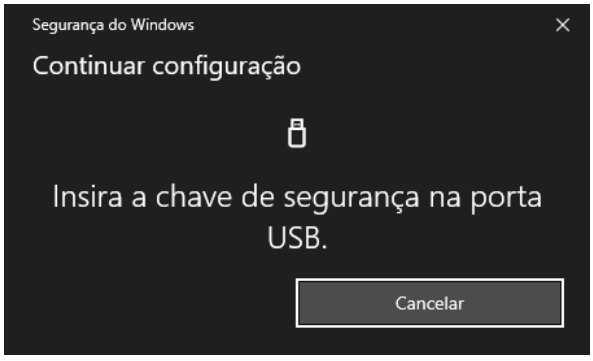

Figura 40: Ecrã "Cross-Platform" do sistema Windows 10.

Figura 41: Ecrã "Cross-Platform" do sistema Android 9.

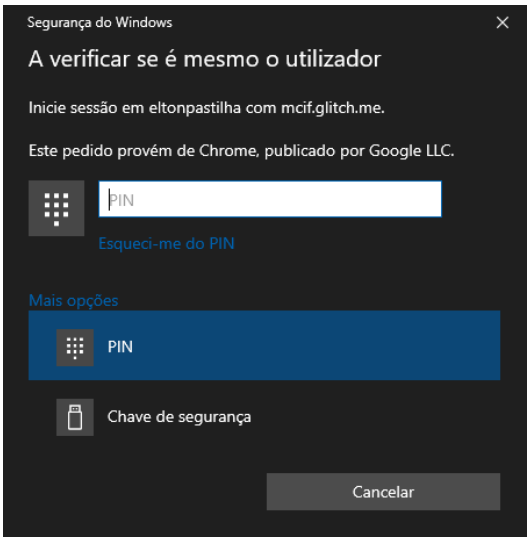

Figura 42: O terceiro ecrã do sistema Windows 10.

O primeiro ecrã que utilizador interage é o ecrã `index`, neste ecrã o utilizador recebe duas escolhas, a opção de registar uma conta de utilizador ou a opção de fazer autenticação, como se pode ver nas Figuras 43 e 44.

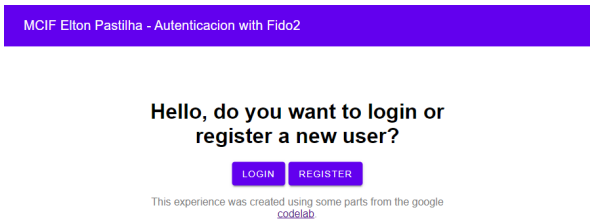

Figura 43: Ecrã `index` da Aplicação Web.

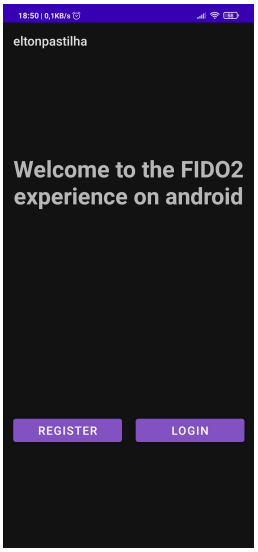

Figura 44: Ecrã `index` da Aplicação Android.

Caso seja selecionada a opção de registar uma conta de utilizador, será mostrado o ecrã `register`. Aqui é possível registar o utilizador, em que será necessário inserir o *username* e escolher qual é tipo de autenticação FIDO2 desejada (`Platform` ou `Cross-Platform`). Ao submeter os dados é mostrado uma janela nova do sistema FIDO2 para assinar o desafio de autenticação FIDO2 escolhido (`Platform` ou `Cross-Platform`). Depois de verificados os dados com sucesso o utilizador é redirecionado para o ecrã `login`.

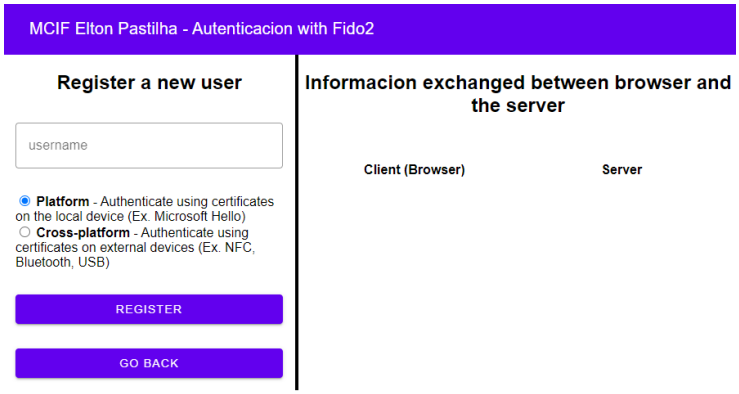

Figura 45: Ecrã "register" da Aplicação Web.

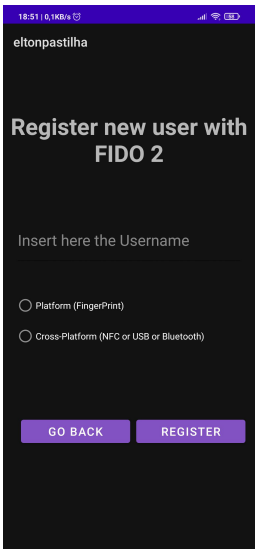

Figura 46: Ecrã "register" da Aplicação Android".

No caso da aplicação Web, será possível visualizar, no lado direito da página, os dados trocados entre o cliente e o servidor relacionados apenas com a parte de autenticação FIDO2.

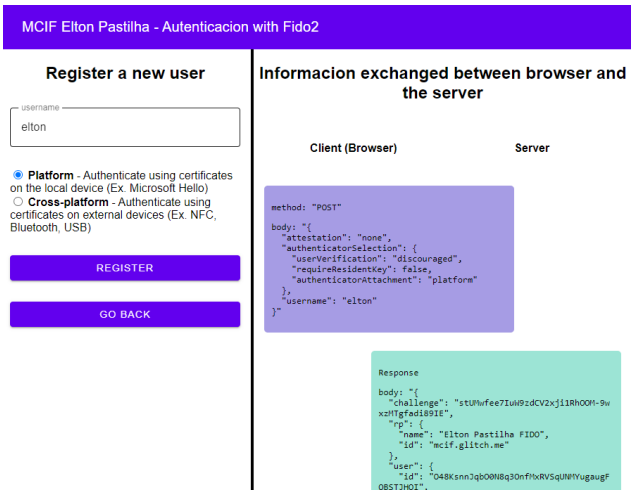

Figura 47: Ecrã "register" da Aplicação Web com informação de autenticação FIDO2.

Caso seja selecionada a opção de autenticar, ou redirecionar, será mostrado o ecrã `login`, onde é possível fazer a autenticação. O utilizador terá de inserir o *username*, ao submeter os dados é mostrado uma janela nova do sistema FIDO2 para assinar o desafio, em que tem de escolher o tipo de autenticação FIDO2 (`Platform` ou `Cross-Platform`). Quando existe mais que um tipo registado na conta do utilizador, depois de escolhido e assinado, serão verificados os dados e em caso de sucesso é redirecionado para o ecrã `home`.

Figura 48: Ecrã "login" da Aplicação Web.

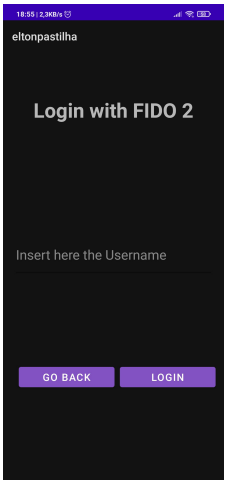

Figura 49: Ecrã "login" da Aplicação Android".

No caso da aplicação Web, será possível visualizar no lado direito da página os dados trocados entre o cliente e o servidor relacionados apenas com a parte de autenticação FIDO2.

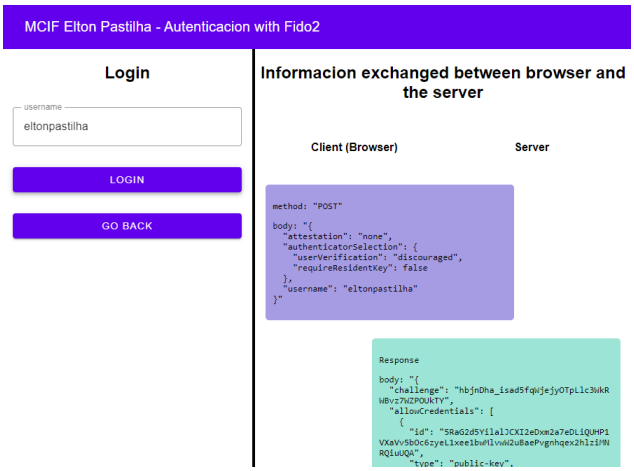

Figura 50: Ecrã "login" da Aplicação Web com informação de autenticação FIDO2.

É no ecrã `home` onde é possível gerir as chaves de segurança da conta autenticada. Aqui é possível ver a lista das chaves de segurança registadas e ainda remover e adicionar chaves de segurança. No caso de adicionar uma chave de segurança, é da mesma forma que no ecrã `register`, a diferença é que não é necessário inserir o *username* já que o sistema já tem essa informação. No caso de pressionar o botão `Sign Out` o utilizador é redirecionado para o ecrã `index` e deixa de estar autenticado.

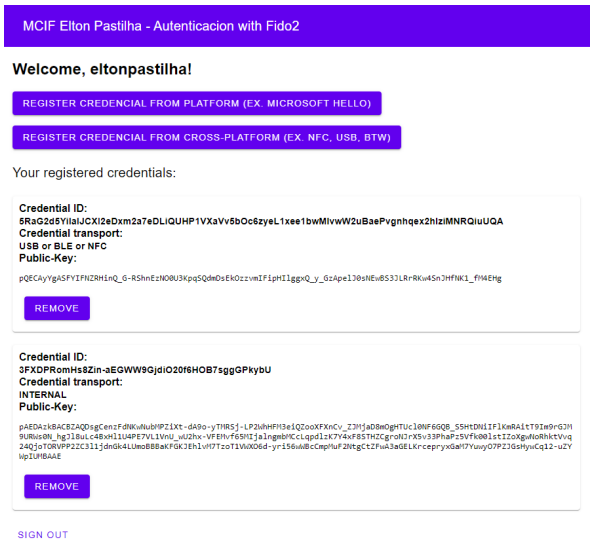

Figura 51: Ecrã "home" da Aplicação Web.

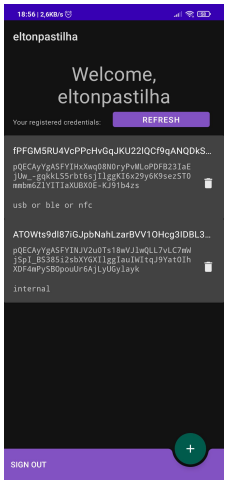

Figura 52: Ecrã "home" da Aplicação Android".

6

FIDO2 NO BITWARDEN

Neste capítulo serão abordados todos os aspetos da implementação do FIDO2 no gestor de palavras passe *open source* Bitwarden. Desde o estudo do *software*, ao desenvolvimento, que inclui as funcionalidades adicionais implementadas no Bitwarden, as tecnologias usadas, a arquitetura do projeto e das suas aplicações, e no final, a estrutura da base de dados.

6.1 BITWARDEN

O Bitwarden é uma plataforma que auxilia tanto organizações como utilizadores individuais, a gerir as palavras passe de acessos a sistemas ou a outras plataformas. Assim, pretende-se que os utilizadores deixem de utilizar a mesma palavra passe para diferentes sistemas ou plataformas. O Bitwarden é *open source* (fonte aberta), contém a possibilidade de ser usado gratuitamente com armazenamento até 1 GB e é aberto à comunidade para fazer alterações ou sugerir ideias para a sua plataforma. Na data da redação deste relatório, este software é compatível, através de página Web, com Android, iOS, Windows, macOS, Linux e vários *browsers* (Chrome, Firefox, Opera, Edge, Safari e entre outros) através de *plugins*.

Relativamente à sua segurança como é *open source*, possibilita que peritos em segurança e desenvolvimento de *software* possam auditar o seu código. Assim são melhorados os seus algoritmos, a sua arquitetura e a sua segurança. Esta aplicação usa encriptação de ponto-a-ponto, administração de controlos e aplicações controladas (Orenstein, 2020).

6.1.1 *Encriptação ponto-a-ponto*

Relativamente à encriptação ponto-a-ponto, o Bitwarden refere que só é possível desencriptar ou visualizar os dados do cofre das palavras passe através do endereço de email e *password* mestre. Toda a informação antes de ser enviada ou recebida é encriptada, até mesmo quando aplicação fica em *standby*, os dados são novamente

encriptados e protegidos, obrigando assim o utilizador a inserir novamente a *password* mestre. No caso das organizações, cada uma tem sua própria chave de criptografia que é partilhada com membros autorizados dessa organização, para aceder à informação. O algoritmo usado para encriptar o cofre é AES-CBC (Cipher Block Chaining).

No caso da *password* mestre é usado PBKDF2 (Password-Based Key Derivation Function 2) com o SHA-256 para a proteger. Quando inserido a palavra passe antes de ser enviada é efetuado o *salt* (saltear a *password*) e a *hash* com o email do utilizador. Quando é recebida do lado do servidor a *hashed password* é novamente efetuada o *salt* com um valor aleatório criptograficamente seguro e de seguida o *hash* desse conjunto. Com isto está pronta a ser guardada na base de dados. No caso de organizações é usado o RSA-2048 para criar a chave de acesso.

6.2 PROJETOS DO BITWARDEN

Como foi referido no capítulo anterior, o Bitwarden suporta muitas plataformas e por isso foi dividido em múltiplos projetos. Esta divisão deve-se ao uso de diferentes tecnologias de desenvolvimento e para facilitar também a organização do desenvolvimento de todas as aplicações. O Bitwarden contém 11 projetos em que 7 desses 11 são aplicações, 2 são a documentação e formas de ajuda, e os outros 2 são bibliotecas auxiliares que têm código ou ficheiros comuns aos 7 projetos de aplicação. Desses 11 projetos os únicos que serão abordados neste trabalho são 3: o *Server*; interface Web; e o *Mobile*. Estes foram escolhidos ou pela sua necessidade ou por serem o foco deste trabalho.

O projeto *Mobile* é o foco deste trabalho – FIDO2 em dispositivos móveis – que este inclui o código usado para criar a aplicação Bitwarden para Android e iOS. O *Mobile* foi desenvolvido com a ferramenta Xamarin da Microsoft, que possibilita construir uma aplicação para diferentes dispositivos como Android, iOS, tvOS, watchOS, macOS, and Windows. O desenvolvimento é realizado em .NET, uma plataforma que inclui a linguagem de programação C#, bibliotecas de base (para trabalhar ficheiros, textos, datas, dispositivos de entrada e saída, *etc*) e ferramentas de editar.

