Universidade Federal de Santa Catarina

Implementação de *Sticky Policies* em um provedor OpenID Connect

Lucas Maltempi Monfardine

Thiago de Andrade Ávila

2017/1

Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
Curso de Sistemas de Informação

Implementação de *Sticky Policies* em um provedor OpenID Connect

Lucas Maltempi Monfardine
Thiago de Andrade Ávila

Florianópolis - SC
2017/1

Lucas Maltempi Monfardine
Thiago de Andrade Ávila

Implementação de *Sticky Policies* em um provedor OpenID Connect

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Carla Merkle Westphall
Universidade Federal de Santa Catarina

Banca Examinadora:

Gabriel Beims Bräscher
Universidade Federal de Santa Catarina

Gerson Luiz Camillo
Universidade Federal de Santa Catarina

Jorge Werner
Universidade Federal de Santa Catarina

Sumário

Lista de ilustrações

Resumo

Com o crescente mercado de sistemas *web*, deve crescer também o cuidado das organizações com os dados sensíveis dos usuários de seus serviços, em especial dados de identificação, também chamados de PII (*personally identifiable information*), enviados pelos usuários para as suas aplicações. Quando usuários compartilham seus dados com um serviço, devem ter controle sobre o uso destes e certeza que o destino destes dados será cumprido, além das regras quanto ao seu uso ou divulgação a terceiros. As políticas de privacidade são os documentos que destinam-se a ajudar o usuário a entender sobre o tratamento dos dados após a coleta. Através delas, o usuário é questionado ou informado sobre as informações coletadas, o uso dessas informações, o tempo de retenção ou da divulgação das mesmas. O *OpenID Connect*, que conta com diversas implementações *Open Source* e baseado no protocolo OAuth 2.0, visa garantir autenticação e autorização entre um usuário e o serviço desejado. Para tanto, fornece as informações sobre o usuário final na forma de um *token*, que contém as informações básicas de perfil sobre o usuário. Através deste trabalho, foi realizada a implementação de uma extensão em um provedor de identidades que utiliza o protocolo *OpenID Connect*, permitindo o envio de políticas de privacidade no contexto da utilização do serviço, juntamente com as informações básicas contidas no *token*. Para tal, é utilizado o contexto de *Sticky Policies*, que agrega as políticas de privacidade aos dados do usuário normalmente enviados pelo protocolo *OpenID Connect*, permitindo que o usuário de um serviço tenha controle sobre a aplicação e divulgação de seus dados.

Palavras-chave: autenticação, *Sticky Policy*, politica de privacidade, *OpenID Connect*.

Abstract

As the web systems market grows up, organizations concern with services classified users data must grow as well, especially identification data, also called PII - Personally Identifiable Information, which is sent from the users to companies applications. When users share their data with a service, they must have control upon it's destination and make sure it's purpose will be fulfilled, beyond third party usage and disclosure rules. Privacy policy are documents intended to help final users understand the handling given to their data after they've been collected. Through them the user is questioned or informed about collected information, it's usage, retention time or even dissemination. The OpenID Connect, which has some Open Source implementations and is based on OAuth 2.0, aims on guarantee authentication and authorization between a user and the desired service. For such, it provides end-user information in form of a token containing basic information about the user's profile. Through this paper an extension to a OpenID Connect identity provider was implemented, allowing the transmission of privacy policies on service utilization context, together with basic user information incorporated on token. By that, the context of Sticky Policies is used, which adds the privacy policies to the user data normally sent by the OpenID protocol, allowing a user to have control upon application and propagation of it's data while using a service.

Keywords: authentication, Sticky Policy, privacy policy, OpenID Connect.

1 Introdução

Visando mais praticidade na administração de ferramentas e segurança na delegação de permissões, cresce a utilização de ferramentas de autenticação (WANGHAM; DOMENECH; MELO, 2013).

Para gerenciar identidades em ambientes de computação podem ser usados os sistemas de gerenciamento de identidades ou *(Identity Management - IdM)*. Os sistemas de IdM são responsáveis pela criação, gerenciamento, uso das identidades e pela infraestrutura que suporta esse conjunto de processos (BERTINO; TAKAHASHI, 2011).

A principal vantagem na utilização desses sistemas de gerenciamento de identidades é a diminuição da complexidade aos usuários, mitigando a administração de uma quantidade grande de identificações e senhas para acesso aos mais diversos serviços e sistemas (SOUZA; WANGHAM; MELLO, 2014).

Na área de gerência de identidades existem atualmente duas ferramentas que são mais conhecidas e vem sendo bastante utilizadas: *Shibboleth* e *OpenID Connect*. O *Shibboleth* utiliza o padrão *Security Assertion Markup Language* (SAML) para trocar informações de segurança (SWITCH, 2016). O *OpenID Connect* (OPENID CONNECT, 2016), por sua vez, é construído sobre o protocolo OAuth 2.0 (OAUTH, 2016) para prover uma camada de identidade entre o usuário e a aplicação desejada.

Dentre as organizações que utilizam *OpenID Connect* estão o Google e o *Massachusetts Institute of Technology* - MIT (OPENID CONNECT, 2016). Mesmo tendo o mesmo propósito, garantir o acesso de um usuário a um recurso ou serviço, o SAML e o *OpenID Connect* trabalham de forma diferente. Enquanto o protocolo SAML é baseado em XML, o *OpenID Connect* utiliza o formato de dados JSON. Dessa forma, o *OpenID Connect* se torna mais adaptável a sistemas embarcados, aplicativos para dispositivos móveis e aplicações *web*, enquanto o SAML se limita apenas a aplicações *web*. As limitações da tecnologia SAML no suporte a dispositivos móveis e sistemas embarcados, como de *Smart-TVs* por exemplo, está garantindo ao *OpenID Connect* maior aplicação na área de serviços de autenticação (SCHWARTZ, 2016; SALDANHA, 2013).

Existem situações que os usuários necessitam expor seus dados pessoais para que possam ter acesso a um serviço específico. Por exemplo, um usuário de um plano de saúde deve apresentar suas informações pessoais para efetuar seu cadastro. O plano de saúde necessita compartilhar os dados deste usuário com especialistas médicos, profissionais de saúde e empresas farmacêuticas que são aceitas pela rede do plano de saúde. A forma como os dados são compartilhados e as possíveis utilizações dos mesmos pelas terceiras partes, devem ser devidamente descritas e aceitas pelo usuário. Os documentos que definem estas escolhas são chamados políticas de privacidade, políticas de uso ou políticas de acesso,

dependendo do contexto em que são utilizados (PEARSON, MONT, 2011).

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho consiste na utilização de uma adaptação do conceito de *Sticky Policies* afim de permitir a transferência de políticas de privacidade do usuário em um modelo estendido de *token ID*, vinculadas aos dados deste usuário em um provedor de identidades baseado no protocolo *OpenID Connect*.

1.2 OBJETIVOS ESPECÍFICOS

São objetivos específicos deste trabalho:

- Garantir que o provedor de identidade *OpenID Connect* faça o envio do *Token ID* com as políticas de privacidade do usuário junto de seus dados;
- Garantir que o provedor de serviço receba o *Token ID* enviado pelo provedor de identidade contendo, além das informações pessoais do usuário, as políticas de privacidade, e faça o reconhecimento das mesmas;
- Apresentar das políticas definidas pelo usuário na aplicação cliente.

1.3 TRABALHOS RELACIONADOS

Existem diversos trabalhos que tratam da utilização de políticas de privacidade e *Sticky Policies*. Villarreal et. al. (2017) define regras para padronização e apresentação das políticas de privacidade, descrevendo categorias e estruturas para facilitar o uso das políticas. Pearson e Mont (2011) descrevem e exemplificam o conceito e a utilização de *Sticky Policies*, demonstrando seu uso em estudo de caso do Projeto Encore. O projeto provê mecanismos para os usuários definirem e alterarem políticas de consentimento. Além disso, utiliza o conceito de *Sticky Policies* para representar e garantir o consentimento e revocação de preferências do usuário final.

2 Fundamentação Teórica

Neste capítulo serão apresentados conceitos relevantes ao desenvolvimento deste trabalho como Identidade, Gerenciamento de Identidade, *Sticky Policies*, além dos principais objetos de estudo, as políticas de privacidade e o protocolo *OpenID Connect*.

2.1 IDENTIDADE

A identidade de uma organização ou pessoa consiste em características individuais, através das quais essa pessoa ou organização são conhecidas. Esses elementos podem ser adquiridos, tais como nome, endereço, números de registros diversos; ou podem ser inerentes, tal como a biometria (JØSANG; POPE, 2005).

O RFC 2828 (2000) define credencial como o conjunto de dados que são transferidos ou apresentados para demonstrar uma identidade ou as autorizações de uma entidade nos sistemas. Pela definição da ISO/IEC 24760 (2011), credencial é a representação de uma identidade.

Ainda de acordo com a ISO/IEC 24760 (2011), atributos são características ou propriedades que podem ser utilizadas para descrever o estado, aparência ou outros aspectos das entidades. O conjunto dos valores de atributos em uma identidade descrevem a entidade em um domínio.

Além da possibilidade do usuário criar atributos para associar à sua identidade digital, administradores também podem associar atributos à esta, como funções, permissões de acesso e dados de funcionários mantidos por provedores de identidade e atributos, sendo estes frequentemente utilizados no suporte à decisões de autorização e para coletar informações de auditoria (STALLINGS, 2011).

Qualquer elemento característico do usuário ou organização pode ser chamado de identificador quando é usado para fins de identificação. Supõe-se que as identidades são únicas, isto é, não há dois seres humanos ou organizações com a mesma identidade. No entanto, a mesma pessoa ou a mesma organização pode ter diferentes identidades em contextos diferentes (JØSANG et al, 2005).

2.1.1 *PERSONALLY IDENTIFIABLE INFORMATION* - PII

Personally identifiable information (PIIs) são dados que podem ser rastreados para um determinado indivíduo como nome, endereço, número de telefone, número do cartão de crédito, data de nascimento, entre outros. Devido à sua sensibilidade e necessi-

dade de segurança, deve-se ter cuidado no manuseio destes dados, principalmente quando incluem dados financeiros ou médicos.

Em contextos comerciais, consumidores exigem proteção e o uso cuidadoso dos PIIs. Para as organizações, a privacidade destes dados inclui leis, políticas, padrões e processos para gerenciar as PII de um indivíduo (PEARSON; MONT, 2011).

De acordo com Rouse (2014), um PII é considerado sensível quando pode resultar em dano ao indivíduo se a privacidade for violada, ou não sensível, se pode ser transmitido sem criptografia sem causar dano ao indivíduo. Os PII sensíveis, incluem dados médicos, biométricos, informações financeiras e documentos identificadores (CPF e RG, por exemplo). Os PII não sensíveis, por outro lado, são comumente encontrados em registros públicos, abertos em *websites* e listas telefônicas.

2.1.2 Autenticação

A autenticação é o processo formalizado de verificação de uma identidade reivindicada por ou para uma entidade (RFC 2828, 2000). Uma entidade é definida pela RFC 2828 (2000) como um elemento ativo de um sistema, um processo automatizado, um subsistema, uma pessoa ou um grupo de pessoas, que incorporam um conjunto específico de recursos. De acordo com a ISO/IEC 24760 (2011) uma entidade também pode ser definida como um item dentro ou fora de um sistema de informação, como uma pessoa, organização, um dispositivo, um subsistema, ou um grupo de pessoas que têm características distintas.

Os sistemas de informática em geral utilizam um conjunto de mecanismos para identificar entidades (elementos ativos, como usuários) que buscam acesso a objetos (arquivos ou capacidades computacionais, elementos passivos). No âmbito da verificação de identidade, a maioria dos métodos de autenticação baseiam-se em um dos três princípios gerais de identificação (TANENBAUM, 2009):

- Algo que o usuário sabe (na utilização de senhas, por exemplo);
- Algo que o usuário tem (por exemplo, posse de um documento que o identifique) e;
- Alguma coisa que o usuário é, ou alguma característica do usuário (autenticação biométrica).

Esses princípios levam a esquemas de autenticação diversos com diferentes propriedades e complexidades.

2.1.3 Autorização

Segundo Stallings (2011) , o propósito de autorização é garantir acesso a serviços ou recursos específicos baseado na autenticação. Os mecanismos de autorização

são utilizados para determinar o nível de acesso que determinada entidade autenticada deve ter ao serviço ou recurso.

Ainda de acordo com Stallings (2011), autorização contribui para reduzir o risco de exposição a acessos maliciosos.

2.1.4 Provedor de Identidade

Provedores de identidade são normalmente responsáveis pela autenticação e delegação das identidades aos provedores de serviço. São responsáveis pela manutenção e distribuição de atributos para os provedores de serviço e a manutenção de informações atualizadas e fidedignas sobre os usuários.

Um provedor de identidade pode vincular aos atributos de identidade por ele armazenados e providos o valor de atributos de identidade fornecidos por outros provedores de identidade. As credenciais geradas por um provedor podem conter não só atributos deste provedor, mas também fornecidos por outros provedores (BERTINO; TAKAHASHI, 2011).

2.1.5 Provedor de Serviço

É o responsável por prover o serviço aos usuários ou aos agentes que representam os usuários. Uma questão importante que concerne aos provedores de serviço refere-se aos níveis de confiança que estes têm com os provedores de identidade e às credenciais recebidas destes. Em outras palavras, é através da relação entre o provedor de serviço e o provedor de identidade que as informações e permissões de acesso a dados e/ou serviços são atribuídos para cada usuário (BERTINO; TAKAHASHI, 2011).

Jøsang et al (2005) também denominam provedores de serviço como *Relying Parties* (RP), ou Terceiras Partes Confiáveis. Para eles, RPs são as entidades que utilizam os provedores de identidade para verificar a identidade dos usuários em seu processo de autenticação.

2.2 GERENCIAMENTO DE IDENTIDADES

Para Stallings (2011), gerenciamento de identidades é uma abordagem centralizada e automatizada para prover acesso a recursos para usuários autorizados. O objetivo do gerenciamento de identidade é definir uma identidade para cada usuário (humano ou processo), associando atributos à identidade e reforçando a forma que a identidade de um usuário pode ser verificada. Outro conceito importante de um sistema de gerenciamento de identidades é o uso do *Single Sign-On* (SSO), que permite a um usuário acessar todos

os recursos da rede depois de uma única autenticação. Os elementos listados a seguir são considerados os principais em um sistema de gerenciamento de identidades:

- Autenticação: Confirmação que um usuário corresponde ao nome de usuário fornecido.
- Autorização: Garantir acesso a serviços ou recursos específicos baseado na autenticação.
- *Accounting*: processo do registro das atividades do acesso e da autorização, para fins de responsabilização.
- Provisionamento: o registro de usuários no sistema.
- Automação do fluxo de trabalho: movimento de dados em um processo do negócio.
- Administração delegada: O uso de controle de acesso baseado em papéis para garantir permissões.
- Sincronização de senhas: Criação de um processo para SSO ou *Reduced Sign-On* (RSO). Enquanto o SSO permite ao usuário acessar todos os recursos da rede após uma única autenticação, o RSO pode envolver múltiplos logins, mas requer menos esforço por parte do usuário do que se cada recurso e serviço mantivesse seu próprio mecanismo de autenticação.
- Auto-serviço de redefinição de senha: permite ao usuário modificar a própria senha.
- Federação: processo onde a autenticação e a permissão são passadas de um sistema para outro - geralmente entre múltiplas empresas, reduzindo assim o número de autenticações necessárias por parte do usuário.

