Autenticação baseada em Zero Conhecimento, com gestão de identidade centrada no utilizador

Pedro Miguel Nicolau Escaleira
escaleira@ua.pt

27/06/2021

Conteúdo

1 Introdução

O trabalho que iremos apresentar neste relatório consiste numa forma alternativa de utilizar o protocolo de autenticação baseado em *Zero-knowledge Proof* apresentado no relatório [1] para realizar a autenticação dum utilizador perante uma entidade, onde o utilizador tem controlo sobre as informações que são transferidas entre a entidade identificadora (*Identity Provider* ou *IdP*) e o provedor de serviço (*Service Provider*, ou *SP*).

Enquanto que no trabalho exposto em [1] o *IdP* enviava toda a informação do utilizador pedida pelo *SP* a este mesmo diretamente, após uma autenticação com sucesso do utilizador feita entre uma *helper application* e o *IdP*, nesta nova solução todos os dados do utilizador passam primeiro pela *helper application* antes de serem enviados para uma ou para a outra entidade. Como é possível observar na figura 1, quando o utilizador acede ao *SP*, este indica à *helper application*, que se encontra a ser executada na máquina pessoal do utilizador, que este se tem de autenticar devidamente e que terá de enviar determinados atributos que serão necessários para o identificar. De seguida, a *helper application* apresenta estes atributos ao utilizador, de forma a que este tenha total conhecimento de que informação pessoal este *SP* necessita para o identificar, e só caso o utilizador aprove que estes dados sejam recolhidos é que a *helper application* irá fazer a sua autenticação perante o *IdP* e obter do mesmo estes atributos. Desde logo, há aqui uma diferença notável em relação à forma como era tratado o pedido de identificação no trabalho anterior, onde o *SP* pedia diretamente ao *IdP* os atributos necessários para autenticar o utilizador e, após esta interação, o *IdP* enviava um pedido de autenticação à *helper application*. Ou seja, neste novo protocolo, **o utilizador passa a ter controlo sobre os dados que são pedidos pelo *SP***, sendo sempre informado de quais são necessários para o identificar e tendo de dar sempre a sua permissão para estes serem tratados pelo *SP* e pedidos ao *IdP*.

Após uma autenticação com sucesso entre a *helper application* e o *IdP*, que é feita usando o protocolo *ZKP* já apresentado no trabalho anterior, o *IdP* envia os atributos previamente pedidos à *helper application*, sendo que mais uma vez todos os dados referentes ao utilizador passam por esta entidade central. Quanto ela os recebe, apresenta-os ao utilizador, de forma a que uma vez mais este tenha pleno conhecimento dos dados que serão partilhados com o *SP*, e só os enviará caso o utilizador dê permissão.

Como podemos também entender, não existe qualquer necessidade do *IdP* saber qual o *SP* que está a fazer o pedido de autenticação e identificação do utilizador nesta nova solução, ao contrário da anterior. O *SP* terá apenas de confiar no *IdP* de forma a que confie na identificação do utilizador providenciada por ele. Assim sendo, esta nova solução permite também a **privacidade do utilizador em relação aos *SPs* que acede perante o *IdP* que o autentica e identifica**.

Como já referido, esta nova solução baseia-se na implementação apresentada no trabalho [1], sendo que houve apenas alguma reestruturação da lógica do código correspondente. Houve também porções que não sofreram quaisquer alterações, sendo elas o protocolo *ZKP* realizado entre a *helper application* e o *IdP*, o chaveiro da *helper application*, usado para fazer o tratamento das chaves e palavras-passe do utilizador localmente de forma segura, e toda a plataforma do *SP* correspondente à conta do utilizador. É também necessário evidenciar que nesta nova solução não foi usado *SAML* para efetuar os pedidos e respostas de atributos, sendo que estes processos foram realizados de forma alternativa, como iremos apresentar neste relatório.