Tal como o projeto *Mobile*, o projeto *Server* também foi desenvolvido usando o editor Visual Studio. O *Server* foi escolhido pela necessidade de testar as funcionalidades do Android e para poder fazer a demonstração num local controlado. Assim, o *Server* possibilita responder aos pedidos do Android e guardar os dados. Também

o *Server* faz uso do .NET, mas este ainda usa as ferramentas e a estrutura de Web do ASP.NET Core.

Com a escolha do projeto *Server*, foi decidido que o projeto Web também seria necessário para poder testar o FIDO2 no Bitwarden em diferentes sistemas operativos. No caso do projeto Web este é desenvolvido em Angular, que é uma *Framework* desenvolvida em javascript. Assim, decidiu-se modificar os 3 projetos para poder fazer a demonstração e também para ajudar na segurança no Bitwarden, pois o Bitwaden só suporta FIDO1 U2F no lado Web (do que foi possível testar). Desta forma pretende-se trazer mais segurança nas aplicações Android e atualizar o FIDO no lado Web e *Server*.

6.3 TECNOLOGIAS USADAS NO PROJETO

Nesta secção serão abordadas as tecnologias usadas para o desenvolvimento do Projeto, que foram analisadas consoante a necessidade e a compatibilidade com os projetos Bitwarden. Assim, será possível enquadrar a funcionalidade do projeto, de forma produtiva e eficaz. As Tecnologias usadas:

- **Microsoft Visual Studio Professional 2019** - Escolhida para desenvolver a aplicação Android e o servidor, foi escolhida porque o próprio projeto tinha sido criado e desenvolvido por esta ferramenta e também porque foi utilizada em ambos os projetos desenvolvidos em .NET;
- **Microsoft Visual Studio Code** - Escolhido para o desenvolvimento da aplicação Web que está em Angular, esta ferramenta tem extensões para tratar projetos do tipo Angular. Também foi escolhido pela sua capacidade de indentação e correção de múltiplos tipos de ficheiros que existem no projeto Web, *Mobile* e *Server*. Outra razão é a sua capacidade de pesquisa nos ficheiros que o Microsoft Visual Studio Professional 2019 não tem, que é de grande auxílio no estudo de como funcionam os projetos;
- **Docker Desktop** - Ferramenta que auxilia na atualização e na gestão de imagens de Docker, no servidor "`hub.docker.com`".
- **DigitalOcean** - É uma plataforma que fornece um servidor à medida, para poder alojar o serviço Bitwarden modificado, para criar o ambiente controlado e poder testar sem que cause problemas a outros utilizadores do Bitwarden;
- **Putty** - Escolhida para autenticar e abrir uma shell remotamente no servidor;

- **WinSCP** - Ferramenta para trocar e visualizar os ficheiros que existem no servidor;

6.4 ARQUITETURA DO BITWARDEN

Compreender a arquitetura do Bitwarden é o ponto crítico para a compreensão do funcionamento do servidor, da aplicação móvel e da aplicação Web. Sem compreender a arquitetura existente torna-se complicado, ou até mesmo impossível, adicionar qualquer funcionalidade ao projeto. É preciso ter em conta que o Bitwarden foi desenvolvido por terceiros e por esta razão o seu estudo foi a parte que mais tempo consumiu no presente trabalho. A arquitetura geral do Bitwarden é relativamente simples, neste capítulo o foco recai sobre a parte da arquitetura relativa à autenticação e da autenticação de dois factores. Assim, não serão detalhadas outras funcionalidades disponíveis no Bitwarden.

Começando pela arquitetura geral, o Bitwarden engloba os seguintes três componentes principais: o servidor (disponível em `eltonpastilha.com`); a aplicação Web; e por fim a aplicação móvel. A comunicação entre eles está representada na Figura 53 em que a forma de comunicação consiste em métodos do HTTP (Get, Post, Put, Delete), para que as aplicações do cliente possam fazer pedidos de autenticação ao servidor e poderem assim ter acesso às suas contas.

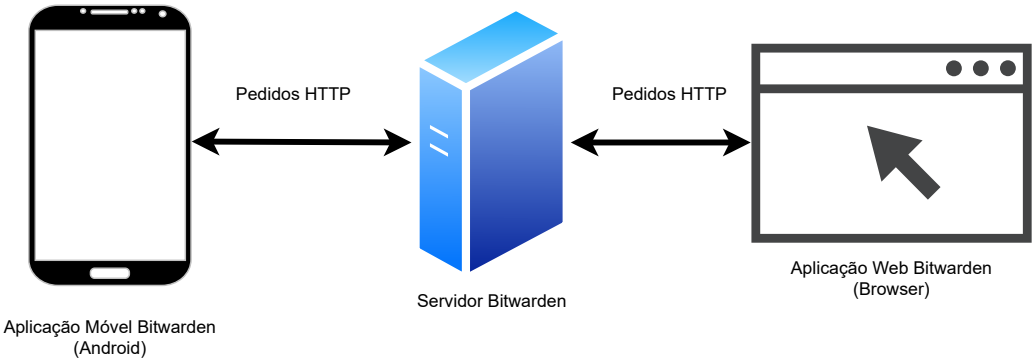

Figura 53: Arquitetura geral do Bitwarden.

As duas sub secções que se seguem contêm a arquitetura do servidor e a arquitetura do Android respetivamente. A arquitetura da aplicação Web faz parte do servidor por isso será abordada na mesma sub secção.

6.4.1 *Arquiteture do Servidor*

Na Arquitetura do Servidor está incluída a aplicação Web (*Front-End*) e a lógica de negócio como API (*Back-End*), quando o utilizador inicia a comunicação com o servidor esse envia aplicação Web para poder interagir com API, numa forma simples e graficamente apelativa. A aplicação Web é desenvolvida em Angular que permite a criação de aplicações *single-page*. Estas páginas são reescritas dinamicamente, em vez de ter de carregar novas páginas inteiras do servidor, mas que aos olhos do utilizador é uma interface com múltiplas páginas.

De seguida será abordado o funcionamento do servidor. O servidor utiliza o sistema Docker (Opensource, 2021), que é um sistema de virtualização ao nível do sistema operativo. O Docker foi desenhado para criar, submeter e correr de forma rápida e mais simples possível, através de contentores (*containers*) que permitem correr as aplicações desenvolvidas e as respetivas dependências de forma isolada do sistema operativo hospedeiro. Os contentores são de pequenas dimensões, pois em vez de usar um sistema operativo completo, o Docker permite que aplicação do contentor tenha acesso ao mesmo kernel Linux que o do sistema operativo da máquina hospedeira. O projeto Bitwarden tem um total de 12 contentores, cada um com uma funcionalidade específica:

- "bitwarden-nginx";
- "bitwarden-identity";
- "bitwarden-api";
- "bitwarden-web";
- "bitwarden-mssql";
- "bitwarden-portal";
- "bitwarden-events";
- "bitwarden-sso";
- "bitwarden-admin";
- "bitwarden-icons";
- "bitwarden-attachments";
- "bitwarden-notifications".

Os principais contentores da arquitetura do servidor são: "bitwarden-nginx", "bitwarden-identity", "bitwarden-api" e "bitwarden-web". O contentor "bitwarden-nginx" tem como prepósito reencaminhar os pedidos para os contentores correctos, como exemplo um pedido ao URL "`eltonpastilha.com/api/two-facto`", em que este é redireccionado para o contentor "bitwarden-api". O contentor "bitwarden-api" é o serviço de API do servidor, ou seja, é onde está a lógica de negócio da aplicação, esta API faz uso da estrutura de dados chamada JSON. No contentor "bitwarden-identity", este recebe pedidos HTTP como o contentor "bitwarden-api", só que este serve para autenticação do utilizador, onde será enviado as credenciais e retornado

o *token* de autenticação. Por fim, o contentor "bitwarden-web" tem a aplicação Web que interage com o utilizador quando este pede pela primeira vez numa sessão.

Para que os contentores possam comunicar entre si e com o cliente têm de estar ligados a uma rede. O Docker permite criar redes virtuais para controlar os acessos à Internet e entre contentores. Neste caso o Bitwarden contem duas redes virtuais, a rede virtual "docker_public" que tem acesso à Internet e a rede virtual "docker_default" que é local, possibilitando somente a comunicação entre os contentores. Para os contentores comunicarem entre si, usam um serviço de Domain Name System (DNS) local do Docker em que o nome de cada contentor é definido aquando da sua criação. Como se pode visualizar na Listagem 1, o exemplo da imagem do contentor "bitwarden-api" na linha 4 é definido o nome de DNS deste contentor. A ligação de rede entre os diversos contentores está representada na Figura 54.

Listagem 1: Exemplo das definições do construção do Docker.

```
1         services:
2             api:
3                 image: destruidor/api:1.39.4
4                 container_name: bitwarden-api
5                 restart: always
6                 volumes:
7                 - ../core:/etc/bitwarden/core
8                 - ../ca-certificates:/etc/bitwarden/ca-certificates
9                 - ../logs/api:/etc/bitwarden/logs
10                env_file:
11                - global.env
12                - ../env/uid.env
13                - ../env/global.override.env
14                networks:
15                - default
16                - public
```

Para o "bitwarden-nginx" estar a redirecionar um pedido da Internet, é necessário configurar um redirecionamento de porto no Docker, que no caso deste projeto é da porta 80 do servidor para a porta 8080 do "bitwarden-nginx" e da porta 443 do servidor para a porta 8443 do "bitwarden-nginx". Isto é possível pois este contentor está ligado à rede que tem acesso à Internet, os contentores que não têm redirecionamento de porto ficam isolados da Internet, como é o caso do "bitwarden-attachments", "bitwarden-mssql" e "bitwarden-web". A Figura 55 mostra os redirecionamentos de portos obtidos através do comando `dockerps`. Esta configuração é realizada nas definições do contentor através da opção "ports". A Listagem 2 tem um exemplo dessa configuração nas linhas 11, 12 e 13. Todas as configurações dos contentores podem ser visualizadas em "`/server/util/Setup/Templates/DockerCompose.hbs`".

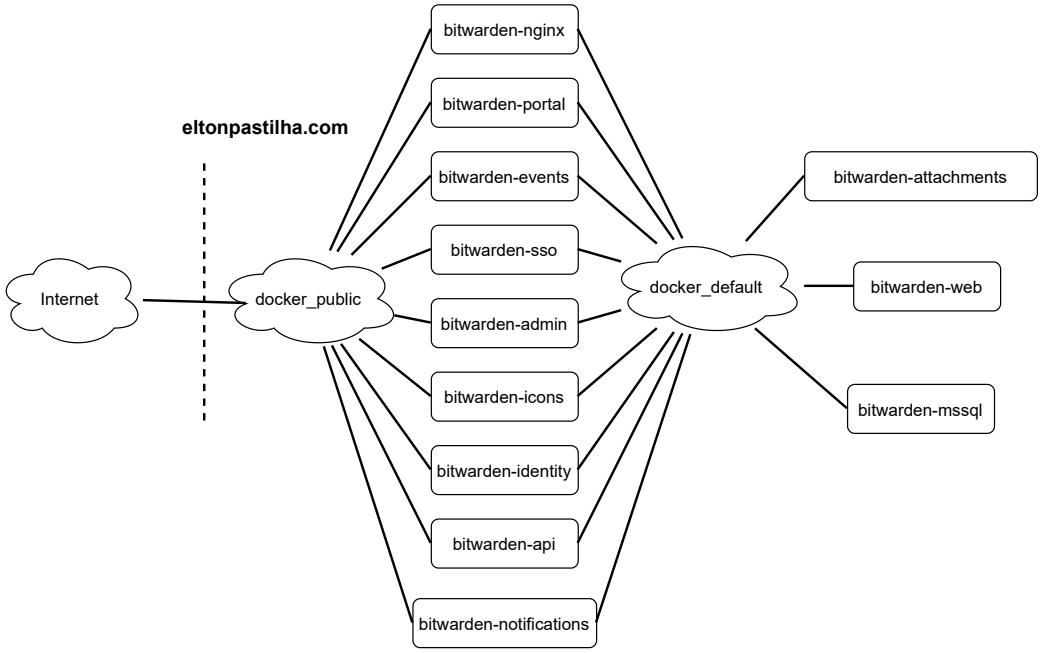

Figura 54: Arquitetura da rede virtual do Docker.

```
CONTAINER ID   IMAGE                                COMMAND                  CREATED       STATUS                    PORTS                                                 NAMES
367333c2d2f4   bitwarden/nginx:1.39.4               "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      80/tcp, 0.0.0.0:80->8080/tcp, 0.0.0.0:443->8443/tcp   bitwarden-nginx
30b5bca18a65   bitwarden/portal:1.39.4              "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-portal
02de209283e4   bitwarden/admin:1.39.4               "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-admin
c40312d61911   destruidor/web:1.39.4                "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)                                                            bitwarden-web
09699800ea43   destruidor/api:1.39.4                "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-api
98495a328a43   bitwarden/events:1.39.4              "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-events
c787fd2a44c4   bitwarden/icons:1.39.4               "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-icons
d453606127b9   bitwarden/notifications:1.39.4       "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-notifications
2d027d8604d1   bitwarden/sso:1.39.4                 "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-sso
ebc7f42f4d98   destruidor/identity:1.39.4           "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)      5000/tcp                                              bitwarden-identity
ab4ebf79f5c3   bitwarden/attachments:1.39.4         "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)                                                            bitwarden-attachments
8ba9c438f075   destruidor/mssql:1.39.4              "/entrypoint.sh"         2 weeks ago   Up 2 weeks (healthy)                                                            bitwarden-mssql
```

Figura 55: Resultado do comando “docker ps”.

Listagem 2: Definições do contentor para o Docker.

```
1         services:
2             nginx:
3                 image: bitwarden/nginx:{{{CoreVersion}}}
4                 container_name: bitwarden-nginx
5                 restart: always
6                 depends_on:
7                 - web
8                 - admin
9                 - api
10                - identity
11                ports:
12                - '80:8080'
13                - '433:8443'
14                volumes:
15                - ../nginx:/etc/bitwarden/nginx
16                - ../letsencrypt:/etc/letsencrypt
17                - ../ssl:/etc/ssl
18                - ../logs/nginx:/var/log/nginx
19                env_file:
20                - ../env/uid.env
21                networks:
22                - default
23                - public
```

Tabela 6: Tabela de redirecionamentos do “bitwarden-nginx”.

Endereço de entrada	**Redirecionamento**	**Conteiner**
/alive	retorna código 200 e com o texto “alive”	
/	web:5000/	“bitwarden-web”
/app-id.json	web:5000/app-id.json	“bitwarden-web”
/.well-known/assetlinks.json	web:5000/assetlinks.json	“bitwarden-web”
/duo-connector.html	web:5000/duo-connector.html	“bitwarden-web”
/u2f-connector.html	web:5000/u2f-connector.html	“bitwarden-web”
/sso-connector.html	web:5000/sso-connector.html	“bitwarden-web”
/attachments/	attachments:5000/	“bitwarden-attachments”
/api/	api:5000/	“bitwarden-api”
/icons/	icons:5000/	“bitwarden-icons”
/notifications/	notifications:5000/	“bitwarden-notifications”
/notifications/hub	notifications:5000/hub	“bitwarden-notifications”
/events/	events:5000/	“bitwarden-events”
/sso	sso:5000	“bitwarden-sso”
/identity	identity:5000	“bitwarden-identity”
/admin	admin:5000	“bitwarden-api”
/portal	portal:5000	“bitwarden-portal”

Assim sendo falta mostrar os redirecionamentos do “bitwarden-nginx” para os outros contentores que se encontram descritas na Tabela 6.