Um sistema de gerenciamento de identidades também pode incorporar às identidades digitais atributos além das informações de identificação e autorização. Um serviço de atributos gerencia a criação e manutenção destes atributos, como por exemplo o endereço residencial de um usuário. O gerenciamento de identidades permite que o usuário informe este dado apenas uma vez, sendo este mantido em um único lugar e liberado aos recursos que solicitam esta informação de acordo com a autorização do usuário e de políticas de privacidade.

2.2.1 OpenID Connect

De acordo com a documentação da ferramenta, o *OpenID Connect* 1.0 é definido como uma camada de identidade implementada sobre o protocolo *OAuth* 2.0. Ela permite que provedores de serviço, na condição de clientes, verifiquem a identidade de um usuário

final baseando-se na autenticação executada por um servidor de autorização, bem como obter informações básicas sobre o perfil do usuário que está se autenticando no sistema de uma forma interoperável utilizando o conceito de *REpresentational State Transfer* (REST), tecnologia que abstrai detalhes da linguagem utilizada, focando na função dos componentes (OPENID CONNECT, 2016). Sua implementação é dividida em módulos, cada um com uma finalidade específica. A Figura 1 mostra como são divididos os módulos do protocolo *OpenID Connect*, divididos em Mínimo, Dinâmico e Completo.

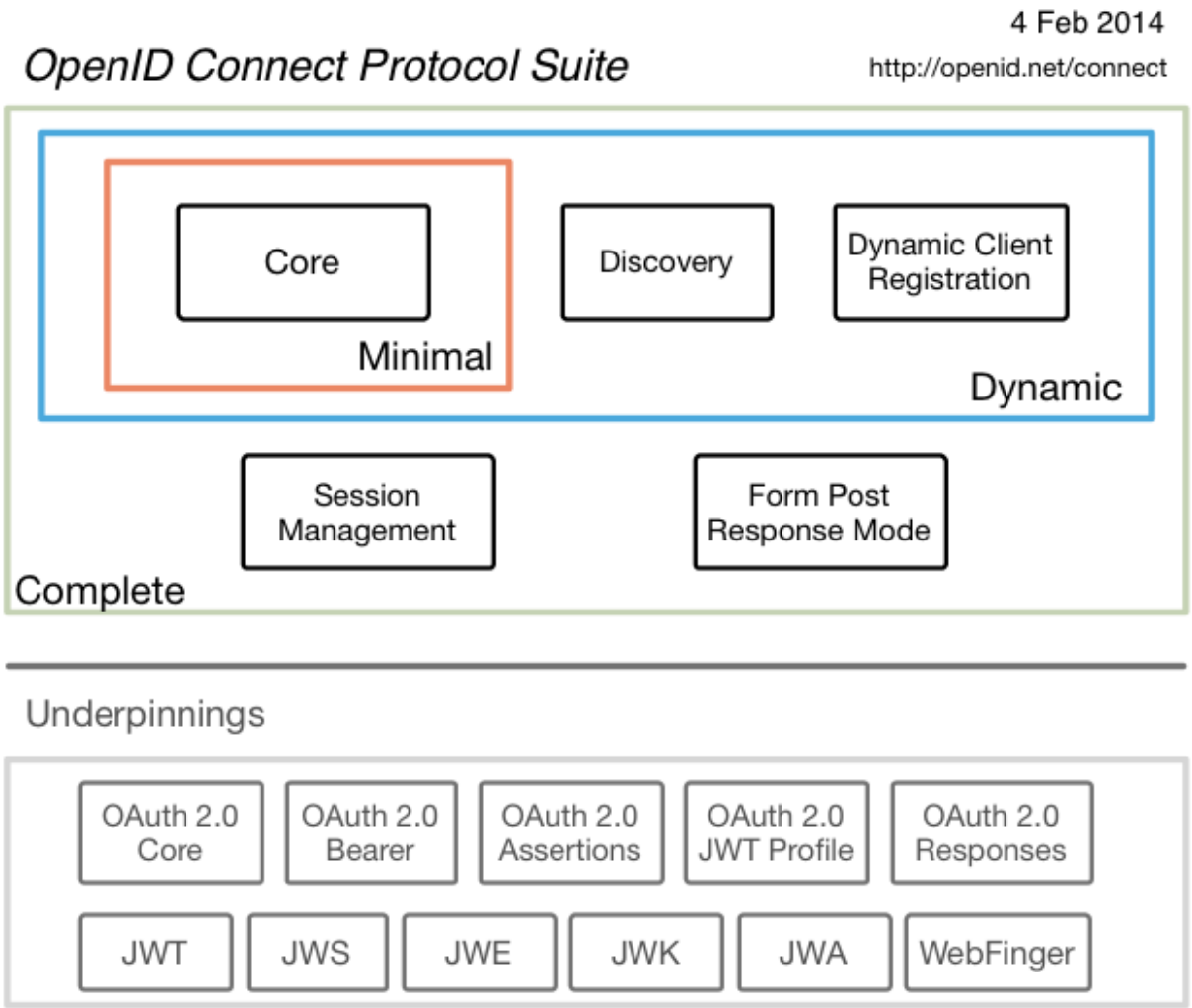

Figura 1 – Diagrama de especificações e guia de implementação (OpenID)

As especificações do módulo *Core* do protocolo *OpenID Connect* 1.0 definem as funcionalidades principais do protocolo: autenticação construída sobre o protocolo *OAuth* 2.0 e o uso de requisições para a troca de informações sobre o usuário final. Além disso, nesse módulo são descritas as considerações sobre segurança e privacidade na utilização do protocolo *OpenID Connect* 1.0.

O *OpenID Connect* 1.0 implementa autenticação como uma extensão do processo

de autorização do protocolo *OAuth* 2.0. O uso dessa extensão deve ser requisitado pelo cliente, incluindo informações *openid* na requisição da autorização. As informações sobre a autenticação realizada são retornadas na forma de uma estrutura JSON *Web Token* (JWT) chamada de Token ID. Servidores de autenticação que implementam o protocolo *OpenID Connect* são também chamados de *OpenID Providers* (OPs) ou provedores de identidade. Clientes *OAuth* 2.0 utilizando *OpenID Connect* são chamados de *Relying Parties* (RPs) ou provedores de serviço.

O JSON *Web Token* (JWT) é definido pelo RFC 7523 (2015) como um *token* de segurança codificado que possibilita o compartilhamento da identidade e de informações de segurança entre domínios de segurança. Um *token* de segurança é geralmente emitido por um provedor de identidade e consumido por uma *Relying Party* (RP), que se baseia em seu conteúdo para identificar o sujeito do *token* para fins relacionados com a segurança. O protocolo *OpenID Connect* segue o fluxo detalhado na Figura 2:

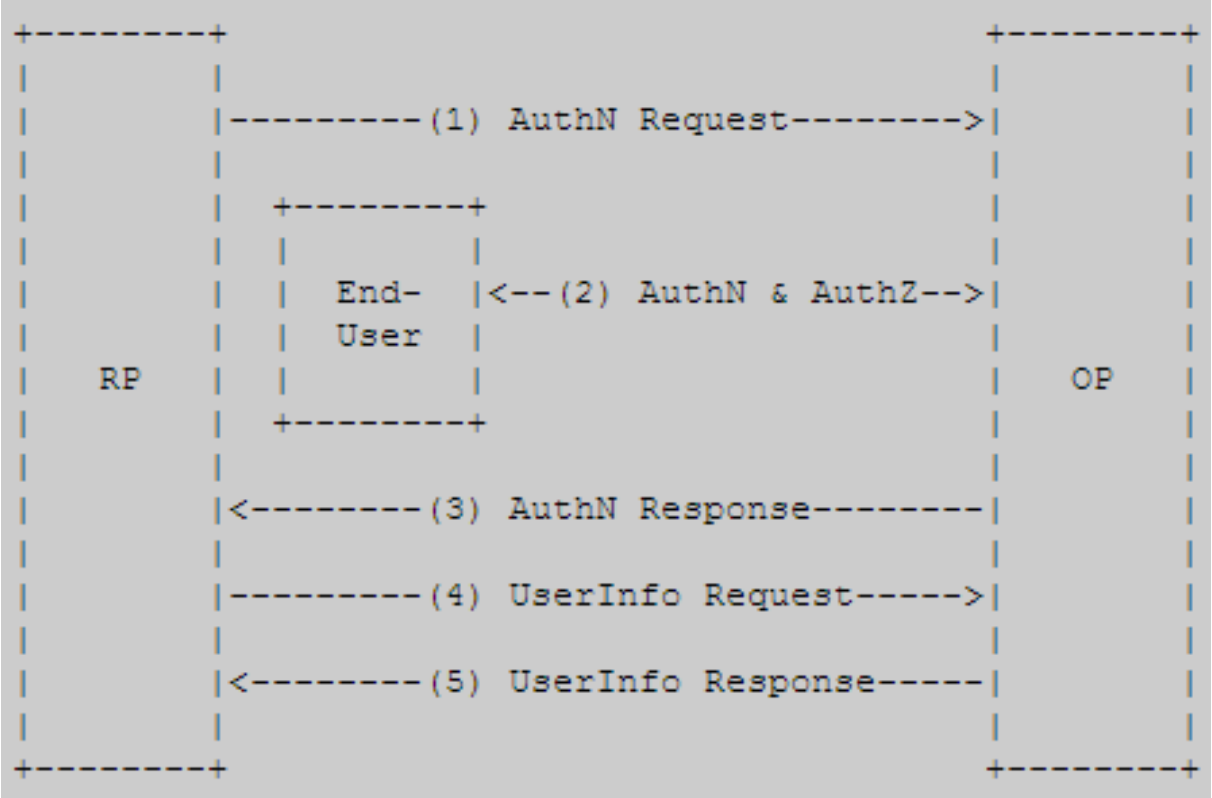

Figura 2 – Fluxo de autenticação através do protocolo OpenID Connect (OpenID)

1. A RP envia uma solicitação de autenticação para o OP;
2. O OP autentica o usuário final e obtém informações de autorização, de acordo com seu consentimento;
3. O OP responde com um *ID token* e, ocasionalmente, um *token* de acesso;

4. A RP pode requisitar ao OP informações sobre o usuário;
5. O OP retorna informações sobre o usuário.

O *Token ID* se assemelha em conceito com uma cartão de identificação no formato JWT. Para a obtenção de um *Token ID* o cliente deve solicitar autenticação ao OP informando o usuário (CONNECT2ID, 2014). Algumas características do usuário estão presentes no *Token ID* e são transmitidas através do JWT, conforme a Figura 3:

```
{
  "sub"       : "alice",
  "iss"       : "https://openid.c2id.com",
  "aud"       : "client-12345",
  "nonce"     : "n-0S6_WzA2Mj",
  "auth_time" : 1311280969,
  "acr"       : "c2id.loa.hisec",
  "iat"       : 1311280970,
  "exp"       : 1311281970,
}
```

Figura 3 – Exemplo de *Token ID* no formato JWT (Connect2ID)

- `sub` - Identificador único do usuário final cadastrado no provedor de identidade;
- `iss` - Especifica a autoridade de emissão;
- `aud` - É gerado para um determinado público alvo, ou seja, o cliente;
- `nonce` - Pode conter um nonce;
- `auth_time` - Pode especificar quando o usuário foi autenticado;
- `acr` - Pode especificar o nível de autorização com que o usuário foi autenticado;
- `iat` - Contém a data em que o JWT foi emitido;
- `exp` - Contém a data de expiração do JWT;
- Pode incluir dados adicionais solicitados sobre o usuário, tais como nome e endereço de e-mail;

- É assinado digitalmente, para que possa ser verificado pelos destinatários;
- Pode ser opcionalmente criptografado para confidencialidade.

O *OAuth* foi concebido inicialmente como um mecanismo de autorização para recursos protegidos ou APIs web. Existem maneiras diferentes de um RP solicitar um Token ID ao OP. Os fluxos para obtenção dos *tokens* são variados e projetados para os diversos tipos de aplicativos, como aplicativos móveis, aplicativos tradicionais baseados em servidores web, entre outros. Os fluxos utilizados no *OpenID Connect* para adquirir os *Tokens ID* são classificados em *Authorization Code Flow*, *Implicit Flow* e *Hybrid Flow*:

Authorization Code Flow - No fluxo de código de autorização, o OP envia um código ao RP, que o utiliza para receber um *Token ID*. É adequado para clientes que possam manter segurança entre eles e o servidor de autorização.

Implicit Flow - Ideal para aplicativos baseados em navegador, usando linguagem de *script* que são executados diretamente pelo navegador. O *Token ID* é recebido diretamente com o redirecionamento do provedor.

Hybrid Flow - raramente utilizado, permite que os aplicativos executados diretamente pelo navegador *(scripts)* ou que são executadas por aplicativos web tradicionais e aplicativos para dispositivos móveis recebam os *Tokens* separadamente um do outro. Essencialmente uma combinação dos fluxos anteriores.

2.2.1.1 *CLAIMS*

Uma *claim* (ou reivindicação) é uma afirmação de que um sujeito, pessoa ou organização, faz sobre si mesmo ou sobre outro assunto. Por exemplo, a declaração pode ser sobre um nome, grupo, preferência de compra, etnia, privilégio, associação ou capacidade. O sujeito que faz a reivindicação ou reivindicações é o provedor.

As especificações do *OpenID Connect* definem um pequeno conjunto de *claims* como padrão. Outras *claims* podem ser usadas em conjunto com as reivindicações padrão. Ao usar tais *claims*, é recomendado que os nomes utilizados sejam resistentes a colisão (OPENID CONNECT, 2016).

2.3 POLÍTICAS DE PRIVACIDADE

De acordo com Caramujo e Silva (2015), políticas de privacidade representam os termos que um usuário precisa aceitar para utilizar os serviços providos por uma organização. É definido por um conjunto de atributos e composto por um ou mais elementos chamados *Statements*, que são as afirmações que detalham as escolhas do usuário sobre o destino e tratamento dos dados. Este documento deve identificar quais os tipos de informações dos usuários serão gerenciadas e podem ser expostas.

Os *Statements*, segundo Caramujo e Silva (2015) e Basso et. al. (2015), são os elementos centrais da política de privacidade, pois descrevem as regras e ações especificadas na política e podem ser divididos em 4 elementos especializados:

Collection, que define qual informação pessoal será coletada pelo provedor de serviço;

Disclosure, que trata sobre a divulgação dos dados e o destino dessa divulgação;

Retention, que identifica o tempo que a informação será mantida;

Usage, que especifica como os dados privados serão utilizados;

O trabalho de Caramujo e Silva (2015), ainda apresenta o conceito de *Informative*, que descreve *statements* genéricos e informativos.

Villarreal et. al. (2017) estabelecem as regras de política de privacidade, definindo inicialmente tuplas contendo os itens: Tipo de Dados (dT), propósito de uso para o dado (pR), tempo em horas de conservação dos dados (tM), o contexto da utilização dos atributos (cN), informação sobre a notificação do usuário quanto ao uso do dado (nT), informação sobre criptografia dos dados (cP). Esta tupla é montada tendo como destino um Provedor de Serviço específico. Para tanto, apresenta-se da seguinte forma:

$$SP.Regra = [dT, pR, tM, cN.nT, cP]$$

onde:

- dT - tipo de dados (nome, CPF, endereço)
- pR - propósito (melhoria de serviço, científico, comercial, governamental)
- tM - tempo de conservação dos dados
- cN - contexto (Financeiro, Compras online, Saúde, Militar)
- nT - notificação (1 para notificar, 0 para não)
- cP - criptografia(1 para cifrar, 0 para não)

Exemplo 1:

$$Loja1.Regra = [nome, cpf; pagamento; 48; compras; 1; 1]$$

No exemplo 1, os tipos de dados (nome e CPF) devem ser utilizados apenas para fins de pagamento, pela Loja1 (Prestador do Serviço), mantendo esses dados por 48

horas, sendo utilizados apenas para compras online, notificando o usuário quanto ao uso e transmitindo-os de forma cifrada.