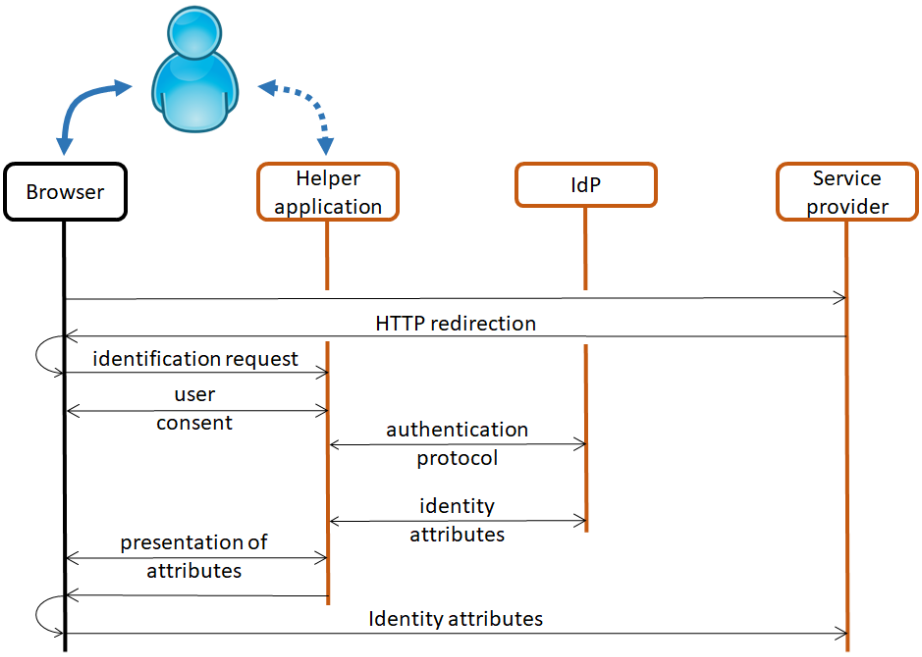

Figura 1: Arquitetura do sistema implementado, providenciada pelo professor da disciplina.

2 Protocolo de identificação

Como já explicado na introdução deste relatório, na secção 1, quando um dado utilizador acede ao *SP*, este necessita que ele seja devidamente identificado. Desta forma, o *SP* possui conhecimento de vários *IdPs*[1], selecionando um no qual é possível o utilizador identificar-se perante o *SP* com sucesso.

Na nossa solução, o *SP* desenvolvido necessita apenas dum único atributo do utilizador, de forma a identificar o mesmo com sucesso: o "*username*". Portanto, quando um utilizador acede ao *SP* e não se encontra autenticado, o *SP* redireciona o *browser* para a *helper application*, indicando os atributos (parâmetro "*id_attrs*") necessários obter para identificar o utilizador, que neste caso é apenas o "*username*", o *IdP* (parâmetro "*idp*") que ela deverá contactar de forma a obter esses atributos, bem como o *URL* correspondente que deverá contactar para iniciar o processo de obtenção dos mesmos (parâmetro "*sso_url*"), o seu (do *SP*) *URL* (parâmetro "*sp*") e o *URL* que a *helper application* deverá contactar quando já tiver recolhido os atributos (parâmetro "*consumer_url*"). Para além destes parâmetros enviados, é enviado também um identificador único (parâmetro "*client*"), criado através dum *UUID* versão 4, de maneira a que quando a *helper application* responder com os atributos pedidos, o *SP* consiga manter estado e saber a correspondência entre pedidos. Na listagem 1 é possível encontrar um exemplo dos parâmetros enviados num destes pedidos.

```
sp="http://127.0.0.1:8081"
idp="http://127.0.0.1:8082"
id_attrs="username"
consumer_url="http://127.0.0.1:8081/identity"
sso_url="http://127.0.0.1:8082/login"
client="300739d4-a21e-45ff-bfcb-f34eb933070d"
```

Listagem 1: Parâmetros do pedido *GET* correspondente ao pedido de identificação enviado pelo *SP* à *helper application*.

Quando a *helper application* recebe este pedido, irá começar por pedir consentimento ao utilizador, como iremos apresentar na secção 4 e, de seguida, irá redirecionar o *browser* para o endereço indicado pelo parâmetro "*sso_url*", de maneira a indicar ao *IdP* que pretende iniciar o processo de *login* (autenticação e identificação) e de forma a receber uma chave que irá ser usada para cifrar toda a comunicação entre estas duas entidades, permitindo a criação dum túnel seguro de comunicação (uma vez que a *helper application* faz um *redirect* para o *IdP* e que este, por sua vez, irá fazer um *redirect* para voltar a "dar controlo à *helper application*", estas duas interações são feitas, pressupomos nós, com *HTTPS*, sendo por isso cifradas num túnel seguro, pelo que a transmissão da chave a ser usada na comunicação posterior entre as duas entidades é feita de forma cifrada e segura). Quando o *IdP* recebe este pedido, faz novamente um *redirect* para a *helper application* enviando não só, como já referido, uma chave *AES* criada de forma aleatória (usando o *urandom*, tal como apresentado no trabalho [1]), mas também parâmetros cujos valores serão necessários aos processos de autenticação e identificação que irão ser implementados de seguida. Como se pode observar na listagem 2, correspondente a um exemplo dos valores dos parâmetros enviados neste redirecionamento, para além da chave, que é enviada através do parâmetro "*key*", são enviados também os valores de "*max_iterations*"e "*min_iterations*", que são necessários para realizar a autenticação *ZKP* como apresentada no trabalho [1], um parâmetro "*client*", que corresponde a um identificador criado de forma aleatória a ser usado na comunicação entre as duas entidades, de maneira a que o *IdP* associe os pedidos deste utilizador em especifico entre si, mantendo estado na comunicação[2], e 3 parâmetros cujos valores correspondem a *URLs*