Passando à parte do código, na Figura 56 pode-se ver como são tratados os pacotes de autenticação e registo de uma nova chave FIDO2 usando a “API”. Quando o utilizador envia a submissão de autenticação (*username* e *password*) este entra num processo no “bitwarden-indentity”. Para decidir se deve ou não fazer um método de autenticação de dois factores, caso seja então feito uma recolha do token do FIDO2 (o *token* são os dados de desafio em JSON) no “Fido2TokenProvider”. Este, por sua vez, faz o pedido ao “UserService” que contém as funcionalidades do FIDO2. O “UserService” pede os dados das chaves FIDO2 que o utilizador tem na base de dados doservidor (“bitwarden-mysql”) e com esses dados pede à biblioteca “`abergs/fido2-net-lib`” que construa o desafio para o utilizador. O desafio é enviado ao utilizador, para que este devolva o desafio assinado. Depois de recebido o desafio assinado é submetido novamente à biblioteca “`abergs/fido2-net-lib`” para verificar e validar o desafio assinado.

No caso de registar novas chaves FIDO2, aí o processo já prossegue através do “bitwarden-api”. Este contentor possui um ficheiro “TwoFactorController”, que contem as funções que processam os pedidos de registo e a eliminação das chaves

FIDO2 do utilizador. Essas funções comunicam com o "UserService", para pedir um desafio ou para submeter uma chave FIDO2 com o desafio assinado. Na aplicação

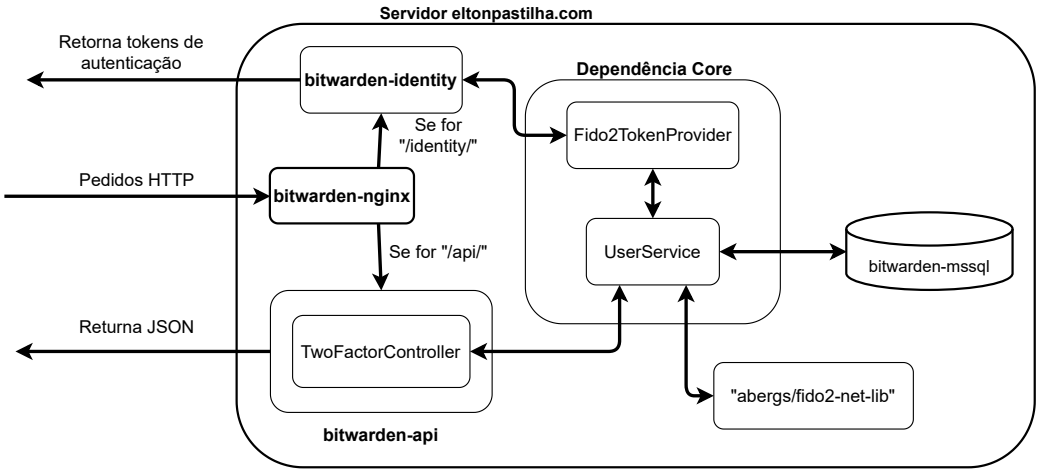

Figura 56: Arquitetura de pedido de autenticação e da API forma simples.

Web só serão mostradas as partes onde a autenticação, ou o registo de chaves FIDO2 são usadas. A arquitetura da aplicação Web está representada na Figura 57. Após o utilizador submeter as credenciais de autenticação recebe o token de autenticação pelo "AuthService", que depois é enviado para "TwoFactorComponent" onde é feito o pedido à API do "navegador", para ser assinado o desafio. Em caso de sucesso é enviado o resultado pelo "AuthService", se falhar a verificação da assinatura por alguma razão, esse desafio fica invalido, então um novo pedido será feito através "ApiService", pois o "TwoFactorComponent" tem acesso para isso. No caso de registo de uma nova chave FIDO2 será tratado pelo "TwoFactorFido2Component" onde serão feitos pedidos à "ApiService" para criar, ver e eliminar chaves FIDO2 e também para tratar dos pedidos à API do "navegador" para assinar o desafio.

6.4.2 *Arquitetura do Android*

Por fim, será abordada a arquitetura da aplicação móvel para Android, que é focada apenas na parte em que o FIDO2 aparece. A arquitetura está representada na Figura 58 e tem duas partes principais: a parte da aplicação móvel e a parte "com.google.android.gms.fido.fido2" que é a API do FIDO2 do sistema operativo Android.

Na parte da aplicação móvel estão vários componentes. O "ApiService" que é responsável por pedir novos desafios e para quando o desafio anterior falha. O "AuthService" é responsável pelo envio das credenciais, bem como o desafio assinado

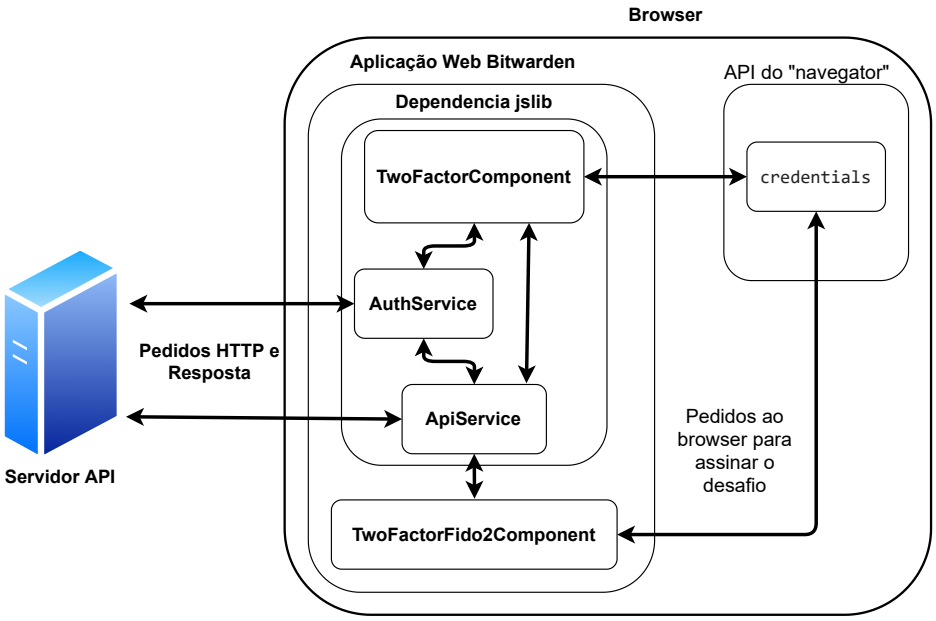

Figura 57: Arquitetura da aplicação Web só a parte do FIDO2.

para poder autenticar. O "MainActivity" é a atividade que corre por detrás do ecrã, que é o intermediario entre o ecrã do utilizador e o sistema FIDO2, ou seja, quando o utilizador pede para se autenticar usando o FIDO2, este componente pede ao "Fido2Service" para iniciar o FIDO2, para o utilizador assinar o desafio. Depois o "Fido2Service" pede ao "Fido2BuilderObject" para construir o pedido e de seguida é enviado para o "Fido2ApiClient". O "Fido2ApiClient" é responsável por mostrar o ecrã do FIDO2 para assinar o desafio. Em seguida o desafio assinado é enviado pelo "AuthService" ao servidor. Como o Android não tem o registo a funcionar na aplicação, o "ApiService" só servirá para pedir desafios para autenticar e não para registar, a razão para isso está na secção 6.10.

O "`com.google.android.gms.fido.fido2`" contém o principal componente "Fido2APIClient", onde é pedido que o Android solicite ao utilizador para usar FIDO2 para assinar o desafio. Este também contém tem o "`com.google.android.gms.fido.fido2.api.common`" onde estão as classes para serem usadas no "Fido2BuilderObject". Este é igual ao usado no Kotlin, pois o Xamarin mapeou o código do FIDO API da Google para que fosse possível ser usado no Xamarin. A diferença é que é necessário instalar o pacote "`Xamarin.GooglePlayServices.Fido`" para ter acesso.

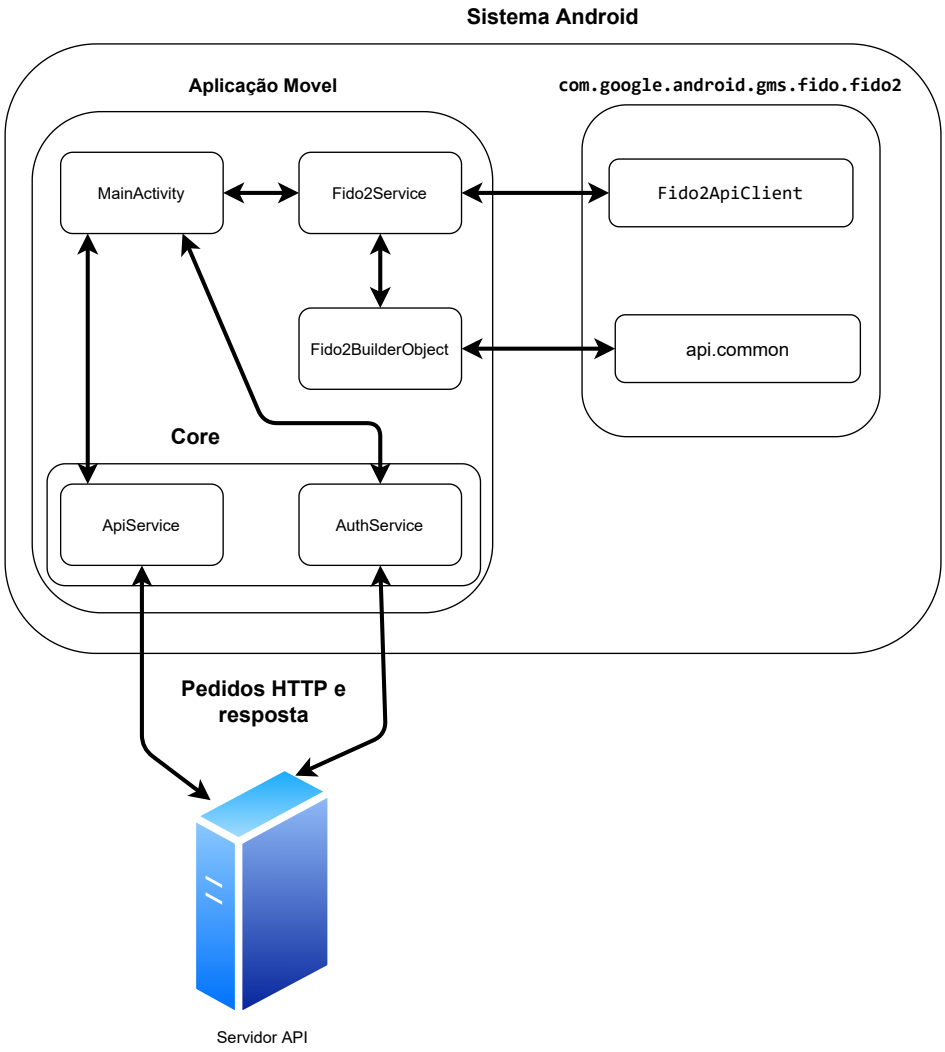

Figura 58: Arquitetura da aplicação Android (parte do FIDO2).

6.5 BASE DE DADOS

A base de dados da plataforma Bitwarden é complexa e apenas serão abordadas as tabelas criadas e que estão relacionadas com o trabalho. O Bitwarden faz uso do Microsoft Structured Query Language (SQL). Desta forma serão explicadas em primeiro lugar as tabelas criadas e de seguida serão mostradas as relações entre essas e as tabelas já existentes. As tabelas novas criadas são a "Fido2Key" e a "Fido2Challenge" cujos respetivos campos podem ser visualizadas nas Tabelas 7 e 8 respetivamente.

Tabela 7: Tabela "Fido2Key".

Nome	**Tipo**	**Explicação**
Id	UNIQUEIDENTIFIER	ID de identificação de cada registo
UserId	UNIQUEIDENTIFIER	ID de identificação do Utilizador
Name	NVARCHAR(200)	Nome da chave dado pelo utilizador
CredentialId	NVARCHAR(MAX)	Identificação da chave FIDO2
PublicKey	NVARCHAR(MAX)	Chave Pública
SignatureCounter	BIGINT	Contador de assinatura
CredentialType	TINYINT	Tipo de chave ("public-key")
AuthenticatorType	TINYINT	Tipo "Platform" ou "Cross-Platform"
Transports	NVARCHAR(200)	Transportes permitidos exemplo "NFC"
Compromised	BIT	Estado para saber se foi comprometida
CreationDate	DATETIME2(7)	Data e hora de criação

Tabela 8: Tabela "Fido2Challenge".

Nome	**Tipo**	**Explicação**
Id	INT	ID de identificação do desafio
UserId	UNIQUEIDENTIFIER	ID de identificação do Utilizador
Origin	NVARCHAR(MAX)	Origem do pedido desafio
Options	NVARCHAR(MAX)	Dados do desafio
Action	SMALLINT	Desafio tipo registar ou autenticar
CreationDate	DATETIME2(7)	Data e hora de criação
TimeOutDate	DATETIME2(7)	Tempo limite do desafio

Para além das duas tabelas referidas, foram também criadas duas vistas (*Views*), uma para cada tabela ("Fido2KeyView" e "Fido2ChallengeView"). A funcionalidade destas vistas é mostrar ou receber todos os dados da tabela. Também foram criadas

seis “PROCEDURE” para o “Fido2Key” e cinco para “Fido2Challenge”, o que cada um faz é descrito a seguir:

- **“Fido2Key_Create”** - Este tem como função receber os dados e inserí-los na tabela “Fido2Key”;
- **“Fido2Key_ReadById”** - Este tem como função ir receber um chave FIDO2 com o ID fornecido;
- **“Fido2Key_ReadByUserId”** - Neste são recebidas todas as chaves FIDO2 registadas que pertencem ao ID do utilizador fornecido;
- **“Fido2Key_UpdateSignatureCounter”** - Para atualizar o contador de assinaturas na chave FIDO2 fornecida através do ID;
- **“Fido2Key_DeleteById”** - Eliminar uma chave FIDO2 pelo ID;
- **“Fido2Key_DeleteByUserId”** - Eliminar todas as chaves FIDO2 de um utilizador;
- **“Fido2Challenge_Create”** - Este tem como função receber os dados e inseri-los na tabela “Fido2Challenge”;
- **“Fido2Challenge_ReadLastCreatedByUserId”** - Receber o último desafio criado para um utilizador;
- **“Fido2Challenge_ReadManyNotExpired”** - Receber todos os desafios que não ultrapassaram o tempo num utilizador;
- **“Fido2Challenge_DeleteById”** - Eliminar um desafio pelo seu ID;
- **“Fido2Challenge_DeleteManyExpired”** - Eliminar 100 desafios que já expirados.