Para uma melhor apresentação e padronização, VILLARREAL et. al. (2017) definem categorias e valores padrão para cada uma delas. Além disso, estende a estrutura, para o refinamento das preferências pessoais dos usuários, adicionando beneficiários.

Tipo de Dados são divididos em:

- Informação Pessoal (PI)
- Preferências e Características Pessoais (PCP)
- Localização (LO)
- Atividades e Hábitos (AH)
- Relações (RS)

Os propósitos de uso são divididos em:

- Melhoria de Serviço (SI)
- Científico (SC)
- Comercial (CO)

Os beneficiários são divididos em:

- PII Principal (PP)
- Provedor de Serviço (SP)
- Terceira Parte (TP)

Contexto, dividido em:

- Financeiro (FI)
- Compras Online (CO)
- Saúde (SA)
- Militar (MI)

Disseminação de dados: 0 ou 1
Criptografia: 0 ou 1

2.3.1 Sticky Policies

As *Sticky Policies*, em seu contexto original, são as políticas de privacidade que são compartilhadas, quando estas são anexadas juntamente com os dados e que descrevem como estes devem ser tratados. Ou seja, são as restrições ou condições que definem a autorização dos usuários quanto à utilização destes dados (PEARSON, MONT, 2011).

Em *Privacy Patterns* (2017), tem-se que as *Sticky Policies* permitem a propagação das políticas entre organizações de confiança, mantendo a correta aplicação das mesmas, além de prover rastreabilidade. Por outro lado, *Privacy Patterns* (2017) declara como problemático o aumento no tamanho dos dados e como principal obstáculo para a implementação, a dificuldade na atualização das políticas.

A 4, mostra a propagação de políticas, juntamente com os dados, configurando o conceito de *Sticky Policy* (TANG, 2008).

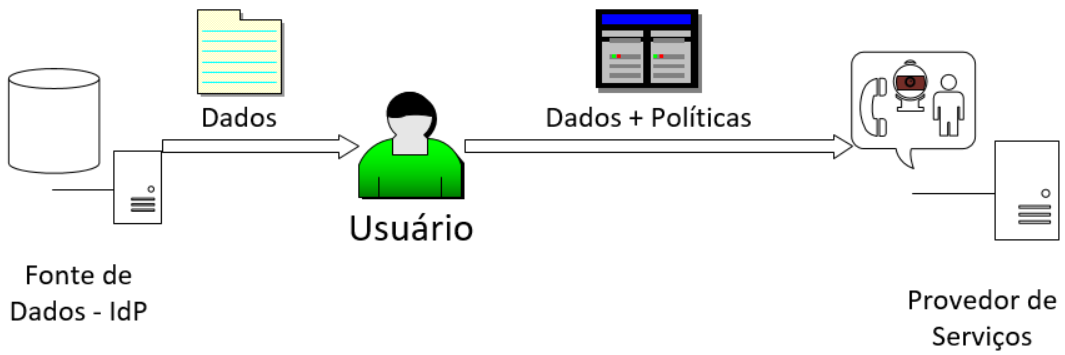

Figura 4 – Representação de esquema de *Sticky Policy*

2.4 FERRAMENTAS DE DESENVOLVIMENTO

Neste seção serão apresentadas informações e características sobre as ferramentas e tecnologias utilizadas durante a etapa de desenvolvimento do projeto.

2.4.1 JavaScript

De acordo com Flanagan (2006), *JavaScript* é uma linguagem de programação interpretada com capacidade de orientação à objetos. O núcleo da linguagem *JavaScript* se assemelha a C, C++ e Java, com arquiteturas de programação como as declarações de *if*, *while* e operadores como o `&&`. Contudo, a similaridade se limita à sintaxe. *Ja-*

vaScript é uma linguagem de tipagem fraca, ou seja, suas variáveis não precisam de um tipo declarado. Objetos em *JavaScript* mapeiam o nome das propriedades para valores proprietários arbitrários. Dessa forma, esses objetos se assemelham mais a tabelas *hash* ou *arrays* associativos do que *structs*, como é o caso da linguagem C, ou objetos, como em C++ ou Java. O mecanismo de herança de orientação a objetos do *JavaScript* é baseado em um modelo de protótipo. Esse modelo é bastante diferente da herança em C++ ou Java. Assim como a linguagem *Perl*, o *JavaScript* é interpretado e se assemelha bastante à essa linguagem em diversas áreas, como o tratamento de expressões regulares e manuseio de *arrays*. O núcleo do *JavaScript* suporta números, *strings* e valores booleanos como tipos primitivos de dados, além do suporte para *arrays*, datas, e objetos de expressão regular.

O *JavaScript* é mais comumente utilizado em navegadores, e nesse contexto, seu núcleo é estendido com objetos que permitem à *scripts* interagirem com o usuário, controlar o navegador e alterar o conteúdo do documento exibido na janela do navegador.

2.4.1.1 Node.js

De acordo com Dayley (2014), Node.js é uma plataforma de desenvolvimento baseada no mecanismo Google V8 *JavaScript*. O código Node.js é escrito em *JavaScript* e então o mecanismo V8 o compila em linguagem de máquina para ser executado. A maioria - se não todo - o código de um serviço pode ser escrita em Node.js, desde o servidor *web*, *scripts* do lado do servidor e qualquer funcionalidade de aplicação web. Uma das características do Node.js é a grande integração entre aplicações *web* e *scripts*, devido ao fato de estes serem executados no mesmo ambiente.

Algumas características da plataforma Node.js:

- ***JavaScript* de ponta a ponta:** uma das maiores vantagens do Node.js é a possibilidade de escrever *scripts* tanto para o servidor quanto para o cliente em *JavaScript*. Isso elimina a dificuldade de alocar parte da lógica de um sistema no cliente e parte no servidor, considerando que os *scripts* do cliente são totalmente compatíveis com o servidor, já que ambos utilizam a mesma linguagem.
- **Escalabilidade orientada em eventos:** O Node.js aplica uma lógica única no tratamento de requisições. Ao invés de ter múltiplas *threads* aguardando por requisições *web*, o Node.js processa todas as requisições na mesma *thread*, usando um modelo de eventos básicos. Isso permite que *webservers* Node.js sejam ampliados de forma que modelos tradicionais de *webserver* não conseguem.

- **Extensibilidade:** O Node.js possui uma grande e ativa comunidade de desenvolvimento, contando com novos módulos que permitem estender suas funcionalidades com frequência. Também é de fácil instalação e inclusão de módulos, passos que podem ser feitos rapidamente.

- **Rápida implementação:** Configurar e iniciar o desenvolvimento em Node.js são atividades simples, como ter um servidor *web* pronto para desenvolvimento de forma instantânea.

2.4.2 NoSQL

NoSQL é um modelo de banco de dados não relacional de alto desempenho. Um banco de dados NoSQL utiliza diversos modelos de dados, como documentos, gráficos, e elementos chave-valor colunares e tem grande reconhecimento pela facilidade de desenvolvimento, desempenho escalável, alta disponibilidade e resiliência. Geralmente um banco de dados NoSQL não faz a utilização de um *schema*, tendo seus dados armazenados de forma semiestruturada, em modelos JSON, XML ou similares que façam a relação entre atributo e valor (Amazon AWS, 2017).

2.4.2.1 MongoDB

Segundo Dayley (2014), o MongoDB é um sistema ágil e altamente escalável de base de dados NoSQL. Seu nome é derivado de "*humongous*", (humilde), enfatizando a escalabilidade e performance que este provê. O MongoDB provê um grande *backend* de armazenamento para sites com alto tráfego e que precisam armazenar dados como comentários de usuários, *blogs* ou outros itens, devido ao fato de ser rapidamente escalável e fácil de implementar.

Algumas características do MongoDB, que garantem boa combinação com Node.js:

- **Orientado a documentos:** Como o MongoDB é orientado à documentos, os dados são armazenados na base de dados em um formato muito próximo do que é utilizado tanto nos scripts do cliente quando do servidor. Isso elimina a necessidade de transferir os dados de linhas para objetos e vice-versa.

- **Alta performance:** MongoDB é uma das bases de dados com mais alta performance disponíveis atualmente, e se destaca no cenário atual, onde há cada vez mais interação com sites e tráfego de dados pesado.

- **Alta disponibilidade:** Seu modelo de dados torna fácil manter a escalabilidade enquanto mantém a alta performance.
- **Alta escalabilidade:** Sua estrutura simplifica a atividade escalar horizontalmente o serviço, fragmentando os dados entre vários servidores.
- **Sem risco de injeção SQL:** Devido ao fato de os objetos serem armazenados como documentos ao invés de utilizar strings SQL, não há risco de haver injeções SQL na aplicação.

3 Proposta: Políticas de Privacidade no OpenID Connect

O avanço de tecnologias de redes de computadores resultou em um crescimento na disponibilidade de aplicações e serviços remotos. Como decorrência, administradores de sistemas necessitam uma base de usuários própria, disponibilizando acesso aos serviços através de informações e níveis de privilégio (MOREIRA et al., 2011).

De acordo com Stallings (2011), o gerenciamento de identidades usa padrões que são essenciais para uma troca segura de identidades através de diferentes domínios e sistemas heterogêneos. Um dos desafios do gerenciamento de identidades é garantir aos usuários que os dados providos por eles a um serviço serão corretamente tratados pelo serviço e pelas outras partes.

Para que isso ocorra, utilizaremos o conceito de políticas de privacidade e as definições que dela decorram para determinar padrões a serviços a fim de prover um serviço seguro.

A criação das políticas de privacidade é possível através da combinação de todos os tipos de dados, todos os propósitos de uso, todos os beneficiários e todos os contextos descritos anteriormente. Assim são definidas as permissões para a utilização do dado em questão. Utilizando o exemplo descrito anteriormente:

$$Loja1.Regra = [nome, cpf; pagamento; 48; compras; 1; 1]$$

Temos que:

```
{
  PI_CO_TP_CO = 1
  PCP_CO_TP_CO = 0
  PCP_SC_TP_FI = 0

  (demais combinacoes)

  Disseminacao = 1
  Criptografia = 1
}
```

Figura 5 – Modelo de combinações de políticas de privacidade do usuário

Na Figura 5, nome e CPF são dados com tipo Informação Pessoal (PI), o

propósito da utilização deste dado é Comercial (CO), o beneficiário é a Loja1 (TP), o contexto é Compras Online (CO). O valor dessa combinação é 1, pela escolha de autorizar a utilização pelo usuário. Se esse é o único objeto de que trata a política, todas as outras combinações possíveis ficariam com o valor 0. Os valores de Disseminação e Criptografia terão valores únicos (1) para todas as combinações das políticas, pois sua aplicação não está no escopo deste trabalho.

A proposta deste trabalho trata da utilização da organização descrita anteriormente para criação de políticas de privacidade, permitindo ao usuário definir a utilização dos seus dados para diversos contextos diferentes. Após isso, utilizar do conceito de *Sticky Policies* para envio destas mesmas políticas de um provedor de identidades *OpenID Connect* a um provedor de serviço, através da adição de *claims* contendo as políticas definidas pelo usuário.

3.1 Problema

As informações sobre a autenticação realizada utilizando o protocolo *OpenID Connect* são enviadas pelo *OpenID Provider*, na forma de uma estrutura *JSON Web Token* (JWT) chamada de *Token* ID. Este *token* carrega informações da identidade e de segurança, mas não contém dados sobre as políticas de privacidade escolhidas pelo usuário.

3.2 Modelo Inicial

A proposta inicial deste trabalho compreende os seguintes pontos:

- Criação de um perfil de usuário e suas políticas de privacidade, compreendendo todos os tipos de dados, propósitos, contextos de uso, beneficiários, definição sobre a disseminação e criptografia dos dados;
- Definição e criação da estrutura para envio das políticas de privacidade do usuário definidas previamente para os provedores de serviço, juntamente com os dados de identificação;
- Implementação dessa estrutura em um provedor de dados baseado no protocolo *OpenID Connect*, permitindo o envio das políticas com o *Token* ID;
- Implementação de modificações em um modelo de cliente baseado em *OpenID Connect*, permitindo que este receba um *token* contendo, além das informações da estrutura original, as políticas de privacidade definidas pelo usuário.

3.3 Desenvolvimento

A modalidade da pesquisa é experimental, pois através de alterações de código e execuções do sistema, pretende-se realizar alterações em uma implementação existente do provedor de identidades *OpenID Connect* para o envio de políticas de privacidade juntamente com o *Token* ID e alterações em uma implementação existente de cliente *OpenID Connect* para o recebimento e exibição das mesmas políticas.

3.4 Recursos de Software

Foram utilizadas como base para o provedor de identidades e do cliente *OpenID Connect* as implementações de Skokan (2017) em Node.js. O provedor de identidades *OpenID Connect* implementado é certificado como um provedor de identidades oficial *OpenID* desde 02 de Janeiro de 2017. O cliente também é certificado como *Relying Party* pelo *OpenID CERTIFICATION* (2017) desde 15 de Dezembro de 2016.

O provedor de identidades fornece uma implementação das especificações do *OpenID*, sem determinar o modelo de persistência a ser utilizado. Oferece, no entanto, uma implementação de praticamente todo o modelo em um modelo de banco de dados de memória que é instanciado com a execução do provedor, guardando usuários, *tokens*, clientes cadastrados, chaves criptográficas, códigos de autorização (para o fluxo de código de autorização) e informações de sessão.

Para a implementação da persistência destes itens, foi utilizado o MongoDB (MONGODB, 2017), sistema ágil e altamente escalável de base de dados NoSQL. Apenas os usuários continuaram sendo persistidos em memória, através da classe account.js.

Como ferramentas de apoio ao desenvolvimento em Node.js, foram utilizados o IntelliJ IDEA (JETBRAINS, 2017), IDE para várias linguagens de programação e o Sublime Text 3 (SUBLIME TEXT, 2017), editor de texto com diversos recursos de grande utilidade para o desenvolvimento.

No âmbito do Node.js, foram utilizadas bibliotecas diversas para a execução das implementações e o desenvolvimento, como os *frameworks express* e *koa*.

Como mecanismo das telas apresentadas, foi utilizado o EJS, linguagem de modelos *client-side* JavaScript, que faz a produção do HTML.

Figura 6 – Tecnologias utilizadas no desenvolvimento do trabalho

3.4.1 Servidor MongoDB

Para suporte à persistência de dados do trabalho foi utilizado o software MongoDB *Server* em sua versão 3.4.4 para sistema operacional Microsoft® Windows® de 64 bits, executado diretamente no sistema local. Também foi utilizado para apoio ao desenvolvimento e consultas de testes o cliente de linha de comando do próprio pacote MongoDB que acompanha os arquivos do servidor.

Foram criadas na base *mydb*, utilizada para o desenvolvimento, as *collections* *access_token*, *authorization_code*, *client*, *client_credentials*, *refresh_token*, *registration_access_token* e *session* para a persistência destes objetos, necessários à execução do cliente e do provedor de identidades.

3.4.2 Cliente OpenID Connect

Como citado na seção 3.4 - Recursos de Software, o cliente *OpenID Connect* utilizado foi desenvolvido em Node.js por Skokan (2017) e disponível em seu repositório no *GitHub*.