[1] É de ter em conta que na nossa solução, sendo um estado de arte, o *SP* usa apenas um *IdP* como entidade identificadora, mas seria simples adicionar múltiplos, sem que isso significasse uma alteração do comportamento das várias entidades desenvolvidas.

[2] Mais uma vez, este valor foi obtido de forma aleatória através dum *UUID* versão 4. É de notar que este valor nada tem a ver com o valor do parâmetro *client* enviado inicialmente do *SP* para a *helper application*, sendo que esse é criado pelo *SP* e este pelo *IdP*, os dois de forma autónoma e independente.

a serem usados nos processos de autenticação e identificação: o valor do parâmetro "*auth_url*", corresponde ao *URL* ao qual a *helper application* se deve dirigir para autenticar o utilizador, usando o protocolo *ZKP*; o valor do parâmetro "*save_pk_url*", corresponde ao *URL* que a *helper application* deve contactar no final da autenticação, de maneira a enviar a uma nova chave pública para o utilizador que acabou de autenticar; e o valor do parâmetro "*id_url*", corresponde ao *URL* que deve ser acedido quando a *helper application* pretender efetuar o processo de identificação do utilizador, onde irá obter a identificação pedida pelo *SP*. Nesta secção iremos apenas focar-nos na utilização do valor do parâmetro "*id_url*", uma vez que os outros dois são usados no processo de autenticação, a ser explicado na secção 3.

```
max_iterations="1000"
min_iterations="300"
client="a8c92a12-d30c-4321-89fb-3bfc040a882f"
key="jL1PzMkuR--Dw-KM06jy7fpMJCkmhFf1jsfWGUfhrTg="
auth_url="http://127.0.0.1:8082/authenticate"
save_pk_url="http://127.0.0.1:8082/save_asymmetric"
id_url="http://127.0.0.1:8082/identification"
```

Listagem 2: Parâmetros do pedido *GET* correspondente ao redirecionamento feito do *IdP* à *helper application*, como resposta ao pedido de *login*.

Após este redirecionamento feito pelo *IdP* para a *helper application*, esta irá começar por verificar se possui um par de chaves para o utilizador que irá identificar e se este par de chaves é ainda válido. Caso uma destas verificações seja falsa, irá iniciar o processo de autenticação *ZKP*, que irá ser explicado na secção 3. Caso contrário, iniciará o processo de identificação do utilizador com o *IdP*. Para isso, envia um pedido *GET* diretamente (não passando pelo *browser*) para o *URL* indicado em "*id_url*", no qual irá apontar quais os atributos que o *SP* requisitou e que este considera como necessários à identificação do utilizador. Neste pedido, é enviada também uma assinatura criada com a chave privada do utilizador, que só a *helper application* possui (e para a qual, como iremos ver na secção 3, o *IdP* é a única entidade que possui a corresponde chave pública), sobre o valor codificado em *Base64* do *JSON dump* dos atributos pedidos. Neste pedido, existem três parâmetros:

- O parâmetro "*client*": correspondente ao identificador do utilizador, obtido no redirecionamento anterior.
- O parâmetro "*ciphered*": corresponde ao texto cifrado, codificado em *Base64*, dos parâmetros privados a serem enviados para o *IdP* i.e., corresponde ao túnel criado entre o *IdP* e a *helper application*. A chave da cifra é a chave obtida no redirecionamento anterior e o *IV* (vetor de inicialização) é criado de forma aleatória através do "*urandom*". O conjunto de parâmetros cifrados enviados é:
  - O parâmetro "*user_id*": é um identificador único usado pelo *IdP* para identificar cada par de chaves de cada *helper application* e cada utilizador, sendo que a *helper application* o envia de forma a que o *IdP* entenda qual a chave pública que corresponde à chave privada deste utilizador. Este valor irá ser melhor explicado na secção 3.
  - O parâmetro "*username*": corresponde ao nome do utilizador que se pretende identificar e ao qual pertence a chave pública identificada pelo "*user_id*".
  - O parâmetro "*id_attrs*": corresponde ao valor do *JSON dump*, codificado em *Base64*, dos atributos que o *SP* indicou serem necessários à identificação do utilizador.
  - O parâmetro "*signature*": corresponde à assinatura do valor enviado no parâmetro "*id_attrs*", criado usando a chave privada do utilizador, cuja chave pública corresponde à chave identificada pelo parâmetro "*user_id*".