A relação entre as tabelas é uma relação de “One-To-Many”, em que cada registo tanto na tabela “Fido2Key” como na tabela “Fido2Challenge”, só pode ter registos de um só utilizador. Ou seja, cada registo só pertence a um único utilizador, mas um utilizador pode ter múltiplos registos – um utilizador pode ter mais que uma chave FIDO2 e pode ter mais que um desafio registado no seu ID. Na Figura 59 pode-se ver a relação ilustrada através do MySql Workbench.

6.6 LOCAL DE TESTES

Neste capítulo será abordado como é que foi aplicado o Bitwarden num ambiente controlado, para colocar funcionalidades desenvolvidas em produção, para executar

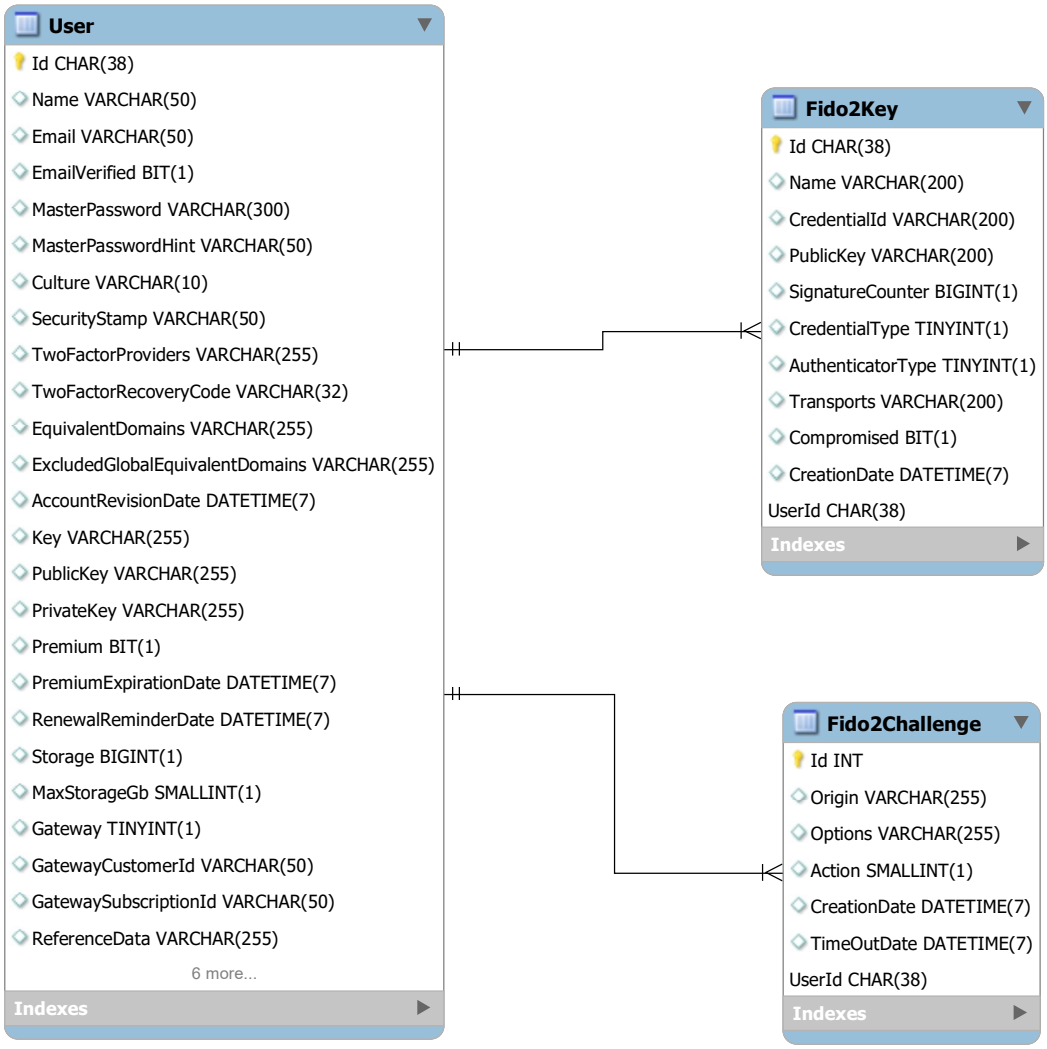

Figura 59: Arquitetura da aplicação Android só a parte do FIDO2.

testes e se as funcionalidades adicionais se executam de forma correta, eficaz e segura.

O ambiente controlado foi criado no serviço de cloud da DigitalOcean, em que permitia ter acesso a um sistema Ubuntu 20.04 (LTS x64) com 25GB de disco e 2GB de memória. Este é o segundo servidor mais barato que a DigitalOcean oferece. Foi escolhido por disponibilizar 2GB de RAM que é o requisito mínimo do SQL *Server* empregue pelo Bitwarden, e por cumprir os requisitos mínimos solicitado pelo Bitwarden (bitwarden, 2021). Depois de ter um servidor, foi decidido registar o dominio "`eltonpastilha.com`" através da NameCheap, aproveitando a oferta do certificado Secure Sockets Layer (SSL) (PositiveSSL) para o domínio. Nas configurações do domínio foi colocado o DNS da DigitalOcean para que o domínio estivesse redirecionado ou interligado com o servidor comprado na DigitalOcean.

Ao ter o servidor preparado para receber o Bitwarden, iniciou-se a instalação do mesmo, que requer o Docker Engine e o Docker Compose, pois o serviço do Bitwarden faz uso do Docker. De seguida foi criado um utilizador no sistema operativo (chamado "bitwarden"), com permissões no grupo Docker e foi criado uma pasta "`/opt/bitwarden`" para esse utilizador. É nesta pasta que vão estar as configurações do Bitwarden e o script para iniciar, atualizar ou desligar o serviço do Bitwarden.

Após ter o sistema preparado foi colocado o script `/opt/bitwarden/bitwarden.sh` (criado pelo Bitwarden) que por sua vez faz uso de outro script (`/opt/bitwarden/bwdata/run.sh`) para instalar, desinstalar ou fazer atualizações consoante a versão e o comando fornecido. Além destas tarefas ainda verifica, antes de executar o comando, se os ficheiros e as pastas necessárias existem. O script "run.sh" tem como objetivo pedir à biblioteca do Docker o `destruidor/setup`, que contém as configurações da sub-network do Docker e o nome dos outros contentores necessários.

Na instalação, recorrendo ao comando "/opt/bitwarden/bitwarden.sh install", são pedidos os dados do servidor, como o nome do domínio, o certificado SSL e ainda o ID e a chave de autorização por parte do Bitwarden que é gratuito (serve para poder notificar acerca de atualizações de segurança e para poder utilizar licenças pagas). Após tudo estar instalado com sucesso são realizadas algumas configurações no "`/opt/bitwarden/bwdata/env/global.override.env`":

Listagem 3: Configurações alteradas no "/opt/bitwarden/bwdata/env/global.override.env"

```
        ...
globalSettings__yubico__clientId=61--3
globalSettings__yubico__key=yb----------------------------------------=
        ...
globalSettings__mail__smtp__host=smtp.gmail.com
globalSettings__mail__smtp__port=587
globalSettings__mail__smtp__ssl=true
globalSettings__mail__smtp__username=<oculto>@gmail.com
globalSettings__mail__smtp__password=<oculto>
        ...
adminSettings__admins=<oculto>
```

Nestas configurações foram adicionadas as chaves Yubico, para usar o yubikey com o seu OTP. Foi também adicionado um serviço de e-mail, neste caso do Gmail, para o servidor enviar emails e ainda foi colocado um email para usar a parte administrativa do servidor. Depois de alteradas as configurações o serviço é reiniciado ("./bitwarden.sh restart") ou iniciado ("./bitwarden.sh start") dependendo se já está em execução ou não, para atualizar as suas configurações. Com isto o servidor ficou preparado e sempre que houver uma atualização efetuada, essa tem de ser enviada para o `hub.docker.com` para que o servidor possa então atualizar usando o comando "./bitwarden.sh update".

Para enviar as atualizações para o `hub.docker.com` é necessário ter no computador o Docker Desktop (no caso do Windows), que envia as imagens Docker para o `hub.docker.com`. Para isso é preciso criar as imagens Docker, em cada projeto e aplicação existe o ficheiro "build.sh" que compila o projeto (usando comandos "dotnet", "npm", *etc*) e constrói a imagem Docker (como exemplo "docker build -t destruidor/api "$DIR/."). No caso do servidor, o projeto na raiz também tem um "build.sh", que chama todos os "build.sh" das aplicações que contém. Neste caso só em 4 aplicações é que é chamado, porque só em 4 aplicações é que foram feitas modificações, em todos os outros não foram efetuados, pelo que não há necessidade de o fazer.

Listagem 4: "build.sh" da aplicação API do Servidor

```
#!/usr/bin/env bash
set -e

DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && "pwd" )"

echo -e "\n## Building API"

echo -e "\nBuilding app"
echo ".NET Core version $(dotnet --version)"
```

```
echo "Restore"
dotnet restore "$DIR/Api.csproj"
echo "Clean"
dotnet clean "$DIR/Api.csproj" -c "Release" -o "$DIR/obj/Docker/publish/Api"
echo "Publish"
dotnet publish "$DIR/Api.csproj" -c "Release" -o "$DIR/obj/Docker/publish/Api"

echo -e "\nBuilding docker image"
docker --version
docker build -t destruidor/api "$DIR/."
```

Listagem 5: “build.sh” do Servidor

```
#!/usr/bin/env bash
set -e

DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

echo ""

if [ $# -gt 1 -a "$1" == "push" ]
then
    TAG=$2

    echo "Pushing Destruidor ($TAG)"
    echo "========================"

    docker push destruidor/api:$TAG
    docker push bitwarden/identity:$TAG
    docker push bitwarden/server:$TAG
    docker push bitwarden/attachments:$TAG
    docker push bitwarden/icons:$TAG
    docker push bitwarden/notifications:$TAG
    docker push bitwarden/events:$TAG
    docker push bitwarden/admin:$TAG
    docker push bitwarden/nginx:$TAG
    docker push bitwarden/sso:$TAG
    docker push bitwarden/portal:$TAG
    docker push destruidor/mssql:$TAG
    docker push destruidor/setup:$TAG
elif [ $# -gt 1 -a "$1" == "tag" ]
then
    TAG=$2

    echo "Tagging Destruidor as '$TAG'"

    docker tag destruidor/api destruidor/api:$TAG
    docker tag destruidor/identity destruidor/identity:$TAG
    # docker tag bitwarden/server bitwarden/server:$TAG
                              ...
    # docker tag bitwarden/portal bitwarden/portal:$TAG
```

```
    docker tag destruidor/mssql destruidor/mssql:$TAG
    docker tag destruidor/setup destruidor/setup:$TAG
else
    echo "Building Bitwarden"
    echo "=================="

    chmod u+x "$DIR/src/Api/build.sh"
    "$DIR/src/Api/build.sh"

    chmod u+x "$DIR/src/Identity/build.sh"
    "$DIR/src/Identity/build.sh"

    # chmod u+x "$DIR/util/Server/build.sh"
    # "$DIR/util/Server/build.sh"
                                ...
    # chmod u+x "$DIR/bitwarden_license/src/Portal/build.sh"
    # "$DIR/bitwarden_license/src/Portal/build.sh"

    chmod u+x "$DIR/util/MsSql/build.sh"
    "$DIR/util/MsSql/build.sh"

    chmod u+x "$DIR/util/Setup/build.sh"
    "$DIR/util/Setup/build.sh"
fi
```

6.7 ALTERAÇÕES EFETUADAS NO SERVIDOR

Como foi possível visualizar no capítulo da arquitetura, o projeto "`bitwarden/server`" contêm múltiplas mini aplicações. Nesta secção serão abordadas as alterações que foram efetuadas para colocar o FIDO2 a funcionar no servidor. Foi necessário modificar 3 mini aplicações: "Setup", "SQL" e "API". Além destas, foram ainda feitas alterações na dependência "Core" que é compilado em conjunto nos projetos "Identity" e "API".

Começando pelo "Setup", foram feitas 2 alterações e criados 2 novos ficheiros. O propósito destes é criar e gerir os "assetLinks.json", que o Android necessita para verificação. As modificações da aplicação foram:

- Criação do ficheiro "server/util/Setup/Templates/AssetLinks.hts" que é um template do "assetLinks.json", que será colocado em "`/opt/bitwarden/bwdata/web`" no servidor, para poder atualizar "`eltonpastilha.com/.well-known/assetlinks.json`" sem ter de modificar código ou de compilar imagem Docker quando necessário;

- Criação do ficheiro "server/util/Setup/AssetLinksBuilder.cs" que visa sincronizar o "`/opt/bitwarden/bwdata/web/assetlinks.json`" com o "`web/assetlinks.json`" que está na aplicação ou na imagem Docker da Web. É realizado este procedimento na instalação e na atualização;
- Alteração no ficheiro "`server/util/Setup/Program.cs`" em que a alteração feita foi adicionar o "AssetLinksBuilder.cs" a lista de tarefas a fazer na instalação e na atualização.
- Alteração no ficheiro "`server/util/Setup/Templates/NginxConfig.hbs`", este ficheiro contém o template das configurações da imagem Nginx Docker. Este local contém as diretivas específicas do servidor e foi adicionado mais uma diretiva que redireciona o pedido por "`eltonpastilha.com/.well-known/assetlinks.json`" para o ficheiro dentro da aplicação Web no local "`web/assetlinks.json`" e devolve o pedido como tipo de conteudo JSON, como pode ver na listagem 6.

Listagem 6: Diretiva adicionada a configuração do Nginx

```
...
location = /.well-known/assetlinks.json {
    proxy_pass http://web:5000/assetlinks.json;
{{#if Ssl}}
    include /etc/nginx/security-headers-ssl.conf;
{{/if}}
    include /etc/nginx/security-headers.conf;
    proxy_hide_header Content-Type;
    add_header Content-Type application/json;
}
...
```

Na mini aplicação "SQL" foram feitas alterações para criar as "tables", as "views" e os "procedures" para que a informação do FIDO2 pudesse ser guardada. No Apêncice A poderá visualizar os ficheiros criados.

Na mini aplicação "API" foi feita uma única alteração no ficheiro "`server/src/Api/Controllers/TwoFactorController.cs`". Esta visa permitir receber os pedido HTTP para receber e devolver informações do FIDO2, em que foram registadas 5 novas rotas só para o FIDO2. As rotas são as seguintes:

- **Rota "`/api/two-factor/get-fido2`"** - Esta rota recebe pedidos POST, para pedir as chaves FIDO2 que o utilizador contém. A razão de ser POST e não GET, é porque o Bitwarden requerer que se mande a *password* mestre (mesmo que utilizador esteja autenticado) quando se trata de informações sensíveis;

- **Rota "`/api/two-factor/get-fido2-regist-challenge`"** - Esta rota recebe pedidos POST para permitir ao utilizador receber um "challenge" para assinar e assim poder registar uma nova chave FIDO2 à sua conta. Tal como na rota anterior é necessário receber também a *password* mestre.
- **Rota "`/api/two-factor/fido2-regist`"** - Esta rota recebe pedidos POST, que possibilitam registar uma chave nova FIDO2, desde que tenha o "challenge" pedido na rota "/api/two-factor/get-fido2-regist-challenge" assinado com sucesso.
- **Rota "`/api/two-factor/get-fido2-auth-challenge`"** - Esta rota recebe pedidos POST e com possibilidade de ser anónimo, ou seja, sem estar autenticado. Esta rota serve para quando o utilizador não está autenticado e necessita de uma validação FIDO2 de dois factores, em que é gerado o "challenge" para ser assinado. Nesta rota é necessário receber o *username* e a *password*.
- **Rota "`/api/two-factor/fido2`"** - Esta rota recebe pedidos DELETE, para que utilizador possa eliminar chaves FIDO2 da sua conta de utilizador. Também aqui é necessário receber a *password* mestre.