O próximo passo para a implantação do cliente está descrito na Figura 7, com

a definição da porta que o sistema passa a escutar (linha 7). Além disso, é definido o endereço do *Issuer*, emissor das identidades que serão utilizadas pelo cliente. No escopo do desenvolvimento deste trabalho, temos o *Issuer* como o Provedor de Identidade *OpenID Connect*. Na linha 6 é realizada esta definição, alocando no mesmo servidor (*localhost*) a porta 3100 para o provedor de identidades *OpenID* Estas modificações são feitas na classe node-openid-client\example\index.js.

index.js

```
1  'use strict';
2
3  const { Issuer } = require('..');
4
5  const {
6    ISSUER = 'http://localhost:3100/',
7    PORT = 3000,
8  } = process.env;
9
10 const appFactory = require('./app');
11
12 Issuer.discover(ISSUER).then((issuer) => {
13   const app = appFactory(issuer);
14   app.listen(PORT);
15 }).catch((err) => {
16   console.error(err); // eslint-disable-line no-console
17   process.exit(1);
18 });
19
```

Figura 7 – Definição de endereço do OP e porta (Fonte: os autores)

Na classe *node-openid-client\example\app.js*, é definida a requisição de autorização que será enviada ao provedor de identidades. Esta requisição está descrita na Figura 2, como o passo (1). Nela, são adicionadas as reivindicações ou *Claims* que serão tratadas pelo cliente quando este receber a resposta ou *Token* ID, o endereço de redirecionamento (para onde o sistema será redirecionado após a resposta do provedor de identidades), o estado, o *nonce* e demais parâmetros de sessão necessários para a requisição. A composição da requisição é definida na Figura 8, entre as linhas 158 e 167. Uma das adaptações desenvolvidas neste trabalho consiste na adição de um objeto *Claim* "políticas", visto na linha 161, como um dos *Claims* constantes nas informações de usuário (*UserInfo*).

app.js

:lient-master

```
router.get('/login', async (ctx, next) => {
  ctx.session.state = crypto.randomBytes(16).toString('hex');
  ctx.session.nonce = crypto.randomBytes(16).toString('hex');

  const authorizationRequest = Object.assign({
    claims: {
      id_token: { email_verified: null },
      userinfo: { name:null, sub: null, email: null, politicas: null},
    },
    redirect_uri: url.resolve(ctx.href, 'cb'),
    scope: 'openid',
    state: ctx.session.state,
    nonce: ctx.session.nonce,
  }, ctx.session.authorization_params);

  const authz = CLIENTS.get(ctx.session.id).authorizationUrl(authorizationRequest);

  ctx.redirect(authz);
  return next();
});
```

Figura 8 – Adição das reivindicações suportadas pelo cliente (Fonte: os autores)

Retratada na Figura 9 a classe *node-openid-client\lib\client.js*, está o trecho de código que define o passo (4) da Figura 2. Nesta etapa é montada a requisição para *UserInfo* ou informações sobre o usuário. Uma das atividades no desenvolvimento deste trabalho foi a adição de código para incluir na requisição a *claim* de políticas, necessárias para que o cliente solicite ao provedor de identidades *OpenID* informações sobre este campo.

client.js

```
770  authFor(endpoint) {
771    switch (this[`${endpoint}_endpoint_auth_method`] || this.token_endpoint_auth_method) {
772      case 'none' :
773        return {
774          body: {
775            client_id: this.client_id,
776          },
777        };
778      case 'client_secret_post':
779        return {
780          body: {
781            client_id: this.client_id,
782            client_secret: this.client_secret,
783          },
784        };
785      case 'private_key_jwt' :
786      case 'client_secret_jwt' : {
787        const timestamp = now();
788        return this.createSign(endpoint).then(sign => sign.update(JSON.stringify({
789          iat: timestamp,
790          exp: timestamp + 60,
791          jti: uuid(),
792          iss: this.client_id,
793          sub: this.client_id,
794          aud: this.issuer[`${endpoint}_endpoint`],
795          politicas: this.client_id,
796        })).final().then((client_assertion) => { // eslint-disable-line camelcase, arrow-body-style
797          return { body: {
798            client_assertion,
799            client_assertion_type: 'urn:ietf:params:oauth:client-assertion-type:jwt-bearer',
800          } };
801        }));
802      }
803      default: {
804        const value = new Buffer(`${this.client_id}:${this.client_secret}`).toString('base64');
805        return { headers: { Authorization: `Basic ${value}` } };
806      }
807    }
808  }
```

Figura 9 – Informações que serão utilizadas na requisição (Fonte: os autores)

3.4.3 Provedor de Identidades OpenID Connect

No processo de implantação do provedor de identidades deve-se definir uma variável de ambiente com nome *MONGODB_URI* e endereço do servidor MongoDB. A Figura 10 descreve o endereço definido no IntelliJ IDE para o MongoDB.

Appearance & Behavior › Path Variables

Name	Value
KOTLIN_BUNDLED	C:\Program Files\JetBrains\IntelliJ IDEA 2017.1.3\plugins\Kotlin\kotli...
MAVEN_REPOSITORY	C:\Users\ADM\.m2\repository
MONGODB_URI	mongodb:\\\localhost:27017\mydb

Figura 10 – Definição de Variável de Caminho na IDE InteliJ (Os autores)

Na Figura 11, é apresentada a função *if* da classe *node-oidc-provider-master\example\index.*

que trata da existência da variável de ambiente e implementa a utilização do MongoDB como adaptador de persistência, caso resultado positivo da análise condicional (linhas 20 a 23, classe *index.js*). Também na Figura 11, nas linhas 13 e 18 da classe *index.js*, são definidos o endereço e porta utilizadas pelo *issuer* ou emissor de identidades, ou seja, o endereço do próprio provedor de identidades *OpenID Connect*.

index.js

```
1  'use strict';
2
3  /* eslint-disable no-console */
4
5  const Provider = require('../lib');
6  const path = require('path');
7  const _ = require('lodash');
8  const bodyParser = require('koa-body');
9  const querystring = require('querystring');
10 const render = require('koa-ejs');
11 const Router = require('koa-router');
12
13 const port = process.env.PORT || 3100;
14
15 const Account = require('./account');
16 const settings = require('./settings');
17
18 const issuer = process.env.ISSUER || 'http://localhost:3100';
19
20 if (process.env.MONGODB_URI) {
21   const MongoAdapter = require('./adapters/mongodb'); // eslint-disable-line global-require
22   settings.config.adapter = MongoAdapter;
23 }
```

Figura 11 – Definições para utilização do banco de dados MongDB (Fonte: os autores)

Na Figura 12 é apresentada a configuração do *framework express.js* realizada na classe *node-oidc-provider-master\example\express.js*, que fornece um conjunto de recursos para aplicativos *web*, recorrente no código de Skokan, necessário para o bom funcionamento do provedor de identidades desenvolvido. Assim como no cliente, neste ponto é feita a declaração do servidor e porta em que o provedor será executado. Além disso, é na classe *express* é instanciado o objeto referente ao provedor, utilizando deste ponto em diante as rotas definidas pelo Node.js para a execução do mesmo, ou seja, para qual página o usuário será direcionado dependendo da ação tomada.

express.js

```
'use strict';

const Provider = require('../lib');
const express = require('express'); // eslint-disable-line import/no-unresolved

const app = express();

const provider = new Provider('http://localhost:3100/op');

provider.initialize().then(() => {
  app.use('/op', provider.callback);
  app.listen(3100);
});
```

Figura 12 – Definições de configuração do provedor de identidades (Fonte: os autores)

Conforme exposto na Figura 13, a linha 30 da classe *node-oidc-provider-master\lib\helpers\defaults.js* define o nome dos *Claims* ou reivindicações habilitadas no provedor de identidades *OpenID* e que este pode fornecer aos seus clientes. Como desenvolvimento das modificações propostas neste trabalho, está a adição do objeto políticas a esta linha, permitindo aos clientes que as solicitem à este provedor e tornando o provedor apto a entregá-las.

defaults.js

```
/*
 * claims
 *
 * description: list of the Claim Names of the Claims that the OpenID Provider MAY be able to
 *   supply values for
 * affects: discovery, ID Token claim names, Userinfo claim names
 */
claims: { acr: null, auth_time: null, iss: null, openid: ['sub'], politicas: null },
```

Figura 13 – (*Claims* habilitadas no provedor de identidades (Fonte: os autores)

Os escopos no *OAuth* 2.0 definem privilégios de acesso requisitados pela aplicação cliente e autorizados pelo usuário final no provedor de autorização. O *OpenID Connect* usa escopos para definir quais *claims* o cliente está requisitando e que estão sujeitos à autorização e consentimento pelo usuário no *OpenID Provider* (RFC6749, 2012).

Os *endpoints* de recursos protegidos podem executar ações diferentes e retornar informações diferentes com base nos valores de escopo e outros parâmetros usados ao solicitar o *Token* de acesso apresentado.

Os valores de escopo definidos para o provedor de identidades são definidos na classe *node-oidc-provider-master\example\settings.js*, conforme demonstrado na Figura 14.

Entre as linhas 18 e 25 são definidos os valores para os *Claims*, incluindo neste intervalo o *claim* de políticas.

settings.js

```
8   module.exports.config = {
9     acrValues: ['session', 'urn:mace:incommon:iap:bronze'],
10    cookies: {
11      long: { signed: true },
12      short: { signed: true },
13    },
14    discovery: {
15      service_documentation: pkg.homepage,
16      version: pkg.version,
17    },
18    claims: {
19      amr: null,
20      address: ['address'],
21      email: ['email', 'email_verified'],
22      phone: ['phone_number', 'phone_number_verified'],
23      profile: ['birthdate', 'family_name', 'gender', 'given_name', 'locale', 'middle_name', 'name',
24        'nickname', 'picture', 'preferred_username', 'profile', 'updated_at', 'website', 'zoneinfo'],
25      politicas: ['politicas'],
26    },
27    features: {
28      devInteractions: false,
29      claimsParameter: true,
30      clientCredentials: true,
31      encryption: true,
32      introspection: true,
33      registration: true,
34      registrationManagement: false,
35      request: true,
36      requestUri: true,
37      revocation: true,
38      sessionManagement: true,
39      backchannelLogout: true,
40      oauthNativeApps: true,
41      pkce: { skipClientAuth: true },
42    },
43    subjectTypes: ['public', 'pairwise'],
44    pairwiseSalt: 'da1c442b365b563dfc121f285a11eedee5bbff7110d55c88',
45    interactionUrl: function interactionUrl(interaction) { // eslint-disable-line no-unused-vars
46      // this => koa context;
47      return `/interaction/${this.oidc.uuid}`;
48    },
49  };
```

Figura 14 – Valores do escopo para o provedor de identidades (Fonte: os autores)

O provedor de identidades *OpenID* deve de alguma maneira persistir as contas de usuários cadastrados para uso posterior. Esta conta deve ter uma propriedade *accountId*, bem como a função *claims()*, para retornar um objeto com reivindicações que correspondem às reivindicações que ele suporta.

As contas de usuários podem estar salvas em um banco de dados ou em classes com as informações em memória. Para o desenvolvimento deste trabalho, foi utilizada uma classe *account.js*, conforme Figura 15, para definir as informações do usuário de teste. Nele, foram registradas as informações pessoais do usuário, bem como as políticas hipoteticamente definidas por ele.

Na imagem 15, na linha 30 da classe *node-oidc-provider-master\example\account.js* temos uma supressão de políticas para uma melhor apresentação na imagem. Todas as combinações possíveis de tipos de dados, todos os propósitos de uso, todos os beneficiários e todos os contextos descritos anteriormente na proposta deste trabalho devem estar descritas nas políticas, resultando em 180 políticas, além das políticas adicionais de disseminação de dados e criptografia.

Da forma que foi desenvolvida a extensão no provedor, basta que neste ponto sejam alteradas, adicionadas ou removidas políticas de privacidades, para que estas sejam enviadas de forma correta para um cliente que receba a *claim* "políticas" e trate-a como determinado e desejado.

account.js

```
class Account {
  constructor(id) {
    this.accountId = id || uuid();
    store.set(this.accountId, this);
  }

  claims() {
    return {
      address: {
        country: '000',
        formatted: '000',
        locality: '000',
        postal_code: '000',
        region: '000',
        street_address: '000',
      },
      politicas: {
        PI_SI_PP_FI: '1',
        PCP_SI_PP_FI: '1',
        LO_SI_PP_FI: '1',
        AH_SI_PP_FI: '1',
            // Políticas suprimidas
        PCP_CO_TP_MI: '1',
        LO_CO_TP_MI: '1',
        AH_CO_TP_MI: '1',
        RS_CO_TP_MI: '1',
        DISSEMINACAO: '1',
        CRIPTOGRAFIA: '1',
      },
      birthdate: '1987-10-16',
      email: 'teste@teste.com',
      email_verified: false,
      family_name: 'Teste',
      gender: 'male',
      given_name: 'Teste',
      locale: 'en-US',
      middle_name: 'Teste',
      name: 'Teste',
      nickname: 'teste',
      phone_number: '+49 000 000000',
      phone_number_verified: false,
      picture: 'http://lorempixel.com/400/200/',
      preferred_username: 'Jdawg',
      profile: 'https://teste.com',
      sub: this.accountId,
      updated_at: 1454704946,
      website: 'http://teste.com',
      zoneinfo: 'Europe/Berlin',

    };
  }
```

Figura 15 – Políticas definidas no perfil do usuário (Fonte: os autores)

4 Resultados Experimentais

Na Figura 16, é apresentada a tela de *setup* ou configuração de um novo cliente. Neste ponto é definido qual fluxo será utilizado para a comunicação entre o cliente e o provedor de identidades *OpenID Connect*, além das trocas de *tokens* entre eles.

As alterações implementadas por este trabalho permitem a utilização de qualquer um dos fluxos determinados pelo protocolo *OpenID Connect*, com o correto funcionamento do envio das políticas para todos eles.

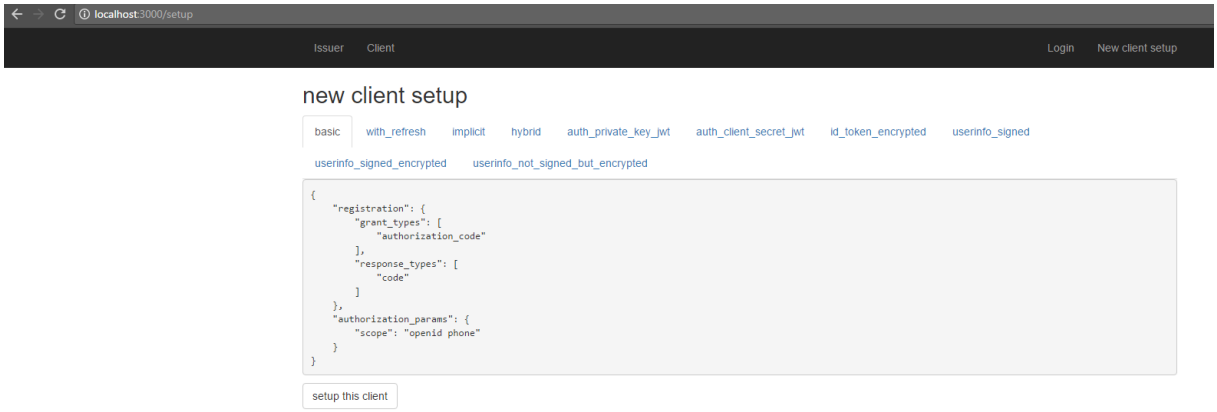

Figura 16 – Configuração de um novo cliente (Tela apresentada no cliente) (Fonte: os autores)

Após a criação e configuração do novo cliente, é apresentada uma tela de descrição do emissor definido no novo cliente, com informações sobre o fluxo a ser utilizado, os métodos de autenticação, o endereço do *endpoint* de autorização, das *Claims* suportadas pelo novo cliente e informações sobre criptografia. Esta tela, apresentada na Figura 17 serve como um registro das informações escolhidas na configuração do novo cliente.