- O parâmetro "*iv*": corresponde ao valor do *IV*, codificado em *Base64*, que foi usado no processo de cifra do conteúdo cifrado a ser enviado.

Quando o *IdP* recebe este pedido, começa por analisar qual a chave associada ao identificador referido através do parâmetro "*client*". De seguida, com essa chave e o valor de *IV*, procede à execução da decifra do conteúdo cifrado da mensagem, obtendo os quatro parâmetros apresentados. Posto isto, obtém a chave pública correspondente ao utilizador referido no parâmetro "*username*"e ao identificador referido no parâmetro "*user_id*". Caso não possua uma chave pública que corresponda a estes dois parâmetros, responde à *helper application* com o código de erro *HTTP 424 Failed Dependency*, indicando que esta chave não existe e que a *helper application* deve autenticar o utilizador usando o protocolo *ZKP* e, de seguida, enviar uma chave pública. Por outro lado, caso o *IdP* possua uma chave pública correspondente a estes dois parâmetros, mas esta já tenha expirado, responderá com um código de erro *HTTP 410 Gone*, indicando uma vez mais que a *helper application* necessita de autenticar o utilizador e enviar uma nova chave pública. Por último, caso possua uma chave pública e esta seja válida, irá começar por verificar se a assinatura é válida de acordo com a chave pública que possui e com o valor do parâmetro "*id_attrs*"(que como já vimos, foi o valor sobre o qual foi criada a assinatura). Caso a assinatura seja válida, irá descodificar o valor dos atributos pedidos e analisar quais tem a capacidade de dar resposta. Na nossa implementação do *IdP*, caso ele receba atributos para os quais não seja capaz de responder, irá ignora-los e responder apenas aos que tem a capacidade de dar resposta, que no nosso caso é apenas o atributo "*username*"[3].

Obtidos os valores dos parâmetros pedidos, o *IdP* começa a preparar a resposta a ser dada à *helper application*. Para isso, codifica em *Base64* o resultado do *JSON dump* da resposta aos atributos (parâmetros e valores correspondentes) e assina este conteúdo com a sua própria chave privada (para a qual o *SP* possui a correspondente chave pública). De seguida, cria uma chave *AES* e um valor de *IV*, ambos de forma aleatória, usando estes para cifrar o valor da resposta aos atributos e a assinatura criados. De seguida, cifra o valor desta nova chave, usando a chave pública do utilizador (que antes tinha sido usada para validar a assinatura do pedido dos atributos). Os conteúdos e a ordem pela qual são cifrados e introduzidos na mensagem final podem ser verificados esquemáticamente na imagem da figura 2. Por fim, cria um novo valor de *IV*, e cifra os três conteúdos cifrados indicados acima usando a chave partilhada entre ele e a *helper application* (de forma a que o conteúdo seja transmitido no túnel seguro criado entre os dois).

Quando a *helper application* recebe a resposta do *IdP*, poderá ter dois comportamentos distintos:

- Caso a resposta seja de erro: irá iniciar a autenticação do utilizador com o *IdP*, através do protocolo *ZKP*, a ser apresentado na secção 3.
- Caso a resposta não seja de erro, irá começar por decifrar o conteúdo cifrado obtido, usando a sua chave partilhada com o *IdP* e o *IV* transmitido na resposta. Daqui, como visto, irá obter 3 valores: um texto cifrado correspondente à cifra da chave usada pelo *IdP* para cifrar os atributos e a assinatura, o valor de *IV* usado para cifrar esses conteúdos e o próprio conteúdo cifrado dos atributos e assinatura. Desta forma, começa por usar a chave privada do utilizador, que só ela possui, para decifrar o valor da chave *AES*. Com esta e com o valor de *IV*, decifra de seguida o texto cifrado correspondente ao valor dos atributos e da assinatura dos mesmos. Obtido o valor dos mesmos, pede novamente permissão ao utilizador para os enviar ao *SP*, como iremos apresentar na secção 4 e, de seguida, envia-os ao *SP*.