Em cada uma das rotas são realizadas as seguintes verificações: permissões, se a *password* mestre está correta e se o utilizador existe. De seguida, é enviado o pedido para o mini-projeto "Core" que contém o "UserService" com código para tratar os pedidos do FIDO2.

Na dependência "Core" foram efetuadas várias alterações, as mais importantes são as seguintes:

- **Ficheiro "Fido2TokenProvider"** (novo) - Este ficheiro está interligado com a autenticação de dois factores. Assim quando o utilizador se autentica e o servidor decide que o utilizador deve autenticar usando o sistema de dois factores, este ficheiro é utilizado para gerar a informação FIDO2, que contém o *challenge* em modo JSON formando assim um token. Quando o utilizador assina o *challenge*, este é enviado também em JSON para o servidor. Este ficheiro é responsável pela verificação do desafio e onde é decidido se o utilizador passou ou não.
- **Ficheiro "UserService"** (alterado) - trata toda informação do FIDO2 entre a plataforma e a biblioteca "`abergs/fido2-net-lib`", nas seguintes funções:
  - **Função "StartFido2RegistrationAsync"** - Inicia o registo de uma nova chave FIDO2, através de requerimento do *challenge* para registar nova chave FIDO2;

- **Função "CompleteFido2RegistrationAsync"** - Valida assinatura do *challenge* e a chave FIDO2 enviada, para registar uma nova chave FIDO2 ao utilizador;
- **Função "StartFido2AuthenticationAsync"** - Inicia a autenticação através do uso do FIDO2, em que é devolvido ao utilizador o *challenge* para assinar, mas este é do tipo autenticação;
- **Função "CompleteFido2AuthenticationAsync"** - Valida o *challenge* assinado para autenticação, usando uma das chaves públicas do FIDO2 registadas nesse mesmo utilizador.

As quatro funções referidas acima contêm código utilizado para chamar a biblioteca "`abergs/fido2-net-lib`". Primeiro é inicializado o serviço FIDO2 através do código presente na Listagem 7. Este recebe o domínio, o nome da plataforma apresentar ao utilizador, o tempo máximo para assinar o *challenge* e a *origin*. A origem do pedido pode ser HTTP ou a origem do pedido com o *challenge* assinado. Se for pedido um *challenge*, a origem desse pedido é guardado na base de dados juntamente com o *challenge* assinado. Se for um pedido a enviar com o *challenge* assinado, então é usada a origem guardada na base de dados, para assim se comparar com a origem do pedido inicial.

Listagem 7: Inicializar FIDO2 da biblioteca

```
1         var fido2Lib = new Fido2(new Fido2Configuration()
2         {
3             ServerDomain = _globalSettings.BaseServiceUri.Domain, //
              ↪  "bitwarden.com"
4             ServerName = _globalSettings.SiteName, // "Bitwarden"
5             Origin = origin, // "bitwarden.com"
6             TimestampDriftTolerance = MaxFido2Time // 36000
7         });
```

No código da Listagem 8 é possível receber a informação e o *challenge* para o utilizador registar uma chave nova. Para isso é necessário indicar o ID do utilizador e o nome para o utilizador confirmar que está a registar na conta certa. Também é necessário colocar a lista de IDs das chaves FIDO2 já registadas, para o FIDO2 cliente não deixar enviar uma chave FIDO2 já registada. Por fim, é adicionada uma das opções como tipo de chave, "Platform" ou "Cross-Platform", a ser registada.

Listagem 8: Criar *challenge* para registar nova chave FIDO2

```
var response = fido2Lib.RequestNewCredential(
    userFido2,  // Informações do utilizador, ID do utilizador e Nome.
    existingCredentials.ToList(), // lista de chaves já registadas.
    authenticatorSelected, // desisão do utilizador se quer chave do
    ↪  tipo "Platform" ou "Cross-Platform".
    AttestationConveyancePreference.None
);
```

No código da Listagem 9 está a função responsável por validar se o *challenge* foi bem assinado e devolve os dados da chave FIDO2, como exemplo chave pública, contador de assinaturas e o ID da chave pública. Esta função recebe os dados enviados pelo FIDO2 cliente, recebe os dados enviados no código anterior e que foram guardados previamente na base de dados, para assim poder verificar se o *challenge* assinado é o mesmo, se a origem é a mesma e ainda recebe uma função (que é construída como no código da Listagem 10), em que verifica se a chave FIDO2 não está registada ou não.

Listagem 9: Verificar assinatura do *challenge* e a nova chave FIDO2

```
var resp = await fido2Lib.MakeNewCredentialAsync(
    authenticatorAttestationRaw, // Resposta do utilizador.
    options, // Informações do challenge enviado ao utilizador .
    callback // Função em que valida se o ID da chave FIDO2 não existe.
);
```

Listagem 10: Verificar assinatura do *challenge* e a nova chave FIDO2

```
IsCredentialIdUniqueToUserAsyncDelegate callback = async
↪  (IsCredentialIdUniqueToUserParams args) =>
{
    // Verificar em todas as chaves se o ID é igual.
    if (keys.Any(key => key.CredentialId.Equals(args.CredentialId)))
    {
        // Se for igual já está registrada essa chave FIDO2.
        return false;
    }
    return true;
};
```

No código da Listagem 11 é possível receber a informação e o *challenge* para o utilizador autenticado usando FIDO2. É necessário colocar a lista de IDs das chaves FIDO2 já registadas, para o FIDO2 cliente não deixar enviar uma resposta de uma assinatura efetuada com uma chave FIDO2 não registada. É ainda colocado uma opção que indica ao cliente FIDO2 que não há necessidade de fazer o utilizador verificar os dados (nome do utilizador e nome da plataforma) antes de assinar.

Listagem 11: Criar *challenge* para registar nova chave FIDO2

```
var response = fido2Lib.GetAssertionOptions(
    existingCredentials.ToList(), // lista de chaves já registadas.
    UserVerificationRequirement.Discouraged // Não há necessidade de
    ↪ pedir ao utilizador para verificar antes de assinar.
);
```

O código da Listagem 12 contém a função que valida se o *challenge* foi bem assinado. Esta função recebe os dados enviados pelo FIDO2 cliente, os dados enviados no código anterior (que foram guardados previamente na base de dados), para assim poder verificar se o *challenge* assinado e a origem são os mesmos. Recebe também a chave pública da chave FIDO2 selecionada, o contador de assinatura da chave FIDO2 e ainda uma função que é construída como apresentado na Listagem 13. Este código valida se a chave FIDO2 existe no utilizador e se não está comprometida.

Listagem 12: Verificar assinatura do *challenge* e a nova chave FIDO2

```
var resp = await fido2Lib.MakeAssertionAsync(
    authenticatorAssertionRawResponse,  // Resposta do utilizador.
    options,  // Informações do challenge enviado ao utilizador .
    publicKey, // Chave publica da chave FIDO2 selecionada.
    (uint)chosenKey.SignatureCounter,  //Contador de assinatura da chave
    ↪ FIDO2 selecionada.
    callback // Função em que valida se a chave FIDO2 pertence ao
    ↪ utilizador e se não está comprometida.
);
```

Listagem 13: Verificar assinatura do *challenge* e a nova chave FIDO2

```
IsUserHandleOwnerOfCredentialIdAsync callback = async (args) =>
{
    // valida se a chave FIDO2 pertence ao utilizador e se não está
    ↪ comprometida.
    return chosenKey.UserId.Equals(user.Id) && !chosenKey.Compromised;
};
```

6.8 ALTERAÇÕES EFETUADAS NO WEB

O projeto Web é uma mini aplicação que corre em conjunto com as outras mini aplicações do projeto *Server*. No entanto, está num projeto separado por usar outro tipo de linguagem que neste caso é Angular, que é uma *Framework* de Javascript. No Javascript não é necessário instalar nenhuma biblioteca adicional, pois o FIDO2 já tem suporte nativo nesta linguagem. Assim, é possível invocar

através dos *browsers* a autenticação com Windows Hello ou com Yubikey. O projeto Web trabalha em conjunto com o projeto "`bitwarden/jslib`", que se encontra no caminho "`web/jslib`". Este contém a parte da autenticação e é igual para todas as mini aplicações que usam Angular, como exemplo o plugin do chrome do Bitwarden.

O aspeto mais importante neste projeto é uso do FIDO2 no Angular, que é muito similar à utilização no Javascript normal. Para pedir a assinatura à API do cliente FIDO2, basta o código presente nas Listagens 14 e 15, desde que já tenha o *challenge* do servidor.

Listagem 14: Assinar um *challenge* para registar uma nova chave FIDO2.

```
1         const cred = await navigator.credentials.create({
2             publicKey: registFido2Challenge,
3         });
```

Listagem 15: Assinar um *challenge* para autenticação.

```
1         var cred = await navigator.credentials.get({
2             publicKey: authFido2Challenge,
3         });
```

Outra alteração importante foi a colocação de "assetlinks.json" na pasta "`web/src/`". Para garantir o acesso por URL a "`bitwarden.com/.well-known/assetlinks.json`". Isto é especialmente importante no Android porque é mandatório e contém informações como o domínio e a hash da aplicação Android.

Listagem 16: AssetLinks.json

```
[
    {
        "relation":[
            "delegate_permission/common.handle_all_urls",
            "delegate_permission/common.get_login_creds"
        ],
        "target":{
            "namespace":"web",
            "site":"https://bitwarden.com"
        }
    },
    {
        "relation":[
            "delegate_permission/common.handle_all_urls",
            "delegate_permission/common.get_login_creds"
        ],
        "target":{
            "namespace":"android_app",
            "package_name":"com.x8bit.bitwarden",
            "md5_cert_fingerprints":[
                "35:E6:B6:4F:53:2F:58:EA:4B:E7:0B:D8:2D:0D:52:53"
            ],"sha1_cert_fingerprints":[
↪   "3A:29:56:60:D4:38:7C:01:C8:BC:BA:C0:6E:57:8A:6F:82:A5:3B:7D"
            ],"sha256_cert_fingerprints":[
                "2C:1A:66:1E:58:EA:C0:92:4B:75:75:4D:A6:37:DD:03:3D:A4:D
↪   8:C3:68:8D:C4:85:65:92:2E:46:3D:41:E3:02"
            ]

        }
    }
]
```

6.9 ALTERAÇÕES EFETUADAS NO ANDROID

No projeto *Mobile* as alterações ocorreram no projeto Android. As alterações feitas à biblioteca "Core" também provocam alterações na compilação do iOS. No entanto, o FIDO2 está desativado para o iOS e colocado como restrição, não tendo sido possível averiguar se este trabalho impacta ou não o iOS. No caso do Android, as alterações foram ativadas e funcionaram nos testes realizados. As alterações efetuadas no "Core" foram ao nível de comunicação entre o servidor e o Android. As principais alterações encontram-se na pasta "`android/src/Android/Fido2System/`", que contém 3 ficheiros, o "Fido2Service.cs", o "Fido2BuilderObject.cs" e o "Fido2BuilderDictionary.cs".

O ficheiro "Fido2Service.cs" tem 3 responsabilidades, a primeira é construir os dados do FIDO2 usando "Fido2BuilderObject.cs" ou "Fido2BuilderDictionary.cs". A segundo é mandar o FIDO2 cliente executar e receber o resultado. Por último, verificar se o resultado tem erros e que tipo de erro é. Caso não haja erro envia a informação.

Os ficheiros “Fido2BuilderObject.cs” e o “Fido2BuilderDictionary.cs” são semelhantes, a diferença entre eles é que um recebe os dados do servidor como uma classe e o outro recebe como um dicionário. No entanto, ambos têm o mesmo objetivo de construir o pedido para FIDO2 cliente usando os dados do servidor.

Antes de fazer os pedidos ao FIDO2 cliente, é necessário inicializar através do código na Listagem 7. Isto se estiver no lado do Android o “Xamarin.GooglePlayServices.Fido” instalado.

Listagem 17: Inicializar o FIDO2 no Xamarin.

```
1         this.fido2ApiClient = Fido.GetFido2ApiClient(this.application);
```

Após inicialização, é necessário receber o desafio do servidor para autenticar, sendo necessário construir o pedido usando o “Fido2BuilderObject” com a função “ParsePublicKeyCredentialRequestOptions”. Depois de construído, o pedido é enviado para a função “GetSignPendingIntent” do “fido2ApiClient”. Assim, é efetuado um pedido ao FIDO2 cliente para criar uma “Intenção Pendente” para ser executado, como se pode ver no código da Listagem 18.

Listagem 18: Assinar um *challenge* para autenticação usando Xamarin.

```
1         var task = this.fido2ApiClient.GetSignPendingIntent(Fido2BuilderObject.P⌋
          ↪  arsePublicKeyCredentialRequestOptions(dataObject));
2
3         task.AddOnSuccessListener((IOnSuccessListener)this.application)
4                 .AddOnFailureListener((IOnFailureListener)this.application)
5                     .AddOnCompleteListener((IOnCompleteListener)this.application⌋
                      ↪  );
```

Quando a tarefa for executada com sucesso é executado a função “OnSuccess” da actividade principal do ecrã. Nesta função é recebida a “Intenção Pendente” a ser executada na função “StartIntentSenderForResult” da actividade principal do ecrã, que neste caso é para autenticar, como se pode visualizar no código da Listagem 19.

Listagem 19: Função “OnSuccess”.

```
public void OnSuccess(Java.Lang.Object result)
{
    // Verificar se o sucesso deste evento é do FIDO2.
    if (result != null && Enum.IsDefined(typeof(Fido2CodesTypes),
    ↪ this.fido2CodesType))
    {
        try
        {
            // Executar o pedido de autenticação usando FIDO2.
            this.application.StartIntentSenderForResult(
                ((PendingIntent)result).IntentSender, // O pedido de
                ↪ autenticação usando FIDO2
                (int)this.fido2CodesType, // Neste caso é
                ↪ RequestSignInUser = 994
                null, 0, 0, 0);
        }
        catch (Exception e)
        {
            // Caso de erro mostra mensagem
            this._platformUtilsService.ShowDialogAsync(this._i18nService
            ↪ .T("Fido2SomethingWentWrong"),
            ↪ _tag_for_user);
        }
    }
}
```

Quando é utilizado o FIDO2 cliente para assinar o desafio é executada a função “OnActivityResult”. Esta função espera até que o código “RequestSignInUser” seja igual a "994” (código de autenticação, com valor 994 definido na classe de enumeração “Fido2CodesTypes”) e quando tal acontece é verificado se houve erro a usar o FIDO2 (“GetByteArrayExtra(Fido.Fido2KeyErrorExtra)”). Caso tenha sucesso é enviado para o servidor toda a informação do FIDO2, incluindo o *challenge* assinado.