Figura 17 – Registro do emissor definido para o novo cliente (Tela apresentada no cliente) (Fonte: os autores)

Quando o usuário de um cliente registrado no provedor tenta fazer seu *login*, é apresentada a tela de *login*, conforme Figura 18. Além de formulário para a submissão de *login* e senha, são novamente apresentadas informações sobre a comunicação entre provedor e cliente.

Apesar de conterem campo para entrar com seu usuário e senha, este formulário

não faz nenhum tipo de verificação de conta válida para o mesmo. Quaisquer valores adicionados a ele serão remetidos à conta de usuário criada em *account.js*. A verificação de usuário e senha para posterior vínculo com contas reais em um banco de dados é definida como um trabalho futuro e não cabe ao escopo deste trabalho.

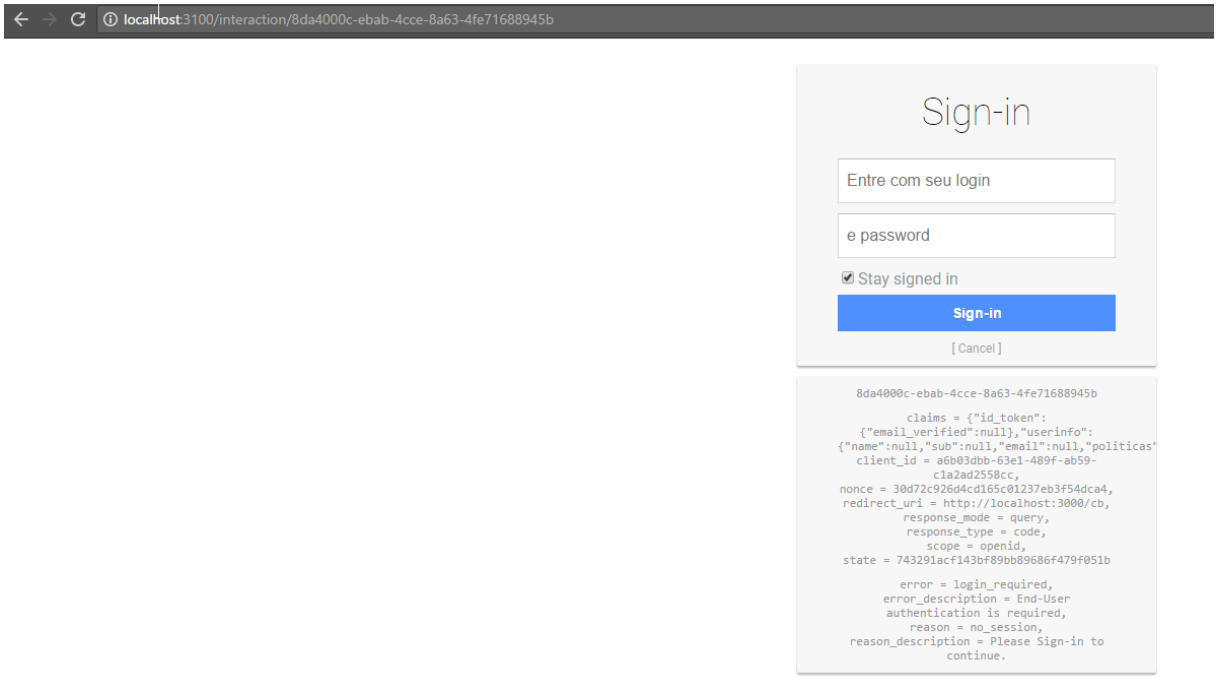

Figura 18 – Tela de login de usuário (Tela apresentada no cliente) (Fonte: os autores)

Se o usuário já efetuou o *login* no provedor de identidades, este fica vigente até que seja feito um *logout* pelo mesmo. Até lá, quaisquer *logins* deste usuário em clientes registrados no mesmo provedor o levarão até a tela de autorização do provedor, utilizando a conta autenticada previamente.

Na Figura 19 foi criado um segundo cliente para logar com o mesmo usuário, sendo novamente direcionado à classe *account.js*, referenciando assim à mesma conta em dois clientes distintos.

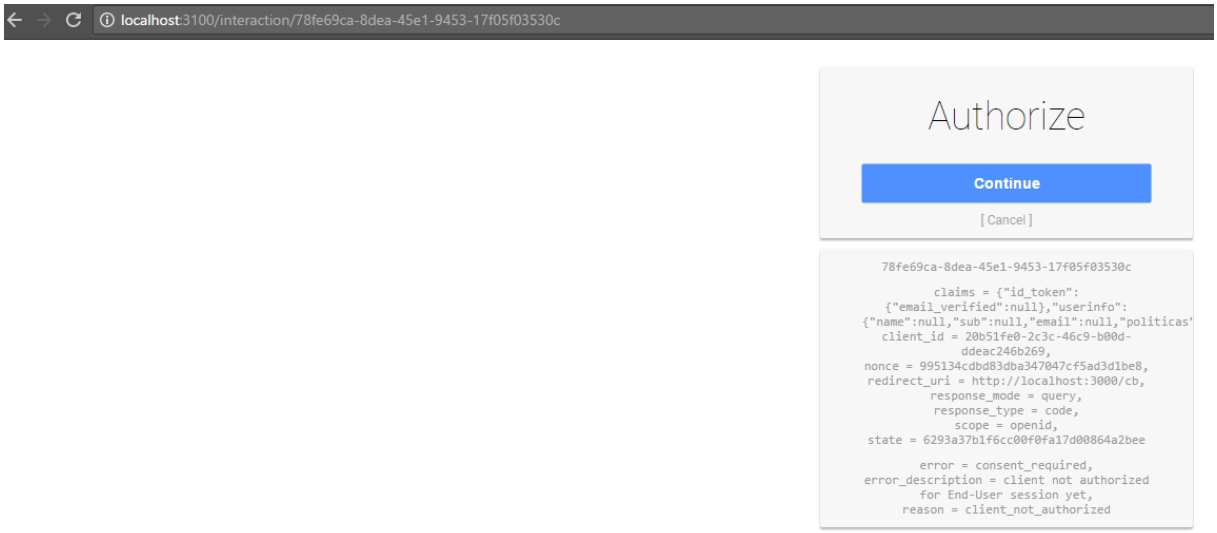

Figura 19 – Tela de autorização para acesso de mesmo usuário em diferente cliente (Tela apresentada no cliente) (Fonte: os autores)

Quando o usuário resolve fazer *logout* de um cliente, é direcionado para a página de *logout* do provedor de identidades, que apresenta a pergunta mostrada na Figura 20, sobre a vontade do usuário de fazer *logout* também do provedor.

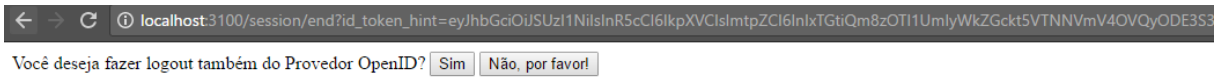

Figura 20 – Tela de autorização para acesso de mesmo usuário em um cliente diferente (Tela apresentada no cliente) (Fonte: os autores)

As figuras 21 e 22 se referem à resposta recebida pelo cliente com os *tokens* ID e de acesso vindos do provedor de identidades *OpenID Connect*. São apresentados os códigos *hash* dos *tokens*, além destes decodificados. Na Figura 23 é apresentado o *UserInfo Response*, que contém as informações pessoais do usuário. Neste mesmo espaço estão as políticas de privacidade que são foco deste trabalho, enviadas juntamente com o *Token* ID. Na imagem, assim como nos dados do usuário em *account.js*, as políticas são suprimidas para melhor apresentação. Apesar disso, na execução, todas as 182 políticas definidas foram mostradas e apresentadas do lado do cliente.

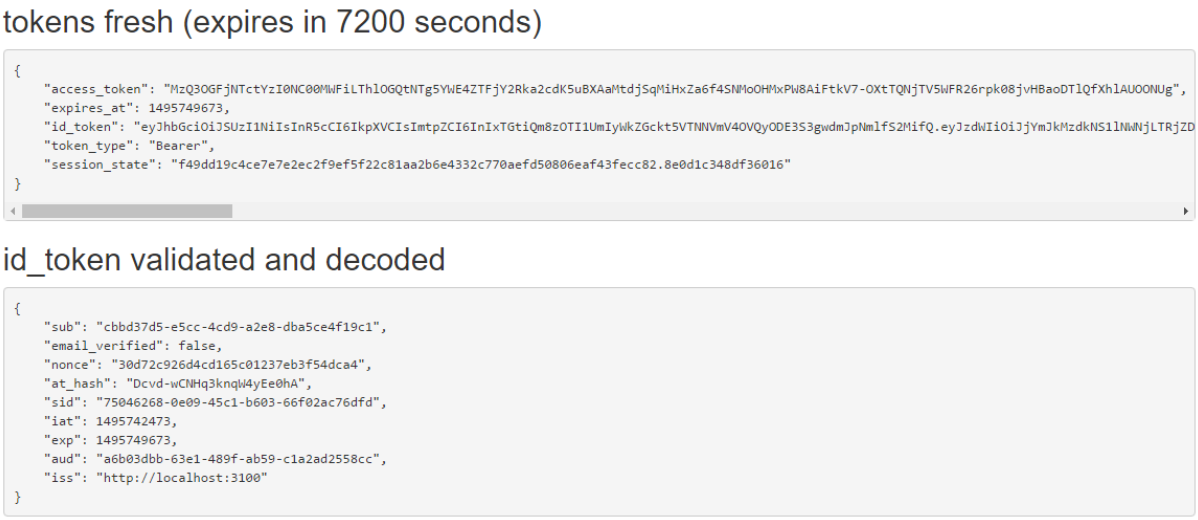

Figura 21 – *Tokens* de Acesso e *Token* ID (Tela apresentada no cliente) (Fonte: os autores)

userinfo response


```
{
      "sub": "f68a8fe0-162e-48af-8279-63a0793fe39b",
      "name": "Teste",
      "email": "teste@teste.com",
      "politicas": {
            "PI_SI_PP_FI": "1",
            "PCP_SI_PP_FI": "1",
            "LO_SI_PP_FI": "1",
            "AH_SI_PP_FI": "1",
            "RS_SI_PP_FI": "1",
             //Políticas Suprimidas
            "AH_CO_TP_MI": "1",
            "RS_CO_TP_MI": "1",
            "DISSEMINACAO": "1",
            "CRIPTOGRAFIA": "1"
      },
      "iat": 1496001673,
      "exp": 1496008873,
      "aud": "c6281d86-682f-42ce-a8fb-bf131b3464ef",
      "iss": "http://localhost:3100"
}
```


introspections

access_token response


```
{
    "active": true,
    "token_type": "access_token",
    "sub": "f68a8fe0-162e-48af-8279-63a0793fe39b",
    "client_id": "c6281d86-682f-42ce-a8fb-bf131b3464ef",
    "exp": 1496008873,
    "iat": 1496001673,
    "sid": "a181d7fc-4428-4744-b73e-8008364064a1",
    "iss": "http://localhost:3100",
    "jti": "MDEzZmJjMjctYmRlNy00NGI4LWFkOGItNWQ3MzUwYmRjZWRi",
    "scope": "openid"
}
```


Figura 22 – *UserInfo Response* e *Access Token Response* (Tela apresentada no cliente) (Fonte: os autores)

5 Considerações Finais

De acordo com KALLELA (2008), a possibilidade de um usuário interagir com diversos serviços utilizando a mesma credencial, utilizando o conceito de Identidade Federada, tem se difundido nos últimos anos. Isso reforça a necessidade já fundamentada de adaptar as tecnologias existentes para que estas forneçam suporte aos novos métodos de autenticação, garantindo opções modernas e seguras para o uso de redes federadas de autenticação.

Ao mesmo tempo, a privacidade é descrita como a questão mais preocupante por consumidores de lojas online nos Estados Unidos, estando à frente de itens como spam e segurança. A divulgação das práticas de privacidade por um site reduz significativamente as preocupações de seus usuários, tornando o ambiente mais confiável para eles (BENASSI, 1999).

Considerando a evidente necessidade de privacidade e controle de disseminação sobre os dados pessoais do usuário em provedores de serviço, acredita-se que a necessidade de se pensar em formas de garantir ao usuário final que este defina qual destino dar a suas informações se torna essencial, e a utilização do modelo de *Sticky Policies* é uma forma efetiva de prover as garantias de segurança e privacidade que o usuário busca.

5.1 Trabalhos Futuros

Ao longo da elaboração deste trabalho foram identificados alguns itens que podem estender os resultados obtidos:

- Implementação de métodos que permitam ao provedor de identidades *OpenID Connect* do protótipo elaborado cadastrar usuários diversos, permitindo ainda que cada um defina suas políticas de privacidade e estes dados sejam salvos no banco de dados, além da identificação de usuário e senha na tela de autorização.
- Implementação das modificações que permitem a transferência de *Sticky Policies* em outros modelos de provedor e cliente *OpenID Connect*, garantindo intercompatibilidade entre sistemas e linguagens.
- Elaboração de documentação e casos de uso para submissão ao *OpenID Foundation*, entidade mantenedora do protocolo *OpenID Connect*, sugerindo a adesão do modelo de *Sticky Policies* como funcionalidade oficial e suportada pelo protocolo.

- Estudo de modelo conceitual para implantação em clientes *OpenID Connect* que certifiquem o usuário final de que suas definições quanto à propagação e disseminação de seus dados serão cumpridas pelo cliente que as recebe, impossibilitando fraudes de forma a não receber as políticas.

6 Referências

AMAZON AWS. **O que é NoSQL?**. 2017 Disponível em: <https://aws.amazon.com/pt/nosql>. Acesso em 27 de maio de 2017.

BASSO, T.; MONTECCHI, L.; MORAES, R.; JINO, M.; BONDAVALLI, A. **Towards a UMO Profile for Privacy-Aware Applications**. 2015 IEEE International Conference on Computer and Information Technology, Los Alamitos, CA, p. 371-378, 2015.

BENASSI, P. **TRUSTe: An Online Privacy Seal Program**. ACM, New York, NY, p 56-59, 1999.

BERTINO, E.; TAKAHASHI, K. **Identity Management: Concepts, Technologies, and Systems**. Artech House, Boston, MA, 2011.

CARAMUJO, J.; DA SILVA, A. R. **Analyzing Privacy Policies based on a Privacy-Aware Profile: The Facebook and LinkedIn case studies**. 2015 IEEE 17th Conference on Business Informatics, p. 77-84, 2015.

CONNECT2ID. **OpenID Connect explained**. 2014. Disponível em: <http://connect2id.com/learn/openid-connect> Acesso em 27 de outubro de 2016

DAYLEY, B. **Node.js, MongoDB, and AngularJS Web Development**. Addyson-Wesley Ed. Michigan, EUA. 2014

FLANAGAN, D. **JavaScript. The definitive Guide**. 5th Edition. O'Reilly Media Inc. 2006

JETBRAINS **InteliJ IDEA**. Disponível em https://www.jetbrains.com/idea/download/ Acesso em 31 de maio de 2017

INTERNATIONAL STANDARD ISO/IEC 24760-1 **Information technology - Security techniques - A framework for identity management — Part 1: Terminology and concepts**. First edition, 2011

JØSANG, A.; FABRE, J.; HAY, B.; DALZIEL, J.; POPE, S. . **Trust requirements in identity management**. In CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research, pages 99–108, Darlinghurst, Australia. 2005.