O processo de envio dos atributos ao *SP* é feito através do envio dum formulário que se auto-submete assim que é carregado pelo *browser*, enviando um pedido *POST*[4] desses atributos para

[3] Contudo, seria de fácil implementação adicionar mais atributos possíveis ao *IdP* criado, havendo uma indicação no código fonte onde introduzir os mesmos e a lógica para obter os seus valores.

[4] Similar ao *POST binding* através do qual a resposta *SAML* era enviada do *IdP* para o *SP* no trabalho apresentado em [1].

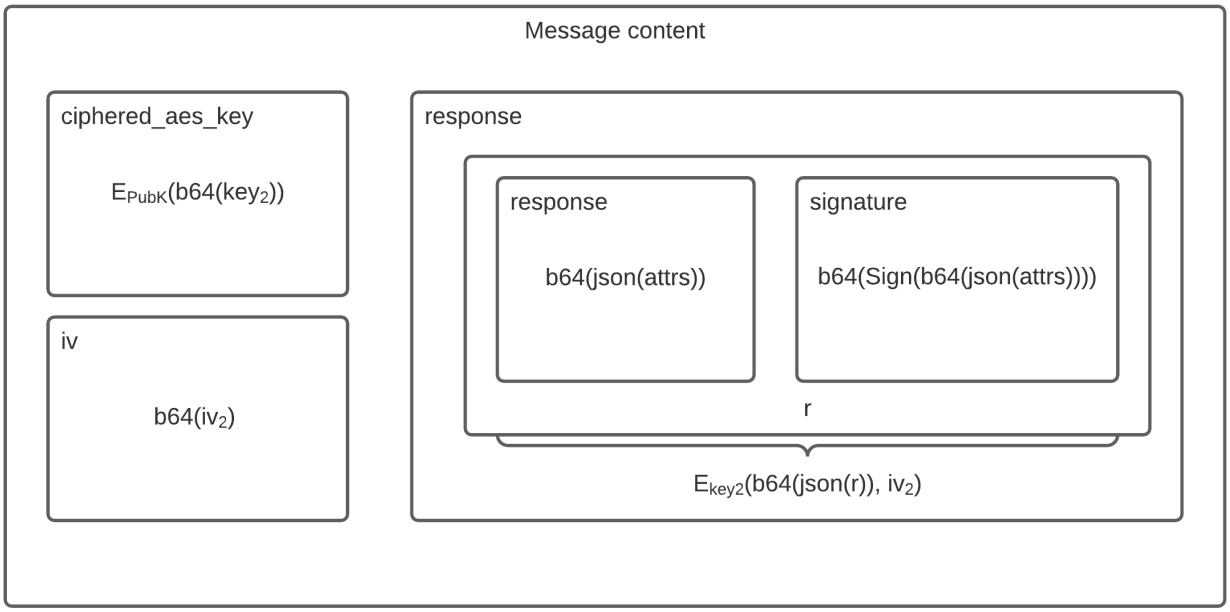

Figura 2: Esquema dos conteúdos da mensagem com os atributos de identidade pedidos enviada pelo *IdP* à *helper application*.

o *URL* inicialmente indicado pelo *SP* (valor do parâmetro "*consumer_url*"enviado por ele no seu redirecionamento inicial). Sendo que quer o envio do formulário para o utilizador, quer o pedido *POST* feito por ele, são dirigidos pelo *browser* do utilizador (e não por uma comunicação direta entre o *SP* e a *helper application*), é tido em conta que existe um túnel seguro *HTTPS* que protege toda a comunicação e garante a cifra de todos os dados enviados e recebidos pelas entidades. Na listagem 3 é possível encontrar o exemplo do conjunto de parâmetros enviados no pedido *POST* para o *SP*. O parâmetro "*response*"contém o *JSON dump* dos valores dos atributos pedidos, codificado em *Base64* e o "*signature*"corresponde à assinatura do valor enviado em "*response*", criada previamente pelo *IdP* com a sua chave privada. Estes dois parâmetros, como já apresentado, foram préviamente obtidos pela *helper application* do *IdP* e enviados como recebidos[5] para o *SP*. Por fim, é também enviado o parâmetro "*client*", que como podemos reparar possui um valor igual ao do parâmetro *client* recebido quando o *SP* pediu que o utilizador fosse identificado, como demonstrado na listagem 1.

```
response="eyJ1c2VybmFtZSI6ICJlc2NhbGVpcmEifQ=="
signature="T-1tbJEOhGNdHTKTpDXh59y4VFuX0WxFkdvjMAdxAfqta0Vrm52UmccugLLTI6MfzaDeTyDthtfZUBsucCxVMsVBoKPH9wRiGkphL5NgZBdXV
    ↪ TajSlGm7ODR3l4ZKGG4iCal3za2fL2QYkfABDuHZQj6Z4eOHz9RVCcTYp1PjJlTwOAPkqkDCBQNeCroH_czuzAgF0s4UjASmKpeeKNCuiK6SfRKW
    ↪ vWqLIDtCZ-50FuCSDgsSkfyrIMae_DDBznGHGhycQ79XDIQrA0AxqUMocyyYPmvlRSLS61zhzgKwQRILy6B15vOhoLRZtxwgnTH8tXpry0XRq13a
    ↪ I0gUYsadg=="
client="300739d4-a21e-45ff-bfcb-f34eb933070d"
```