Listagem 20: Assinar um *challenge* para autenticação.

```
// Tentar ir buscar os erros.
byte[] errorExtra = data.GetByteArrayExtra(Fido.Fido2KeyErrorExtra);
if (errorExtra != null) // Se existir quer dizer que deu erro.
{
    // Mostrar uma mensagem de erro ao utilizador.
    this.HandleErroCode(errorExtra);
}
else
{
    // Verificar se tem dados para enviar
    if (data != null)
    {
        // Enviar os dados para o servidor, para poder autenticar.
        this.SignInUserResponse(data);
    }
}
break;
```

6.10 DIFICULDADES E DECISÕES

Neste projeto houve várias dificuldades, desde da compreensão do código desenvolvido pela equipa do Bitwarden, bem como a aplicação do FIDO2 no Android, na Web e no servidor. As dificuldades foram sendo ultrapassadas e com isso foi possível implementar o FIDO2 no Bitwarden. No entanto, algumas das soluções empregues poderão não ser as melhores devido às limitações impostas pelo código já existente.

Uma das maiores dificuldades foi a falta de documentação, agravada pela falta de comentários no código. Isto obrigou a um grande investimento de tempo no estudo do funcionamento da aplicação. Para ajudar a ultrapassar esta dificuldade foi usada a função de pesquisa do Visual Studio Code. Esta ferramenta permite pesquisas mais avançadas que a do Visual Studio 2019 que se revelou demasiado simples para ser útil neste trabalho. Com o Visual Studio Code foi possível pesquisar onde eram usadas palavras-chaves, como exemplo “Yubikey”, “email”, “auth” e “twofactor”. Assim foi possível identificar em que ficheiros e funções apareciam e porque é que era usado em cada ponto do código. As palavras-chaves “Yubikey” e “email” foram de grande utilidade pois já estavam desenvolvidos na plataforma estes dois métodos de autenticação de dois factores. Outra das dificuldades foi a falta de documentação relativo ao FIDO2 no Xamarin. Verificou-se que a informação sobre o FIDO2 no Xamarin é praticamente inexistente. Esta dificuldade foi ultrapassada por persistência e muitos testes.

No lado do Android, existe uma dificuldade que foi ultrapassada de forma pouco convencional. Para pedir o desafio, o cliente deve enviar a sua origem, mas como o projeto *Mobile* é desenvolvido para iOS e Android, existe código em comum, que está numa zona neutra. Isto significa que as funcionalidades específicas de cada sistema operativo não estão acessíveis nesta zona. A parte responsável por construir um pedido HTTP e os “Header” estão na zona neutra. Isto impede que se possa pedir o envio do hash através do "header"da aplicação Android, ou do iOS. A solução encontrada para contornar este problema foi escrever manualmente o valor. Isto não é a forma ideal de resolver o problema porque não é dinâmica.

No lado do servidor foram encontrados duas dificuldades. A primeira é relativa à forma de guardar as informações do FIDO2 (chaves públicas, ID das chaves e entre outros dados) num só campo na base de dados, em formato JSON. Embora esse campo possa aramazenar até 2GB (NVARCHAR(MAX)), a transformação dos dados em JSON de um utilizador com muitos métodos de autenticação, ou se surgirem erros na transformação do JSON, ficará comprometida em termos de

desempenho. Por esta razão decidiu-se que este campo só iria ser usado para marcar se a autenticação FIDO2 de dois factores está ativo ou desativo. Para guardar as chaves FIDO2 foi criada numa tabela dedicada para este fim.

A outra dificuldade foi em relação ao valor da origem, na biblioteca "`abergs/fido2-net-lib`". Esta foi desenvolvida a pensar apenas no serviço Web e não para o sistema operativo Android. Desta forma, só suporta uma origem de cada vez (a origem permite impedir os ataques de *Phishing*). A solução encontrada foi declarar a biblioteca todas as vezes que surge um pedido para usar a origem do Android e a origem da Web. Com esta abordagem perde-se a proteção contra ataques de *Phishing*. Compromete a verificação do lado do servidor, mas não quer dizer que utilizador vai estar totalmente exposto ao *Phishing*, porque o FIDO2 do cliente também pode verificar se a origem pertence ao domínio dado no pacotes do FIDO2 (este domínio não pode ser alterado pela entidade maliciosa, pois não terá sucesso na autenticação). No caso do Windows Hello é verificada a origem antes de assinar, mas não quer dizer que outros sistemas façam o mesmo. Não foi colocado nenhum parâmetro ou função para impedir os ataques de *Phishing* neste trabalho, pois esta situação já foi reportada no Github da biblioteca. Por esta razão foi decidido esperar pela solução dos autores da biblioteca. No entanto, este problema seria resolvido com um *IF*, em que verificava se era uma origem fidedigna, mas foi decidido esperar pela actualização. Isto demonstra também, se uma aplicação ou uma biblioteca não foi preparada para todas as situações que pode ser usado, pode prejudicar os utilizadores e a plataforma. Pois se um programador que implementar a funcionalidade, não tiver um bom conhecimento do funcionamento do FIDO2, pode comprometer a segurança da autenticação.

Existe ainda outro problema, a biblioteca não permite usar mais que uma origem, na autenticação de dois factores, quando é autenticado e devolvido todos os token de dois factores. Assim, no código onde são devolvidos esses tokens, não foi possível encontrar uma maneira para ter acesso aos "Headers" do pedido para ir buscar a origem. Então foi decidido que a origem era sempre o da Web. Caso o Android queira autenticar, tem de fazer um pedido HTTP à API onde é possível recolher a origem do Android. Isto acontece também quando o utilizador pede um novo desafio, em que por alguma razão o desafio anterior passou do tempo limite ou a assinatura do desafio era inválida. Nestes casos têm se de pedir um novo desafio através da API.

Foi reportado o erro de não poder usar duas origens no projeto "`abergs/fido2-net-lib`". Esta situação foi marcada como "sugestão"(`github.com/abergs/fido2-net-lib/issues/220`). As alterações sugeridas foram as seguintes:

- A primeira alteração era na inicialização do FIDO2 passar a receber múltiplas origens em vez de uma única origem, como se pode ver na Listagem 21 antes e o depois na Listagem 22;

Listagem 21: Inicializar do FIDO2 antes.

```
Fido2(new Fido2Configuration
    {
        ServerDomain = "bitwarden.com",
        ServerName = "Bitwarden",
        Origin = "bitwarden.com",
    },
);
```

Listagem 22: Inicializar do FIDO2 depois, com a correção sugerida.

```
Fido2(new Fido2Configuration
    {
        ServerDomain = "bitwarden.com",
        ServerName = "Bitwarden",
        OriginsAllowed  = [
            "bitwarden.com",
            "android:apk-key-hash:LBpmHljqwJJLdXVNpjfdA",
        ],
    },
);
```

- A segunda alteração, seria na verificação da assinatura do desafio, começar a receber a origem no pedido de autenticação, para assim ser verificado se a origem do pedido e da resposta eram as mesmas. Isto não é requerido pela norma FIDO2, mas como começam a existir múltiplas origens, seria uma boa abordagem de segurança verificar se são iguais;
- A terceira alteração, para cada vez que se recebe uma origem, tanto no pedido do desafio, como na verificação da assinatura, seja verificado se essa origem existe na lista de origens permitidas ("OriginsAllowed"), configuradas no inicializador. Desta forma impedem-se os ataques de *Phishing*, além de permitir origens de iOS e Android.

Com estas sugestão seria possível inicializar uma só vez o FIDO2 e passaria a ser permitido Android, iOS e Web, mantendo assim a verificação da origem. Além disso seria acrescida mais uma verificação entre a origem do pedido do desafio com a origem da resposta com o desafio assinado.

Ainda relativamente às dificuldades encontradas no Android, uma situação foi a aplicação móvel original do Bitwarden não ter forma de alterar a *password* do utilizador nem alterar o sistema de dois factores. Estas operações só estão disponíveis no serviço Web. Sendo assim, foi decidido não fazer a gestão de chaves FIDO2 no

Android, como está feito no projeto da Web. No entanto, o código ficou preparado caso num futuro próximo a comunidade do Bitwarden queira vir a colocar a gestão de chaves no Android.

6.11 DIVULGAÇÃO À COMUNIDADE BITWARDEN

Com o sucesso da implementação do FIDO2 na plataforma Bitwarden, foi partilhado a ideia e o projeto na plataforma de contribuições do Bitwarden (community.bitwarden.com/t/fido2-on-android-web-and-in-the-server/29080). Ao ser partilhado foi descoberto que a comunidade já estava a desenvolver o FIDO2 para o serviço Web, que não foram encontradas nas minhas pesquisas, porque a comunidade Bitwarden estava a usar o nome "WebAuthn" que é o nome que W3C dá ao FIDO2. Este projeto foi concluido a 16 de março de 2021. Foram estudadas as alterações feitas, mas como as alterações que eles tinham do FIDO2, não estavam preparadas para o problema que o projeto "`abergs/fido2-net-lib`" tem sobre as origens (mencionado anteriormente), decidiu-se continuar com o que tinha para poder também demonstrar à comunidade a autenticação com FIDO2 a funcionar no Android.

A 28 de julho de 2021, um programador de Xamarin da comunidade de Bitwarden, pediu a minha ajuda a colocar o FIDO2 no Android a funcionar. A comunicação entre nós foi num *chat* privado na plataforma de contribuições do Bitwarden (community.bitwarden.com), pois contém dados sensíveis dos seus servidores de teste. Na ajuda fornecida ao programador auxiliei a fazer *troubleshooting* ao projeto Android. Os desenvolvedores do Bitwarden colocaram os meus ficheiros e as minhas modificações do projeto Android. Assim tornou-se mais fácil de descobrir o problema, que era do lado do servidor: não disponibilizavam o "`/.well-known/assetlinks.json`" que é necessário para o FIDO2 no Android.

Na comunicação com esse programador também reportei sobre o problema na biblioteca "`abergs/fido2-net-lib`", que não permite mais que uma origem e que já tinha informado a comunidade dessa biblioteca. A comunidade do Bitwarden averiguou a situação e apresentou uma solução que permite manter compatibilidade com as versões anteriores da biblioteca. Essa solução está a ser ainda analisada e disponível em github.com/passwordless-lib/fido2-net-lib/pull/237. A 8 de setembro de 2021, nesse *pull request* a comunidade Bitwarden informou que não iam submeter as alterações do FIDO2 no Android, porque não conseguiram implementar

no iOS o FIDO2 e para assim não haver manutenção diferente do Android para o iOS (Figura 60).

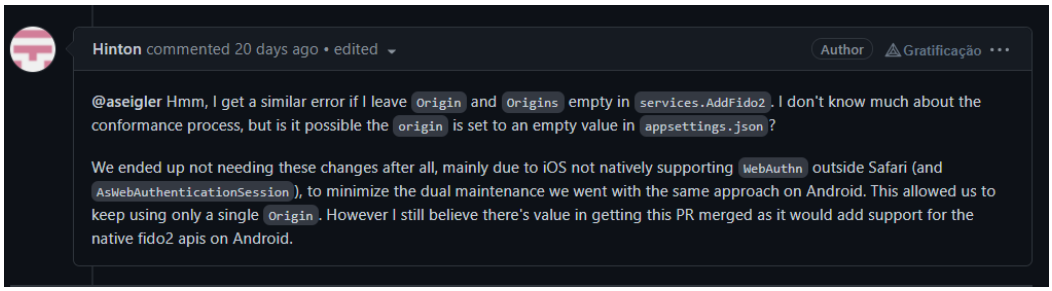

Figura 60: Bitwarden informam que não iam submeter as alterações de FIDO2 no Android (Lib, 2021).

6.12 PRODUTO FINAL

O desenvolvimento do FIDO2 na plataforma Bitwarden está dividido em duas aplicações para o utilizador: a aplicação Web e a aplicação móvel. Ambas têm o mesmo objetivo que é autenticar com FIDO2. No lado da aplicação Web ainda tem a parte de poder adicionar chaves FIDO2, o que quer dizer que ambas as aplicações têm o mesmo número de ecrãs para autenticar. No entanto, só aplicação Web tem a parte de registar. Por essa razão antes de avançar com a demonstração da aplicação e respetiva explicação das suas funcionalidades, são mostradas primeiro as rotas possíveis de ecrã que têm implicação no que foi desenvolvido e que pode ser visualizado na Figura 61.

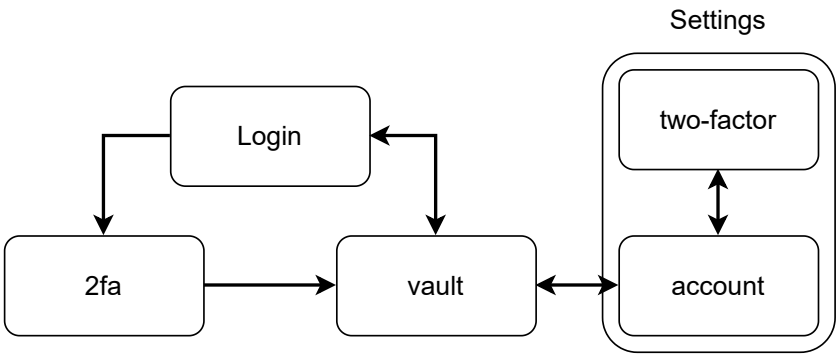

Figura 61: Arquitetura de ecrãs.

Existem ainda os ecrãs que são do sistema operativo na parte de autenticação FIDO2 e que são os mesmos que aparecem no projeto de prova de conceito nas Figuras 38, 39, 40, 41 e 42.

Antes de iniciar a exposição dos ecrãs, é necessário dizer quais os ecrãs que a aplicação Android não possui. Estes são os ecrãs que estão na seção "Settings" da Figura 61. Tendo este facto em mente dá-se início à demonstração dos ecrãs, começando pelo "Login". Este é o ecrã de entrada que um utilizador decide se quer autenticar ou registar uma nova conta, como se pode ver nas Figuras 62 e 63.

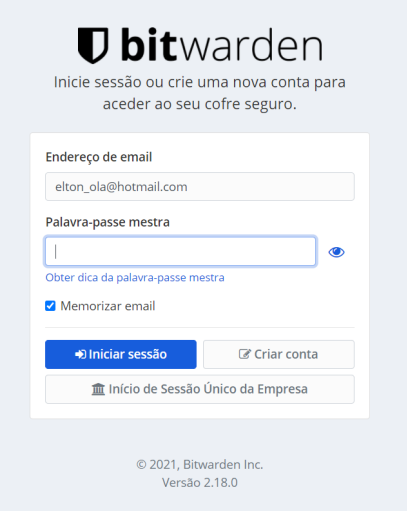

Figura 62: Ecrã "Login" da Aplicação Web.