JØSANG, A.; POPE, S. **User centric identity management**. CITESEER. AusCERT Asia Pacific Information Technology Security Conference. [S.l.] 2005.

KALLELA, J. **Federated Identity Management Solutions. Seminar on Internetworking**, TKKT-110.5190, 2008.

MONGODB **MongoDB Download Center**. Disponível em `https://www.mongodb.com/download-center`. Acesso em 31 de maio de 2017

MOREIRA, Edré Q.; FOSCARINI, Éverton D.; JUNIOR, Gessy C. da Silva; ALIXANDRINA, Lídia A. O.; NETO, Lourival P. V., ROSSETTO, Silvana. **Federação CAFe: Implantação do Provedor de Identidade**. Rio de Janeiro: Escola Superior de Redes, RNP, 2011.

OAUTH. **OAuth 2.0**. Disponível em: `<http://oauth.net/2/>`. Acesso em 05 de junho de 2016.

OPENID CONNECT. **OpenID Connect**. 2016 Disponível em: `<http://openid.net/connect/>`. Acesso em 29 de maio de 2016.

OPENID CERTIFICATION. **OpenID Connect**. 2017 Disponível em: `<http://openid.net/certification/>`. Acesso em 23 de maio de 2017.

PEARSON, S.; MONT, M. C. **Sticky Policies: An Approach for Managing Privacy across Multiple Parties**. Vol. 44, p. 60-68, 2011. Disponível em: `<https://documents.epfl.ch/users/a/ay/ayday/www/mini_project/StickyPolicies.pdf>` Acesso em 12 de Maio de 2017.

PRIVACY PATTERNS. **Sticky Policies**. Disponível em: `<https://privacypatterns.org/patterns/sticky-policy>` Acesso em 11 de Maio de 2017.

RFC 2828. **Internet Security Glossary**. 2000. Disponível em: `<https://www.ietf.org/rfc/rfc2828.txt>`. Acesso em 20 de outubro de 2016.

RFC 6749. **The OAuth 2.0 Authorization Framework**. 2012 Disponível em: <https://www.ietf.org/rfc/rfc2828.txt>. Acesso em 20 de abril de 2017.

RFC 7523. **JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants**. 2015. Disponível em: <https://tools.ietf.org/html/rfc7523> Acesso em 27 de outubro de 2016.

ROUSE, M., **Definition - Personally Identifiable Information (PII)**. 2014. Disponıvel em: <http://searchfinancialsecurity.techtarget.com/definition/personally-identifiable-information> Acesso em 12 de Maio de 2017

SALDANHA, A. **SAML vs. OAuth: Which one should I use?** Disponível em: <https://dzone.com/articles/saml-versus-oauth-which-one> Acesso em 12 de junho de 2016.

SCHWARTZ, M. **The future of cloud identity security and SSO: OpenID Connect. GLUU**. Disponível em: <https://www.gluu.org/resources/documents/articles/the-future-of-cloud-identity-security-and-sso-openid-connect/> Acesso em 12 de junho de 2016.

SKOKAN, F. **Identity, OpenID Connect, OAuth 2.0, SSO, Authorization, Node.js, Ruby**. Repositório GitHub. 2017 Disponıvel em: <https://github.com/panva/> Acesso em 12 de Maio 2017

SOUZA, M. C; WANGHAM, M. S.; DE MELLO, E. R. **Implantação de uma Comunidade Acadêmica Federada para Experimentação usando Framework Shibboleth**. Computer on the Beach, Itajaí, SC, p. 451-453, 2014.

STALLINGS, W. **Network Security Essentials** 4ª Edição. Pearson Education, Inc. 2011

SUBLIME TEXT **Sublime Text**. Disponível em https://www.sublimetext.com/3. Acesso em 31 de maio de 2017

SWITCH Identity Blog - **OpenID Connect meets SAML and Shibboleth**. Disponível em: <http://bit.ly/2gffYXb>. Acesso em 27 de maio de 2016.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 3ª Edição. Editora Pearson. 2009.

TANG, Q. **On Using Encryption Techniques to Enhance Sticky Policies Enforcement**. 2008 Disponıvel em: <https://core.ac.uk/download/pdf/11468820.pdf> Acesso em 12 de Maio de 2017

VILLARREAL, M. E.; VILLARREAL, S. R.; WESTPHAL, C. M.; WERNER, J. **Privacy Token: A Mechanism for User's Privacy Specification in Identity Management Systems for the Cloud**. ICN 2017 - The Sixteenth International Conference on Networks, 2017 Disponível em: https://www.thinkmind.org/download.php?articleid=icn_2017_3_30_30037 Acesso em 05 de Julho de 2017.

WANGHAM, M. S.; DOMENECH, M. C.; DE MELLO, E. R. **Infraestruturas de Autenticação e de Autorização para Internet das Coisas**. Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. Manaus, AM, p. 156-205, 2013.

WEBFINANCE, Inc., **What is privacy policy?** 2011. Disponível em <http://www.businessdictionary.com/definition/privacy-policy.html> Acesso em 17 de maio de 2017

Apêndices

A Classes modificadas no node-openid-client

A.1 index.js

```
1  'use strict';
2
3  const { Issuer } = require('..');
4
5  const {
6  ISSUER = 'http://localhost:3100/',
7  PORT = 3000,
8  } = process.env;
9
10 const appFactory = require('./app');
11
12 Issuer.discover(ISSUER).then((issuer) => {
13 const app = appFactory(issuer);
14 app.listen(PORT);
15 }).catch((err) => {
16 console.error(err); // eslint-disable-line no-console
17 process.exit(1);
18 });
```

Código Fonte A.1 – example/index.js

A.2 APP.JS

```
'use strict';

/* eslint-disable import/no-extraneous-dependencies */

const _ = require('lodash');
const Koa = require('koa');
const crypto = require('crypto');
const url = require('url');
const uuid = require('uuid');
const jose = require('node-jose');
const path = require('path');
const Router = require('koa-router');
const body = require('koa-body');
const session = require('koa-session');
const render = require('koa-ejs');

const PRESETS = require('./presets');

module.exports = (issuer) => {
const app = new Koa();

if (process.env.NODE_ENV === 'production') {
app.proxy = true;

app.use(async (ctx, next) => {
if (ctx.secure) {
await next();
} else {
ctx.redirect(ctx.href.replace(/^http:\/\//i, 'https://'));
}
});
}

app.keys = ['some secret hurr'];
app.use(session(app));

const CLIENTS = new Map();
const TOKENS = new Map();

render(app, {
cache: false,
layout: '_layout',
root: path.join(__dirname, 'views'),
});

app.use(async (ctx, next) => {
```

```
47 ctx.session.id = ctx.session.id || uuid();
48 await next();
49 });
50
51 app.use(async (ctx, next) => {
52 try {
53 await next();
54 } catch (error) {
55 await ctx.render('error', { issuer, error, session: ctx.session });
56 }
57 });
58
59 app.use(async (ctx, next) => {
60 if (!CLIENTS.has(ctx.session.id) && !ctx.path.startsWith('/setup')) {
61 ctx.redirect('/setup');
62 }
63 await next();
64 });
65
66 const router = new Router();
67
68 router.get('/', async (ctx, next) => {
69 await ctx.render('index', { session: ctx.session, issuer });
70 return next();
71 });
72
73 router.get('/rpframe', async (ctx, next) => {
74 const clientId = CLIENTS.get(ctx.session.id).client_id;
75 const sessionState = TOKENS.get(ctx.session.id).session_state;
76 await ctx.render('rp_frame', { session: ctx.session, layout: false,
      issuer, clientId, sessionState });
77 return next();
78 });
79
80 router.get('/setup', async (ctx, next) => {
81 await ctx.render('setup', { session: ctx.session, presets: PRESETS,
      issuer });
82 return next();
83 });
84
85 router.post('/setup/:preset', async (ctx, next) => {
86 let keystore;
87 const preset = PRESETS[ctx.params.preset];
88 ctx.session.loggedIn = false;
89
90 if (preset.keystore) {
91 keystore = jose.JWK.createKeyStore();
```

```
92 await keystore.generate.apply(keystore, preset.keystore);
93 }
94
95 const metadata = Object.assign({
96 post_logout_redirect_uris: [url.resolve(ctx.href, '/')],
97 redirect_uris: [url.resolve(ctx.href, '/cb')],
98 }, preset.registration);
99
100 const client = await issuer.Client.register(metadata, keystore);
101 client.CLOCK_TOLERANCE = 5;
102 CLIENTS.set(ctx.session.id, client);
103 ctx.session.authorization_params = preset.authorization_params;
104
105 ctx.redirect('/client');
106 return next();
107 });
108
109 router.get('/issuer', async (ctx, next) => {
110 await ctx.render('issuer', {
111 issuer,
112 keystore: (await issuer.keystore()),
113 session: ctx.session,
114 });
115 return next();
116 });
117
118 router.get('/client', async (ctx, next) => {
119 await ctx.render('client', { client: CLIENTS.get(ctx.session.id),
    session: ctx.session, issuer });
120 return next();
121 });
122
123 router.get('/logout', async (ctx, next) => {
124 const id = ctx.session.id;
125 ctx.session.loggedIn = false;
126
127 if (!TOKENS.has(id)) {
128 return ctx.redirect('/');
129 }
130
131 const tokens = TOKENS.get(id);
132 TOKENS.delete(id);
133
134 const client = CLIENTS.get(id);
135
136 try {
137 await Promise.all([
```

```
138 tokens.access_token ? client.revoke(tokens.access_token, 'access_token')
        : undefined,
139 tokens.refresh_token ? client.revoke(tokens.refresh_token, '
    refresh_token') : undefined,
140 ]);
141 } catch (err) {}
142
143 ctx.redirect(url.format(Object.assign(url.parse(issuer.
    end_session_endpoint), {
144 search: null,
145 query: {
146 id_token_hint: tokens.id_token,
147 post_logout_redirect_uri: url.resolve(ctx.href, '/'),
148 },
149 })));
150
151 return next();
152 });
153
154 router.get('/login', async (ctx, next) => {
155 ctx.session.state = crypto.randomBytes(16).toString('hex');
156 ctx.session.nonce = crypto.randomBytes(16).toString('hex');
157
158 const authorizationRequest = Object.assign({
159 claims: {
160 id_token: { email_verified: null },
161 userinfo: { name:null, sub: null, email: null, politicas: null},
162 },
163 redirect_uri: url.resolve(ctx.href, 'cb'),
164 scope: 'openid',
165 state: ctx.session.state,
166 nonce: ctx.session.nonce,
167 }, ctx.session.authorization_params);
168
169 const authz = CLIENTS.get(ctx.session.id).authorizationUrl(
    authorizationRequest);
170
171 ctx.redirect(authz);
172 return next();
173 });
174
175 router.get('/refresh', async (ctx, next) => {
176 if (!TOKENS.has(ctx.session.id)) {
177 ctx.session = null;
178 ctx.redirect('/');
179 } else {
180 const tokens = TOKENS.get(ctx.session.id);
```

```
181 const client = CLIENTS.get(ctx.session.id);
182
183 const refreshed = await client.refresh(tokens);
184 refreshed.session_state = tokens.session_state;
185
186 TOKENS.set(ctx.session.id, refreshed);
187
188 ctx.redirect('/user');
189 }
190
191 return next();
192 });
193
194 router.get('/cb', async (ctx, next) => {
195 const state = ctx.session.state;
196 delete ctx.session.state;
197 const nonce = ctx.session.nonce;
198 delete ctx.session.nonce;
199 const client = CLIENTS.get(ctx.session.id);
200 const params = client.callbackParams(ctx.request.req);
201
202 TOKENS.set(ctx.session.id,
203 await client.authorizationCallback(url.resolve(ctx.href, 'cb'), params,
       { nonce, state }));
204
205 ctx.session.loggedIn = true;
206
207 ctx.redirect('/user');
208
209 return next();
210 });
211
212 router.post('/cb', body({ patchNode: true }), async (ctx, next) => {
213 const state = ctx.session.state;
214 delete ctx.session.state;
215 const nonce = ctx.session.nonce;
216 delete ctx.session.nonce;
217 const client = CLIENTS.get(ctx.session.id);
218 const params = client.callbackParams(ctx.request.req);
219
220 TOKENS.set(ctx.session.id,
221 await client.authorizationCallback(url.resolve(ctx.href, 'cb'), params,
       { nonce, state }));
222
223 ctx.session.loggedIn = true;
224
225 ctx.redirect('/user');
```

```
226
227 return next();
228 });
229
230 function rejectionHandler(error) {
231 if (error.name === 'OpenIdConnectError') {
232 return error;
233 }
234
235 throw error;
236 }
237
238 router.get('/user', async (ctx, next) => {
239 if (!TOKENS.has(ctx.session.id)) {
240 ctx.session.loggedIn = false;
241 return ctx.redirect('/client');
242 }
243 const tokens = TOKENS.get(ctx.session.id);
244 const client = CLIENTS.get(ctx.session.id);
245
246 const context = {
247 tokens,
248 userinfo: undefined,
249 id_token: tokens.id_token ? tokens.claims : undefined,
250 session: ctx.session,
251 introspections: {},
252 issuer,
253 };
254
255 const promises = [];
256
257 _.forEach(tokens, (value, key) => {
258 if (key.endsWith('token') && key !== 'id_token') {
259 const p = client.introspect(value, key)
260 .then((result) => {
261 context.introspections[key] = result;
262 })
263 .catch(rejectionHandler);
264 promises.push(p);
265 }
266 return undefined;
267 });
268
269 if (tokens.access_token) {
270 const p = client.userinfo(tokens)
271 .then(userinfo => client.fetchDistributedClaims(userinfo))
272 .then(userinfo => client.unpackAggregatedClaims(userinfo))
```

```
273 .then((result) => {
274 context.userinfo = result;
275 })
276 .catch(rejectionHandler);
277 promises.push(p);
278 }
279
280 await Promise.all(promises);
281 await ctx.render('user', context);
282
283 return next();
284 });
285
286 app.use(router.routes());
287 app.use(router.allowedMethods());
288
289 return app;
290 };
```