Listagem 3: Parâmetros do pedido *POST* à resposta da *helper aplication* ao *SP* com os valores dos atributos pedidos.

Ao receber este pedido, o *SP* começa por verificar se possui alguma comunicação pendente com o cliente indicado no parâmetro *client* e em caso afirmativo verifica a assinatura da resposta obtida. No decorrer desta verificação, começa por obter a chave pública do *IdP* (que tinha enviado à *helper*

[5] Isto é, não efetua nenhuma alteração nos valores destes dois parâmetros entre os processos de receção do *IdP* e envio para o *SP*, de maneira a não introduzir nenhum problema na descodificação dos mesmos e na validade da assinatura criada.

application para esta identificar o utilizador), obtendo-a do certificado público deste, que possui armazenado localmente. Tendo agora a chave pública, verifica a assinatura recebida perante a resposta obtida. Caso a assinatura seja válida, o utilizador é identificado com sucesso e é-lhe permitido acesso aos seus recursos pessoais no *SP*. Caso contrário, apresenta uma mensagem de erro ao utilizador.

Desta maneira, com o processo descrito, é possível haver uma identificação do utilizador, sem que neste processo o *IdP* tenha qualquer conhecimento do *SP* que fez o pedido de identificação, uma vez que nenhum dado que possa permitir identificar o *SP* é enviado, no processo, ao *IdP*. Para além disso, e como iremos também melhor explorar na secção 4, o utilizador tem conhecimento absoluto sobre os atributos que são pedidos pelo *SP* ao *IdP* e quais os valores que são enviados como resposta, tendo a possíbilidade de terminar o processo de identificação em qualquer uma destas fases, caso não esteja de acordo com alguma destas informações ser pedida ou partilhada.

3 Protocolo de autenticação

O protocolo de autenticação implementado na nossa solução apresenta, uma vez mais, duas variantes: um protocolo baseado em *Zero Knowledge Proof*, ou *ZKP*, e um protocolo baseado na utilização de criptografia assimétrica. Sobre o primeiro não nos iremos extender muito, uma vez que a sua implementação é similar à realizada no trabalho apresentado em [1]: é recebido, no redirecionamento que o *IdP* faz para a *helper application*, o intervalo de iterações que o ele aceita, como demonstrado na secção 2 e, de seguida, a *helper application* verifica se esse intervalo é intercetado com o que ela aceita; em caso positivo, ela define o número de iterações a serem usadas, informando o *IdP* deste valor e, posteriormente, as duas entidades começam uma série de desafios e respostas baseados num segredo que ambas partilham, que neste caso é a palavra-passe do utilizador, um número de vezes dado pelo número de iterações acordado entre as duas entidades. É de relembrar que neste protocolo as duas entidades se autenticam mutuamente, estabelecendo uma cadeia de confiança entre as elas durante o protocolo e, caso uma deixe de confiar na outra, o protocolo continua como expectável, mas sem que a entidade que desconfia dê informação "desnecessária"sobre o segredo partilhado à outra. É também de realçar que neste conjunto de desafios e respostas, a *helper application* acede iterativamente ao *URL* do *IdP* correspondente ao valor de "*auth_url*", apresentado na secção 2, enviando o seu desafio e a resposta ao desafio recebido na iteração anterior com o *IdP*, e o *IdP* responde a cada chamada com a sua resposta a este novo desafio e com um novo desafio para a *helper application* responder na próxima chamada. Para informação mais detalhada sobre este protocolo, pode ser consultada a secção 3.1 do documento [1].