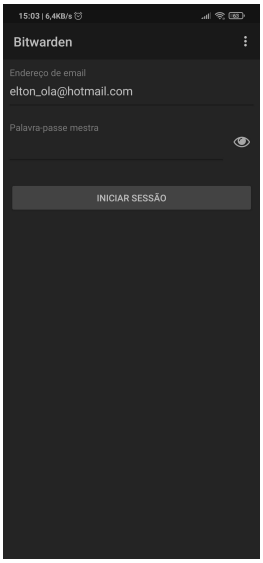

Figura 63: Ecrã "Login" da Aplicação Android.

Quando o utilizador insere as suas credenciais para se autenticar e tem o sistema FIDO2 de dois factores ativo, é levado para a página de autenticação usando FIDO2. O FIDO2 é logo iniciado quando a página é carregada, pedindo de seguida ao FIDO2 cliente para assinar o desafio. Em caso de erro, ou não ser iniciado, o utilizador pode pedir um novo desafio para assinar e concluir a autenticação e ser redirecionado para o ecrã "vault".

O ecrã "vault" apresenta, neste caso, autenticação com sucesso, mas na aplicação serve para aceder às suas informações e às palavras-chaves que estão guardadas na plataforma. Este ecrã possibilita a gestão da autenticação FIDO2 no caso da aplicação Web, em que pode ser acedido indo para seção "Settings". Aí existe o ecrã "two-factor", que é acessível pelo botão "Início de sessão de dois passos". Este ecrã permite a gestão de todos os métodos de dois factores disponíveis.

No ecrã "two-factor" é então aberto uma mini-janela quando selecionada a gestão do FIDO2 no ecrã. Esta mini-janela pede a chave mestre da conta e de seguida é aberto então a parte onde tem as chaves FIDO2 listadas com o seu nome, o seu tipo e se estão comprometidas ou não. Também é possibilitado a eliminação de uma ou todas as chaves FIDO2 e o registo de uma nova chave FIDO2. Esta última ação é

Figura 64: Ecrã "2FA" da Aplicação Web.

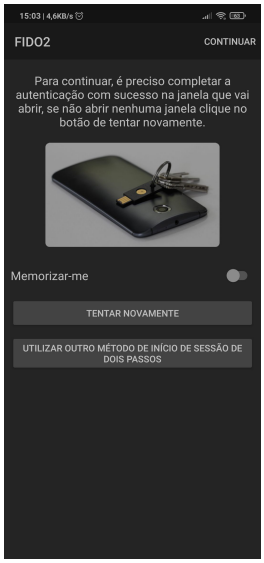

Figura 65: Ecrã "2FA" da Aplicação Android.

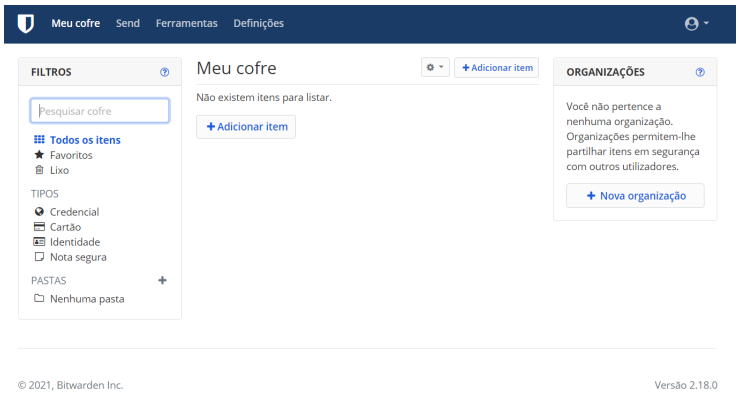

Figura 66: Ecrã "vault" da Aplicação Web.

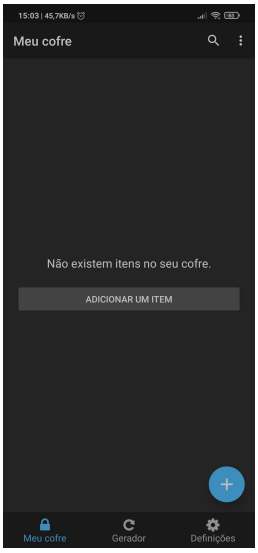

Figura 67: Ecrã "vault" da Aplicação Android.

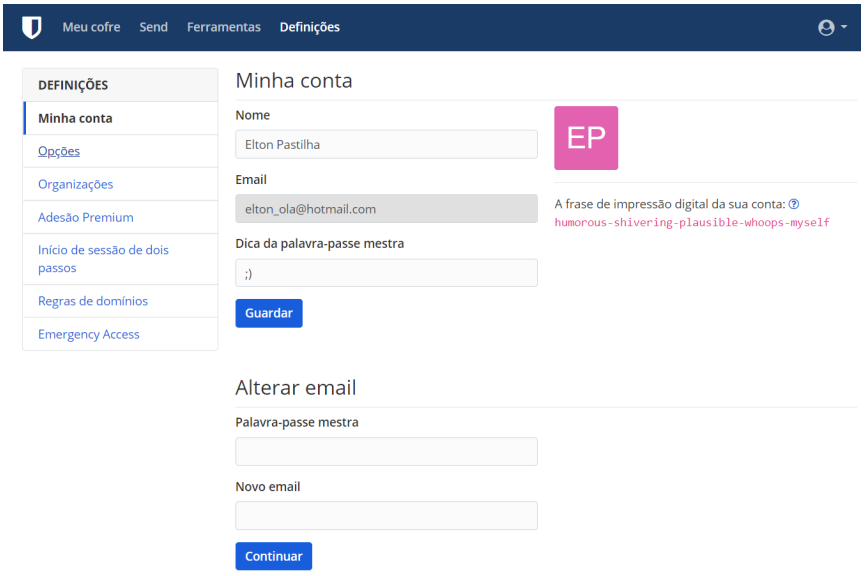

Figura 68: Ecrã "account" da Aplicação Web.

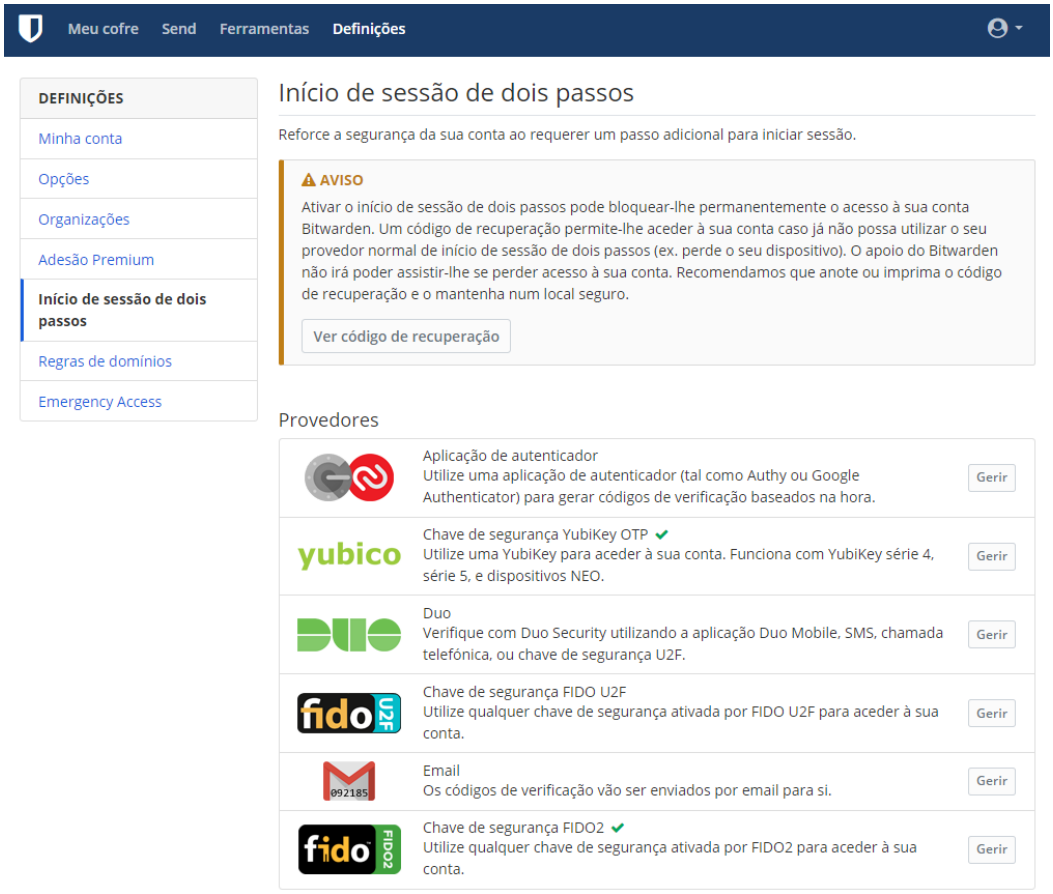

Figura 69: Ecrã "two-factor" da Aplicação Web.

realizada através dos campos que estão disponíveis onde é necessário o nome que se quer dar à chave FIDO2 e que tipo de sistema FIDO2 será usado “Platform” ou “Cross-Platform”.

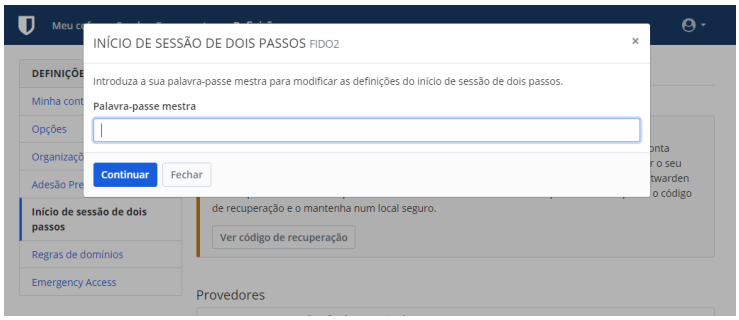

Figura 70: Ecrã “two-factor” com a mini-janela a pedir chave mestra.

Figura 71: Ecrã “two-factor” com a mini-janela de gestão do FIDO2.

7

CONCLUSÕES

Existem diversos tipos de autenticação, em que cada um tem uma forma diferente de trabalhar e de proteger o sistema. Hoje em dia sabemos que o uso de autenticação por meio de *password* tem-se mostrado pouco eficaz contra ataques de entidades maliciosas, que conseguem obter acesso através de ataques simples como *Phishing*, ou através de ataques com dicionários de *password*. Isto porque os utilizadores têm limitações e também porque esta forma de autenticação não resiste bem a estes tipo de ataques. Ao verificar-se que autenticação com *password* é fraca, foram criadas formas de autenticação complementares. A forma de autenticação que tem mostrado maior eficácia contras os ataques referidos em cima é a autenticação baseada em chaves de criptografia assimétrica. Assim, temos o FIDO2 que suporta a autenticação com chaves assimétricas para a combater ataques à autenticação como exemplo *Phishing*.

Como explicado ao longo deste relatório, o FIDO2 vem dar apoio ao combater os problemas da autenticação com *password*. Este pode ser usado como *passwordless* ou como autenticação de dois factores. Esta tecnologia permite utilizar dois tipos de autenticadores, o autenticador interno (por exemplo o Windows Hello) e o autenticador externo (por exemplo o dispositivo Yubikey). O FIDO2 na versão atual já é apresentado como "*Standard*" adotado pelo World Wide Web Consortium (W3C). Por essa razão decidiu-se aplicar esta tecnologia a um projeto *open-source* e com informação crítica. A aplicação escolhida foi o Bitwarden que tem como propósito gerir as *passwords* dos utilizadores.

Durante o desenvolvimento deste trabalho, todas as questões, decisões sobre requisitos e melhores maneiras de execução das aplicações, foram feitas com cuidado para não haver um impacto no desenvolvimento fluido de todo o trabalho. Mesmo quando houve situações de conflito de informações e outros problemas durante o desenvolvimento. Na implementação do FIDO2 verificou-se uma grande falta de documentação de implementação para Android. Este facto tornou mais complexo o desenvolvimento deste trabalho. As dificuldades foram sendo ultrapassadas através do uso de um ambiente controlado que se revelou ser uma parte crítica do pro-

jeto. Quer para testar, quer para aprender como implementar o FIDO2 (prova de conceito).

O projeto no final teve o desfecho esperado que era a implementação do FIDO2 no Bitwarden. Desta forma tornando a aplicação e a plataforma mais segura contra ataques de entidades maliciosas. Foi ainda partilhado o trabalho desenvolvido com a comunidade do Bitwarden. Esta solicitou ajuda na implementação do FIDO2 para Android, onde foram usados os ficheiros criados por mim para a aplicação Android. A partilha de informação com a comunidade do Bitwarden foi útil, mas como o FIDO2 tem limitações relativamente à implementação em sistemas operativos móveis, a comunidade do Bitwarden desistiu de implementar o FIDO2 nas suas aplicações móveis (Android e iOS).

7.1 LIMITAÇÕES

Atualmente existe uma vasta informação acerca do protocolo do FIDO2. Essa informação pode ser lida na FIDO Alliance ou no W3C. A informação lá descrita é bastante detalhada, mas existe uma grande falta de documentação de como implementar o FIDO2 nos sistemas operativos móveis Android e iOS. Este problema levou a comunidade Bitwarden a desistir de implementar o FIDO2 no iOS, e com isso levou com que não fosse adicionado o FIDO2 no Android. O FIDO2 é um bom sistema de autenticação mas atualmente o custo de implementação e a falta de documentação sobre a implementação, leva à desistência desta tecnologia.

O Android disponibiliza informações sobre a API do FIDO2 e tem alguns exemplos de implementação. No entanto, não tem documentação sobre como chamar essa API e a falta de documentação de erros que API retorna. Uma das situações que o Bitwarden pediu ajuda, foi precisamente tentar entender os erros que API retornava.

No iOS, mesmo não sendo o foco do projeto, foi procurada informação para tentar ajudar a comunidade do Bitwarden. Infelizmente não foi encontada informação que ajudasse na sua implementação, a única informação encontrado foi no projeto de demonstração disponível aqui: `github.com/singularkey/iosfido2demo`.

Em relação ao *open source* “abergs/fido2-net-lib”, este tem uma limitação na versão atual. Essa limitação impede de usar mais que uma origem, o que torna o processo complicado, em casos como, uma plataforma que tem uma aplicação Android e uma aplicação Web. Não é difícil de contornar esta limitação, só que ao

contornar este problema, pode deixar esta autenticação com problemas de *Phishing*, como mencionado no 6.10.

7.2 TRABALHO FUTURO

Embora este projeto tenha tido sucesso na implementação do FIDO2 no Android e na Web, é importante relembrar que este projeto ainda tem muito mais para oferecer no que toca às funcionalidades que o FIDO2 oferece. A começar pela implementação do FIDO2 nos sistemas iOS e nos dispositivos IoT. A FIDO Alliance fornece documentação para implementação do FIDO2 em dispositivos IoT, usando um autenticador interno para autenticar nos servidores ou nos serviços, em que esse protocolo chama-se o FIDO Device Onboard (FDO).

BIBLIOGRAFIA

Ah Kioon, Mary Cindy, Zhao Shun Wang e Shubra Deb Das (2013). «Security analysis of MD5 algorithm in password storage». Em: *Applied Mechanics and Materials*. Vol. 347. Trans Tech Publ, pp. 2706–2711.