Código Fonte A.2 – example/app.js

A.3 CLIENT.JS

```
'use strict';

const util = require('util');
const assert = require('assert');
const http = require('http');
const crypto = require('crypto');
const querystring = require('querystring');
const jose = require('node-jose');
const uuid = require('uuid');
const base64url = require('base64url');
const url = require('url');
const _ = require('lodash');
const got = require('got');
const tokenHash = require('oidc-token-hash');

const errorHandler = require('./error_handler');
const expectResponse = require('./expect_response');
const TokenSet = require('./token_set');
const OpenIdConnectError = require('./open_id_connect_error');
const now = require('./unix_timestamp');

const CALLBACK_PROPERTIES = require('./consts').CALLBACK_PROPERTIES;
const CLIENT_DEFAULTS = require('./consts').CLIENT_DEFAULTS;
const JWT_CONTENT = require('./consts').JWT_CONTENT;

const issuerRegistry = require('./issuer_registry');

const map = new WeakMap();
const format = 'compact';

function bearer(token) {
return `Bearer ${token}`;
}

function instance(ctx) {
if (!map.has(ctx)) map.set(ctx, { metadata: {} });
return map.get(ctx);
}

function cleanUpClaims(claims) {
if (_.isEmpty(claims._claim_names)) delete claims._claim_names;
if (_.isEmpty(claims._claim_sources)) delete claims._claim_sources;
return claims;
}

function assignClaim(target, source, sourceName) {
```

```
47 return (inSource, claim) => {
48 if (inSource === sourceName) {
49 assert(source[claim] !== undefined, `expected claim "${claim}" in "${
      sourceName}"`);
50 target[claim] = source[claim];
51 delete target._claim_names[claim];
52 }
53 };
54 }
55
56 function getFromJWT(jwt, position, claim) {
57 assert.equal(typeof jwt, 'string', 'invalid JWT type, expected a string'
      );
58 const parts = jwt.split('.');
59 assert.equal(parts.length, 3, 'invalid JWT format, expected three parts'
      );
60 const parsed = JSON.parse(base64url.decode(parts[position]));
61 return typeof claim === 'undefined' ? parsed : parsed[claim];
62 }
63
64 function getSub(jwt) {
65 return getFromJWT(jwt, 1, 'sub');
66 }
67
68 function getIss(jwt) {
69 return getFromJWT(jwt, 1, 'iss');
70 }
71
72 function getHeader(jwt) {
73 return getFromJWT(jwt, 0);
74 }
75
76 function getPayload(jwt) {
77 return getFromJWT(jwt, 1);
78 }
79
80 function assignErrSrc(sourceName) {
81 return (err) => {
82 err.src = sourceName;
83 throw err;
84 };
85 }
86
87 function authorizationParams(params) {
88 assert.equal(typeof params, 'object', 'you must provide an object');
89
90 const authParams = _.chain(params).defaults({
```

```
client_id: this.client_id,
scope: 'openid',
response_type: 'code',
}).forEach((value, key, object) => {
if (value === null || value === undefined) {
delete object[key];
} else if (key === 'claims' && typeof value === 'object') {
object[key] = JSON.stringify(value);
} else if (typeof value !== 'string') {
object[key] = String(value);
}
}).value();

assert(authParams.response_type === 'code' || authParams.nonce,
'nonce MUST be provided for implicit and hybrid flows');

return authParams;
}

function claimJWT(jwt) {
try {
const iss = getIss(jwt);
const keyDef = getHeader(jwt);
assert(keyDef.alg, 'claim source is missing JWT header alg property');

if (keyDef.alg === 'none') return Promise.resolve(getPayload(jwt));

const getKey = (() => {
if (!iss || iss === this.issuer.issuer) {
return this.issuer.key(keyDef);
} else if (issuerRegistry.has(iss)) {
return issuerRegistry.get(iss).key(keyDef);
}
return this.issuer.constructor.discover(iss).then(issuer => issuer.key(
   keyDef));
})();

return getKey
.then(key => jose.JWS.createVerify(key).verify(jwt))
.then(result => JSON.parse(result.payload));
} catch (error) {
return Promise.reject(error);
}
}

const deprecatedKeystore = util.deprecate(keystore => keystore,
'passing keystore directly is deprecated, pass an object with keystore
```

```
    property instead');
137
138 class Client {
139 /**
140 * @name constructor
141 * @api public
142 */
143 constructor(metadata, keystore) {
144 const properties = Object.assign({}, CLIENT_DEFAULTS, metadata);
145
146 if (String(properties.token_endpoint_auth_method).endsWith('_jwt')) {
147 assert(this.issuer.token_endpoint_auth_signing_alg_values_supported,
148 'token_endpoint_auth_signing_alg_values_supported must be provided on
    the issuer');
149 }
150
151 ['introspection', 'revocation'].forEach((endpoint) => {
152 _.defaults(properties, {
153 [`${endpoint}_endpoint_auth_method`]: properties.
    token_endpoint_auth_method,
154 [`${endpoint}_endpoint_auth_signing_alg`]: properties.
    token_endpoint_auth_signing_alg,
155 });
156 if (String(properties[`${endpoint}_endpoint_auth_method`]).endsWith('
    _jwt')) {
157 assert(this.issuer[`${endpoint}
    _endpoint_auth_signing_alg_values_supported`],
158 `${endpoint}_endpoint_auth_signing_alg_values_supported must be provided
     on the issuer`);
159 }
160 });
161
162
163 _.forEach(properties, (value, key) => {
164 instance(this).metadata[key] = value;
165 if (!this[key]) {
166 Object.defineProperty(this, key, {
167 get() { return instance(this).metadata[key]; },
168 });
169 }
170 });
171
172 if (keystore !== undefined) {
173 assert(jose.JWK.isKeyStore(keystore), 'keystore must be an instance of
    jose.JWK.KeyStore');
174 instance(this).keystore = keystore;
175 }
```

```
176
177 this.CLOCK_TOLERANCE = 0;
178 }
179
180 /**
181 * @name authorizationUrl
182 * @api public
183 */
184 authorizationUrl(params) {
185 assert(this.issuer.authorization_endpoint, 'authorization_endpoint must
       be configured');
186 return url.format(_.defaults({
187 search: null,
188 query: authorizationParams.call(this, params),
189 }, url.parse(this.issuer.authorization_endpoint)));
190 }
191
192 /**
193 * @name authorizationPost
194 * @api public
195 */
196 authorizationPost(params) {
197 const inputs = authorizationParams.call(this, params);
198 const formInputs = Object.keys(inputs)
199 .map(name => `<input type="hidden" name="${name}" value="${inputs[name]}
       "/>`).join('\n');
200
201 return `<!DOCTYPE html>
202 <head>
203 <title>Requesting Authorization</title>
204 </head>
205 <body onload="javascript:document.forms[0].submit()">
206 <form method="post" action="${this.issuer.authorization_endpoint}">
207 ${formInputs}
208 </form>
209 </body>
210 </html>`;
211 }
212
213 /**
214 * @name callbackParams
215 * @api public
216 */
217 callbackParams(input) { // eslint-disable-line
218 const isIncomingMessage = input instanceof http.IncomingMessage;
219 const isString = typeof input === 'string';
220
```

```
221 assert(isString || isIncomingMessage, '#callbackParams only accepts
       string urls or http.IncomingMessage');
222
223 let uri;
224 if (isIncomingMessage) {
225 const msg = input;
226
227 switch (msg.method) {
228 case 'GET':
229 uri = msg.url;
230 break;
231 case 'POST':
232 assert(msg.body, 'incoming message body missing, include a body parser
       prior to this call');
233 switch (typeof msg.body) {
234 case 'object':
235 case 'string':
236 if (Buffer.isBuffer(msg.body)) {
237 return querystring.parse(msg.body.toString('utf-8'));
238 } else if (typeof msg.body === 'string') {
239 return querystring.parse(msg.body);
240 }
241
242 return msg.body;
243 default:
244 throw new Error('invalid IncomingMessage body object');
245 }
246 default:
247 throw new Error('invalid IncomingMessage method');
248 }
249 } else {
250 uri = input;
251 }
252
253 return _.pick(url.parse(uri, true).query, CALLBACK_PROPERTIES);
254 }
255
256 /**
257 * @name authorizationCallback
258 * @api public
259 */
260 authorizationCallback(redirectUri, parameters, checks) {
261 const params = _.pick(parameters, CALLBACK_PROPERTIES);
262 const toCheck = checks || {};
263
264 if (this.default_max_age && !toCheck.max_age) toCheck.max_age = this.
       default_max_age;
```

```
265
266 if (toCheck.state !== parameters.state) {
267 return Promise.reject(new Error('state mismatch'));
268 }
269
270 if (params.error) {
271 return Promise.reject(new OpenIdConnectError(params));
272 }
273
274 let promise;
275
276 if (params.id_token) {
277 promise = Promise.resolve(new TokenSet(params))
278 .then(tokenset => this.decryptIdToken(tokenset, 'id_token'))
279 .then(tokenset => this.validateIdToken(tokenset, toCheck.nonce, '
    authorization', toCheck.max_age));
280 }
281
282 if (params.code) {
283 const grantCall = () => this.grant({
284 grant_type: 'authorization_code',
285 code: params.code,
286 redirect_uri: redirectUri,
287 code_verifier: toCheck.code_verifier,
288 })
289 .then(tokenset => this.decryptIdToken(tokenset, 'id_token'))
290 .then(tokenset => this.validateIdToken(tokenset, toCheck.nonce, 'token',
     toCheck.max_age))
291 .then((tokenset) => {
292 if (params.session_state) tokenset.session_state = params.session_state;
293 return tokenset;
294 });
295
296 if (promise) {
297 promise = promise.then(grantCall);
298 } else {
299 return grantCall();
300 }
301 }
302
303 return promise;
304 }
305
306 /**
307 * @name oauthCallback
308 * @api public
309 */
```

```
oauthCallback(redirectUri, parameters, checks) {
const params = _.pick(parameters, CALLBACK_PROPERTIES);
const toCheck = checks || {};

if (toCheck.state !== parameters.state) {
return Promise.reject(new Error('state mismatch'));
}

if (params.error) {
return Promise.reject(new OpenIdConnectError(params));
}

if (params.code) {
return this.grant({
grant_type: 'authorization_code',
code: params.code,
redirect_uri: redirectUri,
code_verifier: toCheck.code_verifier,
});
}

return Promise.resolve(new TokenSet(params));
}

/**
* @name decryptIdToken
* @api private
*/
decryptIdToken(token, use) {
if (
(use === 'userinfo' && !this.userinfo_encrypted_response_alg) ||
(use === 'id_token' && !this.id_token_encrypted_response_alg)
) {
return Promise.resolve(token);
}

let idToken = token;

if (idToken instanceof TokenSet) {
assert(idToken.id_token, 'id_token not present in TokenSet');
idToken = idToken.id_token;
}

let expectedAlg;
let expectedEnc;

if (use === 'userinfo') {
```

```
357 expectedAlg = this.userinfo_encrypted_response_alg;
358 expectedEnc = this.userinfo_encrypted_response_enc;
359 } else {
360 expectedAlg = this.id_token_encrypted_response_alg;
361 expectedEnc = this.id_token_encrypted_response_enc;
362 }
363
364 const header = JSON.parse(base64url.decode(idToken.split('.')[0]));
365
366 assert.equal(header.alg, expectedAlg, 'unexpected alg received');
367 assert.equal(header.enc, expectedEnc, 'unexpected enc received');
368
369 const keystoreOrSecret = expectedAlg.match(/^(RSA|ECDH)/) ?
370 Promise.resolve(instance(this).keystore) : this.joseSecret(expectedAlg);
371
372 return keystoreOrSecret.then(keyOrStore => jose.JWE.createDecrypt(
      keyOrStore).decrypt(idToken)
373 .then((result) => {
374 if (token instanceof TokenSet) {
375 token.id_token = result.payload.toString('utf8');
376 return token;
377 }
378 return result.payload.toString('utf8');
379 }));
380 }
381
382 /**
383 * @name validateIdToken
384 * @api private
385 */
386 validateIdToken(tokenSet, nonce, returnedBy, maxAge) {
387 let idToken = tokenSet;
388
389 const expectedAlg = (() => {
390 if (returnedBy === 'userinfo') return this.userinfo_signed_response_alg;
391 return this.id_token_signed_response_alg;
392 })();
393
394 const isTokenSet = idToken instanceof TokenSet;
395
396 if (isTokenSet) {
397 assert(idToken.id_token, 'id_token not present in TokenSet');
398 idToken = idToken.id_token;
399 }
400
401 idToken = String(idToken);
402
```

```
403 const timestamp = now();
404 const parts = idToken.split('.');
405 const header = JSON.parse(base64url.decode(parts[0]));
406 const payload = JSON.parse(base64url.decode(parts[1]));
407
408 const verifyPresence = (prop) => {
409 if (payload[prop] === undefined) {
410 throw new Error(`missing required JWT property ${prop}`);
411 }
412 };
413
414 assert.equal(header.alg, expectedAlg, 'unexpected algorithm received');
415
416 if (returnedBy !== 'userinfo') {
417 ['iss', 'sub', 'aud', 'exp', 'iat'].forEach(verifyPresence);
418 }
419
420 if (payload.iss !== undefined) {
421 assert.equal(this.issuer.issuer, payload.iss, 'unexpected iss value');
422 }
423
424 if (payload.iat !== undefined) {
425 assert.equal(typeof payload.iat, 'number', 'iat is not a number');
426 assert(payload.iat <= timestamp + this.CLOCK_TOLERANCE, 'id_token issued
        in the future');
427 }
428
429 if (payload.nbf !== undefined) {
430 assert.equal(typeof payload.nbf, 'number', 'nbf is not a number');
431 assert(payload.nbf <= timestamp + this.CLOCK_TOLERANCE, 'id_token not
       active yet');
432 }
433
434 if (maxAge || (maxAge !== null && this.require_auth_time)) {
435 assert(payload.auth_time, 'missing required JWT property auth_time');
436 assert.equal(typeof payload.auth_time, 'number', 'auth_time is not a
       number');
437 }
438
439 if (maxAge) {
440 assert(payload.auth_time + maxAge >= timestamp - this.CLOCK_TOLERANCE, '
       too much time has elapsed since the last End-User authentication');
441 }
442
443 if (nonce !== null && (payload.nonce || nonce !== undefined)) {
444 assert.equal(payload.nonce, nonce, 'nonce mismatch');
445 }
```

```
446
447 if (payload.exp !== undefined) {
448 assert.equal(typeof payload.exp, 'number', 'exp is not a number');
449 assert(timestamp - this.CLOCK_TOLERANCE < payload.exp, 'id_token expired
    ');
450 }
451
452 if (payload.aud !== undefined) {
453 if (!Array.isArray(payload.aud)) {
454 payload.aud = [payload.aud];
455 } else if (payload.aud.length > 1 && !payload.azp) {
456 throw new Error('missing required JWT property azp');
457 }
458 }
459
460 if (payload.azp !== undefined) {
461 assert.equal(this.client_id, payload.azp, 'azp must be the client_id');
462 }
463
464 if (payload.aud !== undefined) {
465 assert(payload.aud.indexOf(this.client_id) !== -1, 'aud is missing the
    client_id');
466 }
467
468
469 if (returnedBy === 'authorization') {
470 assert(payload.at_hash || !tokenSet.access_token, 'missing required
    property at_hash');
471 assert(payload.c_hash || !tokenSet.code, 'missing required property
    c_hash');
472 }
473
474 if (tokenSet.access_token && payload.at_hash !== undefined) {
475 assert(tokenHash(payload.at_hash, tokenSet.access_token), 'at_hash
    mismatch');
476 }
477
478 if (tokenSet.code && payload.c_hash !== undefined) {
479 assert(tokenHash(payload.c_hash, tokenSet.code), 'c_hash mismatch');
480 }
481
482 if (header.alg === 'none') {
483 return Promise.resolve(tokenSet);
484 }
485
486 return (header.alg.startsWith('HS') ? this.joseSecret() : this.issuer.
    key(header))
```

```
.then(key => jose.JWS.createVerify(key).verify(idToken).catch(() => {
throw new Error('invalid signature');
}))
.then(() => tokenSet);
}

/**
* @name refresh
* @api public
*/
refresh(refreshToken) {
let token = refreshToken;

if (token instanceof TokenSet) {
if (!token.refresh_token) {
return Promise.reject(new Error('refresh_token not present in TokenSet')
    );
}
token = token.refresh_token;
}

return this.grant({
grant_type: 'refresh_token',
refresh_token: String(token),
})
.then((tokenset) => {
if (!tokenset.id_token) {
return tokenset;
}
return this.decryptIdToken(tokenset, 'id_token')
.then(() => this.validateIdToken(tokenset, null, 'token', null));
});
}

/**
* @name userinfo
* @api public
*/
userinfo(accessToken, options) {
let token = accessToken;
const opts = _.merge({
verb: 'get',
via: 'header',
}, options);

if (token instanceof TokenSet) {
if (!token.access_token) {
```

```
533 return Promise.reject(new Error('access_token not present in TokenSet'))
    ;
534 }
535 token = token.access_token;
536 }
537
538 const verb = String(opts.verb).toLowerCase();
539 let httpOptions;
540
541 switch (opts.via) {
542 case 'query':
543 assert.equal(verb, 'get', 'providers should only parse query strings for
    GET requests');
544 httpOptions = { query: { access_token: token } };
545 break;
546 case 'body':
547 assert.equal(verb, 'post', 'can only send body on POST');
548 httpOptions = { body: { access_token: token } };
549 break;
550 default:
551 httpOptions = { headers: { Authorization: bearer(token) } };
552 }
553
554 if (opts.params) {
555 if (verb === 'post') {
556 _.defaultsDeep(httpOptions, { body: opts.params });
557 } else {
558 _.defaultsDeep(httpOptions, { query: opts.params });
559 }
560 }
561
562 return got[verb](this.issuer.userinfo_endpoint, this.issuer.httpOptions(
    httpOptions))
563 .then(expectResponse(200))
564 .then((response) => {
565 if (JWT_CONTENT.exec(response.headers['content-type'])) {
566 return Promise.resolve(response.body)
567 .then(jwt => this.decryptIdToken(jwt, 'userinfo'))
568 .then((jwt) => {
569 if (!this.userinfo_signed_response_alg) return JSON.parse(jwt);
570 return this.validateIdToken(jwt, null, 'userinfo', null)
571 .then(valid => JSON.parse(base64url.decode(valid.split('.')[1])));
572 });
573 }
574
575 return JSON.parse(response.body);
576 })
```

```
577 .then((parsed) => {
578 if (accessToken.id_token) {
579 assert.equal(getSub(accessToken.id_token), parsed.sub, 'userinfo sub
       mismatch');
580 }
581
582 return parsed;
583 })
584 .catch(errorHandler);
585 }
586
587 /**
588 * @name derivedKey
589 * @api private
590 */
591 derivedKey(len) {
592 const cacheKey = `${len}_key`;
593 if (instance(this)[cacheKey]) {
594 return Promise.resolve(instance(this)[cacheKey]);
595 }
596
597 const derivedBuffer = crypto.createHash('sha256')
598 .update(this.client_secret)
599 .digest()
600 .slice(0, len / 8);
601
602 return jose.JWK.asKey({ k: base64url(derivedBuffer), kty: 'oct' }).then
       ((key) => {
603 instance(this)[cacheKey] = key;
604 return key;
605 });
606 }
607
608 /**
609 * @name joseSecret
610 * @api private
611 */
612 joseSecret(alg) {
613 if (String(alg).match(/^A(128|192|256)(GCM)?KW$/)) {
614 return this.derivedKey(RegExp.$1);
615 }
616
617 if (instance(this).jose_secret) {
618 return Promise.resolve(instance(this).jose_secret);
619 }
620
621 return jose.JWK.asKey({ k: base64url(this.client_secret), kty: 'oct' }).
```

```
    then((key) => {
instance(this).jose_secret = key;
return key;
});
}

/**
* @name grant
* @api public
*/
grant(body) {
assert(this.issuer.token_endpoint, 'issuer must be configured with token endpoint');
return this.authenticatedPost('token', { body: _.omitBy(body, _.isUndefined) })
.then(expectResponse(200))
.then(response => new TokenSet(JSON.parse(response.body)));
}

/**
* @name revoke
* @api public
*/
revoke(token, hint) {
assert(this.issuer.revocation_endpoint, 'issuer must be configured with revocation endpoint');
assert(!hint || typeof hint === 'string', 'hint must be a string');

const body = { token };
if (hint) body.token_type_hint = hint;
return this.authenticatedPost('revocation', { body })
.then((response) => {
if (response.body) {
return JSON.parse(response.body);
}
return {};
});
}

/**
* @name introspect
* @api public
*/
introspect(token, hint) {
assert(this.issuer.introspection_endpoint, 'issuer must be configured with introspection endpoint');
assert(!hint || typeof hint === 'string', 'hint must be a string');
```

```
664
665 const body = { token };
666 if (hint) body.token_type_hint = hint;
667 return this.authenticatedPost('introspection', { body })
668 .then(expectResponse(200))
669 .then(response => JSON.parse(response.body));
670 }
671
672 /**
673 * @name fetchDistributedClaims
674 * @api public
675 */
676 fetchDistributedClaims(claims, accessTokens) {
677 const distributedSources = _.pickBy(claims._claim_sources, def => !!def.
       endpoint);
678 const tokens = accessTokens || {};
679
680 return Promise.all(_.map(distributedSources, (def, sourceName) => {
681 const opts = {
682 headers: { Authorization: bearer(def.access_token || tokens[sourceName])
       },
683 };
684
685 return got(def.endpoint, this.issuer.httpOptions(opts))
686 .then(response => claimJWT.call(this, response.body), errorHandler)
687 .then((data) => {
688 delete claims._claim_sources[sourceName];
689 _.forEach(claims._claim_names, assignClaim(claims, data, sourceName));
690 }).catch(assignErrSrc(sourceName));
691 })).then(() => cleanUpClaims(claims));
692 }
693
694 /**
695 * @name unpackAggregatedClaims
696 * @api public
697 */
698 unpackAggregatedClaims(claims) {
699 const aggregatedSources = _.pickBy(claims._claim_sources, def => !!def.
       JWT);
700
701 return Promise.all(_.map(aggregatedSources, (def, sourceName) => {
702 const decoded = claimJWT.call(this, def.JWT);
703
704 return decoded.then((data) => {
705 delete claims._claim_sources[sourceName];
706 _.forEach(claims._claim_names, assignClaim(claims, data, sourceName));
707 }).catch(assignErrSrc(sourceName));
```

```
708 })).then(() => cleanUpClaims(claims));
709 }
710
711 /**
712 * @name authenticatedPost
713 * @api private
714 */
715 authenticatedPost(endpoint, httpOptions) {
716 return Promise.resolve(this.authFor(endpoint))
717 .then(auth => got.post(this.issuer[`${endpoint}_endpoint`], this.issuer.
       httpOptions(_.merge(httpOptions, auth)))
718 .catch(errorHandler));
719 }
720
721 /**
722 * @name createSign
723 * @api private
724 */
725 createSign(endpoint) {
726 let alg = this[`${endpoint}_endpoint_auth_signing_alg`];
727 switch (this[`${endpoint}_endpoint_auth_method`]) {
728 case 'client_secret_jwt':
729 return this.joseSecret().then((key) => {
730 if (!alg) {
731 alg = _.find(this.issuer[`${endpoint}
       _endpoint_auth_signing_alg_values_supported`],
732 signAlg => key.algorithms('sign').indexOf(signAlg) !== -1);
733 }
734
735 return jose.JWS.createSign({
736 fields: { alg, typ: 'JWT' },
737 format,
738 }, { key, reference: false });
739 });
740 case 'private_key_jwt': {
741 if (!alg) {
742 const algz = _.chain(instance(this).keystore.all())
743 .map(key => key.algorithms('sign'))
744 .flatten()
745 .uniq()
746 .value();
747
748 alg = _.find(this.issuer[`${endpoint}
       _endpoint_auth_signing_alg_values_supported`],
749 signAlg => algz.indexOf(signAlg) !== -1);
750 }
751
```

```
752 const key = instance(this).keystore.get({ alg, use: 'sig' });
753 assert(key, 'no valid key found');
754
755 return Promise.resolve(jose.JWS.createSign({
756 fields: { alg, typ: 'JWT' },
757 format,
758 }, { key, reference: true }));
759 }
760 /* istanbul ignore next */
761 default:
762 throw new Error('createSign only works for _jwt token auth methods');
763 }
764 }
765
766 /**
767 * @name authFor
768 * @api private
769 */
770 authFor(endpoint) {
771 switch (this[`${endpoint}_endpoint_auth_method`] || this.
       token_endpoint_auth_method) {
772 case 'none' :
773 return {
774 body: {
775 client_id: this.client_id,
776 },
777 };
778 case 'client_secret_post':
779 return {
780 body: {
781 client_id: this.client_id,
782 client_secret: this.client_secret,
783 },
784 };
785 case 'private_key_jwt' :
786 case 'client_secret_jwt' : {
787 const timestamp = now();
788 return this.createSign(endpoint).then(sign => sign.update(JSON.stringify
       ({
789 iat: timestamp,
790 exp: timestamp + 60,
791 jti: uuid(),
792 iss: this.client_id,
793 sub: this.client_id,
794 aud: this.issuer[`${endpoint}_endpoint`],
795 politicas: this.client_id,
796 })).final().then((client_assertion) => { // eslint-disable-line
```

```
    camelcase, arrow-body-style
return { body: {
client_assertion,
client_assertion_type: 'urn:ietf:params:oauth:client-assertion-type:jwt-
    bearer',
} };
}));
}
default: {
const value = new Buffer(`${this.client_id}:${this.client_secret}`).
    toString('base64');
return { headers: { Authorization: `Basic ${value}` } };
}
}
}


/**
* @name inspect
* @api public
*/
inspect() {
return util.format('Client <%s>', this.client_id);
}

/**
* @name register
* @api public
*/
static register(properties, opts) {
const options = (() => {
if (!opts) return {};
if (_.isPlainObject(opts)) return opts;
return { keystore: deprecatedKeystore(opts) };
})();

const keystore = options.keystore;
const initialAccessToken = options.initialAccessToken;

assert(this.issuer.registration_endpoint, 'issuer does not support
    dynamic registration');

if (keystore !== undefined && !(properties.jwks || properties.jwks_uri))
     {
assert(jose.JWK.isKeyStore(keystore), 'keystore must be an instance of
    jose.JWK.KeyStore');
assert(keystore.all().every((key) => {
```

```
838 if (key.kty === 'RSA' || key.kty === 'EC') {
839 try { key.toPEM(true); } catch (err) { return false; }
840 return true;
841 }
842 return false;
843 }), 'keystore must only contain private EC or RSA keys');
844 properties.jwks = keystore.toJSON();
845 }
846
847 const headers = { 'Content-Type': 'application/json' };
848
849 if (initialAccessToken) headers.Authorization = `Bearer ${
       initialAccessToken}`;
850
851 return got.post(this.issuer.registration_endpoint, this.issuer.
       httpOptions({
852 headers,
853 body: JSON.stringify(properties),
854 }))
855 .then(expectResponse(201))
856 .then(response => new this(JSON.parse(response.body), keystore))
857 .catch(errorHandler);
858 }
859
860 get metadata() {
861 return instance(this).metadata;
862 }
863
864 /**
865 * @name fromUri
866 * @api public
867 */
868 static fromUri(uri, token) {
869 return got(uri, this.issuer.httpOptions({
870 headers: { Authorization: bearer(token) },
871 }))
872 .then(expectResponse(200))
873 .then(response => new this(JSON.parse(response.body)), errorHandler);
874 }
875
876 /**
877 * @name requestObject
878 * @api public
879 */
880 requestObject(input, algorithms) {
881 assert.equal(typeof input, 'object', 'pass an object as the first
       argument');
```

```
const request = input || {};
const algs = algorithms || {};

_.defaults(algs, {
sign: this.request_object_signing_alg,
encrypt: {
alg: this.request_object_encryption_alg,
enc: this.request_object_encryption_enc,
},
}, {
sign: 'none',
});

const signed = (() => {
const alg = algs.sign;
const header = { alg, typ: 'JWT' };
const payload = JSON.stringify(_.defaults({}, request, {
iss: this.client_id,
aud: this.issuer.issuer,
client_id: this.client_id,
}));

if (alg === 'none') {
return Promise.resolve([
base64url(JSON.stringify(header)),
base64url(payload),
'',
].join('.'));
}

const symmetrical = alg.startsWith('HS');

const getKey = (() => {
if (symmetrical) return this.joseSecret();
const keystore = instance(this).keystore;

assert(keystore, `no keystore present for client, cannot sign using ${
   alg}`);
const key = keystore.get({ alg, use: 'sig' });
assert(key, `no key to sign with found for ${alg}`);
return Promise.resolve(key);
})();

return getKey
.then(key => jose.JWS.createSign({
fields: header,
format,
```

```
928 }, { key, reference: !symmetrical }))
929 .then(sign => sign.update(payload).final());
930 })();
931
932 if (!algs.encrypt.alg) return signed;
933 const fields = { alg: algs.encrypt.alg, enc: algs.encrypt.enc, cty: 'JWT
        ' };
934
935 /* eslint-disable arrow-body-style */
936 return this.issuer.key({
937 alg: algs.encrypt.alg,
938 enc: algs.encrypt.enc,
939 use: 'enc',
940 }, true).then((key) => {
941 return signed.then((cleartext) => {
942 return jose.JWE.createEncrypt({ format, fields }, { key })
943 .update(cleartext)
944 .final();
945 });
946 });
947 /* eslint-enable arrow-body-style */
948 }
949 }
950
951 Object.defineProperty(Client.prototype, 'grantAuth', {
952 get: util.deprecate(/* istanbul ignore next */ function grantAuth() {
953 return this.authFor('token');
954 }, 'client#grantAuth is deprecated'),
955 });
956
957 module.exports = Client;
```