Após uma autenticação bem sucedida realizada com o protocolo *ZKP*, a *helper application* acede ao *URL* correspondente ao valor de "*save_pk_url*", apresentado na secção "2", de forma a enviar uma chave pública pertencente ao utilizador e à *helper application* em questão. Desta forma, tal como em [1], começa por criar um par de chaves *RSA*, de forma aleatória, armazenando localmente e de forma segura a chave privada correspondente e enviando a chave pública para o *IdP*, fazendo um pedido ao *URL* mencionado. De relembrar que o envio da chave pública e a resposta dada pelo *IdP* ao mesmo são feitas através do túnel seguro que foi usado para toda a comunicação anterior (ou seja, é usado o túnel onde é usada a chave partilhada entre as duas entidades) e, para além disso, toda a comunicação é também cifrada usando uma chave obtida de forma determinística a partir das múltiplas respostas aos desafios enviados entre as duas entidades durante o protocolo *ZKP*. Para mais informações sobre estes tópicos, é favor consultar as secções 3.3 e 4 do documento [1].

Finalmente, o segundo protocolo de autenticação, baseado em criptografia assimétrica, é realizado com base no par de chaves criadas como acima referido. No trabalho anterior, tinha também sido criado um protocolo de autenticação baseado em chaves assimétricas, mas que usava um conjunto de desafios e respostas para fazer todo o processo de autenticação. Contudo, nesta nova solução, sendo que este par de chaves já é usado no processo de identificação, como descrito na secção 2, este processo acaba por simultâneamente permitir a autenticação das duas entidades (da *helper application* e do *IdP*). Sendo que a *helper application*, no protocolo de identificação, começa por enviar os atributos assinados com a chave pública e a resposta do *IdP* é enviada cifrada com a correspondente chave privada, cada uma das entidades pode autenticar a outra. O *IdP*, ao verificar a assinatura dos atributos, pode concluir se a *helper application* com quem se encontra a comunicar é ou não quem diz ser, uma vez que só a *helper application* verídica pode possuir a chave privada e, por outro lado, a *helper application*, ao decifrar a resposta do *IdP*, caso consiga decifrar a mesma com sucesso, significa que se encontra a comunicar com o *IdP* verídico, uma vez que só este poderia possuir a chave pública que corresponde à chave privada usada (já que a chave pública foi enviada de maneira segura e privada para o *IdP*). De notar que caso o *IdP* receba uma assinatura inválida, este termina o processo de identificação do utilizador, uma vez que este

não pode ser autenticado com sucesso, respondendo à *helper application* com uma mensagem de erro *HTTP 401 Unauthorized*. Da mesma forma, caso o *IdP* com quem a *helper application* se encontra a comunicar não seja quem diz ser, esta não irá conseguir decifrar o conteúdo da resposta, terminando também o processo de autenticação e identificação do utilizador, apresentando-lhe uma mensagem de erro.

4 Notificação e consentimento do utilizador

Como já referido na secção 2, existem dois momentos distintos em que a *helper application* pede o consentimento ao utilizador, interrompendo momentâneamente o protocolo de identificação. O primeiro é logo após receber o pedido de identificação por parte do *SP*, onde o utilizador é questionado se consente em que os atributos pedidos pelo *SP* sejam, de facto, requisitados ao *IdP*. Na figura 3 é possível encontrar um exemplo da página apresentada ao utilizador neste caso onde, como podemos constatar, é dada também informação da identidade do *SP* que pediu os atributos e do *IdP* a quem estes vão ser requisitados, para além da enumeração dos próprios atributos.