Alliance, FIDO (jun. de 2020). *FIDO2: Moving the World Beyond Passwords using WebAuthn & CTAP*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/fido2.

— (s.d.). *Alliance Overview*. Website. URL: https://fidoalliance.org/overview/.

Alliance, Fido (nov. de 2018). *Authenticator Level 3*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/authenticator-certification-levels/authenticator-level-3.

— (abr. de 2019a). *Authenticator Level 1*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/authenticator-certification-levels/authenticator-level-1.

— (abr. de 2019b). *Authenticator Level 2*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/authenticator-certification-levels/authenticator-level-2.

— (abr. de 2019c). *Authenticator Level 3+*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/authenticator-certification-levels/authenticator-level-3-plus.

— (abr. de 2019d). *Certification Overview*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification.

— (nov. de 2019e). *FIDO U2F Implementation Considerations*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-implementation-considerations-v1.2-ps-20170411.pdf.

— (jul. de 2021a). *Biometric Component Certification*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/biometric-component-certification.

— (jul. de 2021b). *Certified Authenticator Levels*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/authenticator-certification-levels.

Alliance, Fido (mar. de 2021c). *Client to Authenticator Protocol (CTAP)*. [Online; accessed 21. Jul. 2021]. URL: https://fidoalliance.org/specs/fido-v2.1-rd-20210309.

— (mai. de 2021d). *FIDO Alliance Metadata Service*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/metadata.

— (ago. de 2021e). *Functional Certification*. [Online; accessed 11. Jun. 2021]. URL: https://fidoalliance.org/certification/functional-certification.

Babich, Aleksandra (abr. de 2019). *Biometric Authentication. Types of biometric identifiers*. Website. [Online; accessed 11. Jun. 2021]. URL: https://www.theseus.fi/bitstream/handle/10024/44684/Babich_Aleksandra.pdf.

bitwarden (set. de 2021). *Install and Deploy*. [Online; accessed 11. Jun. 2021]. URL: https://bitwarden.com/help/article/install-on-premise.

Bonneau, Joseph et al. (2015). «Passwords and the evolution of imperfect authentication». Em: *Communications of the ACM* 58.7, pp. 78–87.

Consortium, World Wide Web (abr. de 2021a). *Web Authentication: An API for accessing Public Key Credentials - Level 2*. [Online; accessed 11. Jun. 2021]. URL: https://www.w3.org/TR/webauthn-2//#attestation-signature.

— (jul. de 2021b). *Web Authentication: An API for accessing Public Key Credentials - Level 3*. [Online; accessed 11. Jun. 2021]. URL: https://w3c.github.io/webauthn//#sctn-defined-attestation-formats.

Creese, Sadie et al. (jul. de 2013). «Relationships between Password Choices, Perceptions of Risk and Security Expertise». Em: *Human Aspects of Information Security, Privacy, and Trust*. Berlin, Germany: Springer, pp. 80–89. ISBN: 978-3-642-39344-0. DOI: 10.1007/978-3-642-39345-7_9.

Eldefrawy, Mohamed Hamdy, Khaled Alghathbar e Muhammad Khurram Khan (2011). «OTP-Based Two-Factor Authentication Using Mobile Phones». Em: *2011 Eighth International Conference on Information Technology: New Generations*, pp. 327–331. DOI: 10.1109/ITNG.2011.64.

FidoAlliance (s.d.). *History of FIDO Alliance*. Website. URL: https://fidoalliance.org/overview/history/.

Gibson, Steve (out. de 2019). *Welcome to SQRL*. Website. URL: https://www.grc.com/sqrl/SQRL_Explained.pdf.

Heartfield, Ryan e George Loukas (2015). «A taxonomy of attacks and a survey of defence mechanisms for semantic social engineering attacks». Em: *ACM Computing Surveys (CSUR)* 48.3, pp. 1–39.

Kelly (nov. de 2020). *YubiKey FIPS*. Website. [Online; accessed 11. Jun. 2021]. URL: https://support.yubico.com/hc/en-us/articles/360013761699-YubiKey-FIPS.

Lib, Passwordless (set. de 2021). *Support Multiple Origins*. [Online; accessed 11. Jun. 2021]. URL: https://github.com/passwordless-lib/fido2-net-lib/pull/237.

LoginRadius (jun. de 2019). *What is Multi-Factor Authentication (MFA)*. Website. URL: https://www.loginradius.com/blog/start-with-identity/2019/06/what-is-multi-factor-authentication/.

Maayan, Gilad David (s.d.). *5 Authentication Methods that Can Prevent the Next Breach*. Website. URL: https://www.idrnd.ai/5-authentication-methods-that-can-prevent-the-next-breach/.

OneLogin (s.d.). *What's the Difference Between OTP, TOTP and HOTP?* Website. URL: https://www.onelogin.com/learn/otp-totp-hotp.

Opensource (set. de 2021). *What is Docker?* [Online; accessed 11. Jun. 2021]. URL: https://opensource.com/resources/what-docker.

Orenstein, Gary (jul. de 2020). *Vault Security in the Bitwarden Password Manager*. Website. https://bitwarden.com/blog/post/vault-security-bitwarden-password-manager/ acedido em 2021.05.26. URL: https://bitwarden.com/blog/post/vault-security-bitwarden-password-manager/.

Petsas, Thanasis et al. (2015). «Two-factor authentication: is the world ready? Quantifying 2FA adoption». Em: *Proceedings of the eighth european workshop on system security*, pp. 1–7.

Priberam Informática, S. A. (jun. de 2021). *fido*. website. [Online; accessed 11. Jun. 2021]. URL: https://dicionario.priberam.org/fido.

Stallings, William (out. de 2018). *Cryptography and Network Security: Principles and Practice, eBook, Global Edition*. Toronto, Ontario, Canada: Pearson Education. ISBN: 978-1-29215859-4. URL: https://books.google.pt/books?id=dogWDQAAQBAJ&dq=Cryptography+and+Network+Security:+Principles+and+Practice,+Global+Edition&hl=en&sa=X&redir_esc=y.

Sun, San-Tsai et al. (2011). «What makes users refuse web single sign-on? An empirical investigation of OpenID». Em: *Proceedings of the Seventh Symposium on Usable Privacy and Security*, pp. 1–20.

Taneski, Viktor, Marjan Heričko e Boštjan Brumen (2019). «Systematic overview of password security problems». Em: *Acta Polytechnica Hungarica* 16.3.

Thomas, Kurt e Angelika Moscicki (mai. de 2019). *New research: How effective is basic account hygiene at preventing hijacking*. Website. URL: https://security.googleblog.com/2019/05/new-research-how-effective-is-basic.html.

Tzur-David, Shimrit (jun. de 2020). *Your Complete Guide to FIDO, FIDO2 and WebAuthn*. [Online; accessed 11. Jun. 2021]. URL: https://doubleoctopus.com/blog/your-complete-guide-to-fido-fast-identity-online.

W3C, Community of (abr. de 2021). *Web Authentication: An API for accessing Public Key Credentials - Level 2*. [Online; accessed 19. Oct. 2021]. URL: https://www.w3.org/TR/webauthn-2.

Wang, Ding e Ping Wang (2015). «Offline dictionary attack on password authentication schemes using smart cards». Em: *Information security*. Springer, pp. 221–237.

Whittaker, Zack (nov. de 2018). *A leaky database of SMS text messages exposed password resets and two-factor codes*. Website. URL: https://techcrunch.com/2018/11/15/millions-sms-text-messages-leaked-two-factor-codes/.

APÊNDICES

A

APÊNCICE A

Lista de alterações efetuadas nos projetos Bitwarden, separado por projeto e tambem separado pelos que foram criados e modificados.

- Ficheiros no projeto Server:
  - Ficheiros criados:
    - `server/Util/Setup/Templates/AssetLinks.hbs`
    - `server/Util/Setup/AssetLinksBuilder.cs`
    - `server/Util/Migrator/2021-02-08\_00\_Fido2.sql`
    - `server/src/Sql/dbo/Tables/Fido2Key.sql`
    - `server/src/Sql/dbo/Tables/Fido2Challenge.sql`
    - `server/src/Sql/dbo/Views/Fido2KeyView.sql`
    - `server/src/Sql/dbo/Views/Fido2ChallengeView.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_Create.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_DeleteById.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_DeleteByUserId.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_ReadById.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_ReadByUserId.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Key\_UpdateSignatureCounter.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Challenge\_Create.sql`
    - `server/src/Sql/dbo/StoredProcedures/Fido2Challenge\_DeleteById.sql`

    * server/src/Sql/dbo/StoredProcedures/Fido2Challenge_DeleteManyExpired.sql
    * server/src/Sql/dbo/StoredProcedures/Fido2Challenge_ReadLastCreatedByU
      sql
    * server/src/Sql/dbo/StoredProcedures/Fido2Challenge_ReadManyNotExpired
      sql
    * server/src/Core/Models/Table/Fido2Key.cs
    * server/src/Core/Models/Table/Fido2Challenge.cs
    * server/src/Core/Repositories/IFido2KeyRepository.cs
    * server/src/Core/Repositories/IFido2ChallengeRepository.cs
    * server/src/Core/Repositories/SqlServer/Fido2KeyRepository.cs
    * server/src/Core/Repositories/SqlServer/Fido2ChallengeRepository.cs
    * server/src/Core/Enum/Fido2ActionType.cs
    * server/src/Core/Identity/Fido2TokenProvider.cs
    * server/src/Core/Models/Api/Response/TwoFactor/TwoFactorFido2ResponseMo
      cs

  – Ficheiros Modificados:

    * server/Util/Setup/Templates/NginxConfig.hbs (linhas: 87-97)
    * server/Util/Setup/Program.cs (linhas: 261-263)
    * server/Util/Setup/EnvironmentFileBuilder.cs (linhas: 25, 92)
    * server/src/Core/GlobalSettings.cs (linhas: 51)
    * server/src/Core/Core.csproj (linhas: 29)
    * server/src/Core/Enum/TwoFactorProviderType.cs (linhas: 12)
    * server/src/Core/IdentityServer/BaseRequestValidator.cs (linhas: 369, 397, 425-431)
    * server/src/Core/Models/TwoFactorProvider.cs (linhas: 51)
    * server/src/Core/Models/Api/Request/TwoFactorRequestModels.cs (linhas: 3-4, 248-276, 296-311)
    * server/src/Core/Services/IUserService.cs (linhas: 10, 30-35)

    * `server/src/Core/Services/Implementations/UserService.cs` (linhas: 24-29, 36-38, 61-62, 70, 125-127, 505-848, 1079-1083)
    * `server/src/Core/Utilities/ServiceCollectionExtensions.cs` (linhas: 78, 295-296)
    * `server/src/Api/Controllers/TwoFactorController.cs` (linhas: 17-18, 266-322)

- Ficheiros no projeto Web:
  - Ficheiros criados:
    * `web/src/assetlinks.json`
    * `web/app/settings/two-factor-fido2.component.html`
    * `web/app/settings/two-factor-fido2.component.ts`
    * `web/images/two-factor/7.png`
    * `web/images/fido2key.jpg`
    * `web/jslib/src/models/request/twoFactorFido2ChallengeRequest.ts`
    * `web/jslib/src/models/request/twoFactorFido2Response.ts`
  - Ficheiros Modificados:
    * `web/entrypoint.sh` (linhas: 34)
    * `web/webpack.config.js` (linhas: 102)
    * `web/app/accounts/two-factor.component.html` (linhas: 47-57, 80-86)
    * `web/app/settings/two-factor-setup.component.html` (linhas: 55)
    * `web/app/settings/two-factor-setup.component.ts` (linhas: 27, 39, 128-134)
    * `web/app/settings/two-factor-verify.component.ts` (linhas: 71-73)
    * `web/app/app.module.ts` (linhas: 139, 402, 458)
    * `web/locales/pt-PT/messages.json` (linhas: 683-685, 731-736, 1254-1262, 1308-1313, 1347-1373)
    * `web/app/services/services.modules.ts` (linhas: 156-159, 163-166)
    * `web/build.sh` (linhas: 13, 18, 26, 32)

    * `web/jslib/src/abstractions/api.service.ts` (linhas: 67-71, 128, 281-283, 300-302)
    * `web/jslib/src/angular/components/two-factor.component.ts` (linhas: 14, 31-34, 51-53, 69, 128-137, 199-205, 242-319)
    * `web/jslib/src/enum/twoFactorProviderType.ts` (linhas: 9)
    * `web/jslib/src/services/api.service.ts` (linhas: 75-80, 135-137, 862-891, 958-972)
    * `web/jslib/src/services/auth.service.ts` (linhas: 68-75, 125-127, 214-217)

- Ficheiros no projeto Android:
  - Ficheiros criados:
    * `android/src/Android/Fido2System/Fido2BuilderDictionary.cs`
    * `android/src/Android/Fido2System/Fido2BuilderObject.cs`
    * `android/src/Android/Fido2System/Fido2Service.cs`
    * `android/src/Core/Enum/Fido2CodesTypes.cs`
    * `android/src/Core/Models/Data/Fido2AssertionResponse.cs`
    * `android/src/Core/Models/Data/Fido2AuthenticatorSelection.cs`
    * `android/src/Core/Models/Data/Fido2CredentialDescriptor.cs`
    * `android/src/Core/Models/Data/Fido2PubKeyCredParam.cs`
    * `android/src/Core/Models/Data/Fido2RP.cs`
    * `android/src/Core/Models/Data/Fido2User.cs`
    * `android/src/Core/Models/Request/Fido2AuthenticationChallengeRequest.cs`
    * `android/src/Core/Models/Request/TwoFactorFido2ChallengeRequest.cs`
    * `android/src/Core/Models/Response/Fido2AuthenticationChallengeResponse.cs`
    * `android/src/Core/Models/Response/Fido2RegistrationChallengeResponse.cs`
    *

*

– Ficheiros Modificados:

  * `android/src/Android/MainActivity.cs` (linhas: 22-24, 35, 122-129, 147-148, 263-283, 291-327)
  * `android/src/Android/Android.csproj` (linhas: 95-97, 121-123, 281-285)
  * `android/src/Core/Enum/TwoFactorProviderType.cs` (linhas: 12)
  * `android/src/Core/Services/ApiService.cs` (linhas: 312-317, 435-437)
  * `android/src/Core/Services/AuthService.cs` (linhas: 92-98, 124-125, 203-207)
  * `android/src/Core/Abstractions/IApiService.cs` (linhas: 54)
  * `android/src/Core/Abstractions/IPlatformUtilsService.cs` (linhas: 28)
  * `android/src/App/Pages/Accounts/TwoFactorPage.xaml.cs` (linhas: 93-106, 182-185)
  * `android/src/App/Pages/Accounts/TwoFactorPage.xaml` (linhas: 117-140)
  * `android/src/App/Pages/Accounts/TwoFactorPageViewModel.cs` (linhas: 66-67, 76, 80, 89, 143-145)
  * `android/src/App/Resources/AppResources.pt-PT.resx` (linhas: 62-63, 70-75, 87-93, 99-101, 1798-1833)
  * `android/src/App/Services/MobilePlatformUtilsService.cs` (linhas: 137-144)