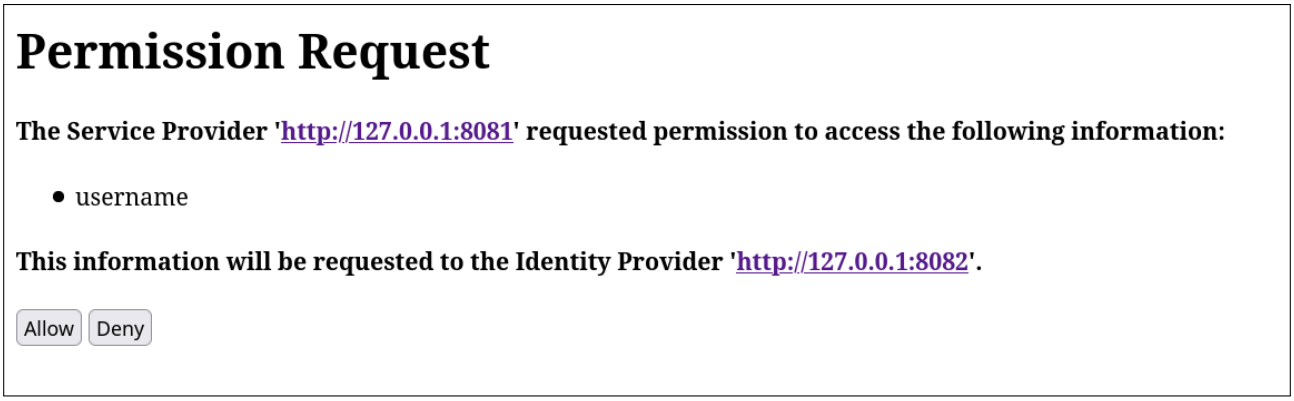

Figura 3: Página apresentada pela *helper application*, de forma a informar e a pedir o consentimento ao utilizador dos parâmetros pedidos pelo *SP* ao *IdP*.

Caso o utilizador aceite, o protócolo de identificação prossegue, sendo que de seguida é feito o pedido ao utilizador para que introduza as credênciais do chaveiro da *helper application*, de maneira a que esta prossiga para a realização dos protocolos de autenticação e identificação. Caso contrário, o processo é terminado e uma mensagem indicativa desse facto é apresentada ao utilizador.

O outro momento em que o utilizador é presenteado com um pedido de consentimento é quando a *helper application* recebe os valores dos atributos pedidos ao *IdP*. Na figura 4 é possível encontrar um exemplo desta página, onde mais uma vez é apresentada a identidade do *IdP* onde foram recolhidos os valores dos atributos e do *SP* para o qual estes irão ser enviados. São também apresentados novamente os atributos e seus valores obtidos do *IdP*.

Caso, uma vez mais, o utilizador aceite em que estes atributos e seus valores sejam enviados ao *SP*, o processo de identificação continua da forma habitual. Caso contrário, é apresentada uma página ao utilizador que o informa que o processo de identificação foi impedido pelo mesmo.

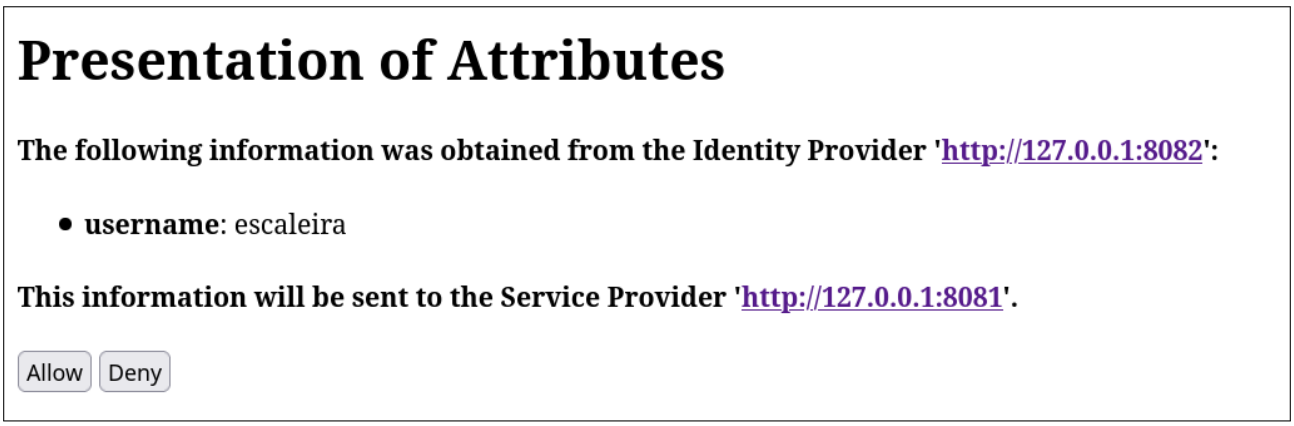

Figura 4: Página apresentada pela *helper application*, de forma a informar e a pedir o consentimento ao utilizador para que os valores dos parâmetros obtidos do *IdP* possam ser enviados ao *SP*.

5 Referências

[1] P. Escaleira, *Autenticação baseada em Zero Conhecimento, com cache de credenciais assimétricas temporárias*, 2021, [Acedido em 24-06-2021]. [Online]. Available: https://github.com/oEscal/zkp/blob/master/report.pdf