Código Fonte A.3 – lib/client.js

B Classes modificadas no node-oidc-provider

B.1 index.js

```
1  'use strict';
2
3  /* eslint-disable no-console */
4
5  const Provider = require('../lib');
6  const path = require('path');
7  const _ = require('lodash');
8  const bodyParser = require('koa-body');
9  const querystring = require('querystring');
10 const render = require('koa-ejs');
11 const Router = require('koa-router');
12
13 const port = process.env.PORT || 3100;
14
15 const Account = require('./account');
16 const settings = require('./settings');
17
18 const issuer = process.env.ISSUER || 'http://localhost:3100';
19
20 if (process.env.MONGODB_URI) {
21 const MongoAdapter = require('./adapters/mongodb'); // eslint-disable-
      line global-require
22 settings.config.adapter = MongoAdapter;
23 }
24
25 settings.config.findById = Account.findById;
26 const clients = settings.clients;
27
28 const provider = new Provider(issuer, settings.config);
29
30 if (process.env.HEROKU) {
31 provider.defaultHttpOptions = { timeout: 15000 };
32 }
33
34 provider.initialize({
35 clients,
36 keystore: { keys: settings.certificates },
37 integrity: { keys: settings.integrityKeys },
38 }).then(() => {
```

```
render(provider.app, {
cache: false,
layout: '_layout',
root: path.join(__dirname, 'views'),
});

provider.app.keys = ['some secret key', 'and also the old one'];

if (process.env.NODE_ENV === 'production') {
provider.app.proxy = true;
_.set(settings.config, 'cookies.short.secure', true);
_.set(settings.config, 'cookies.long.secure', true);

provider.app.middleware.unshift(function* ensureSecure(next) {
if (this.secure) {
yield next;
} else if (this.method === 'GET' || this.method === 'HEAD') {
this.redirect(this.href.replace(/^http:\/\//i, 'https://'));
} else {
this.body = {
error: 'invalid_request',
error_description: 'do yourself a favor and only use https',
};
this.status = 400;
}
});
}

const router = new Router();

router.get('/interaction/:grant', function* renderInteraction(next) {
const cookie = provider.interactionDetails(this.req);
const client = yield provider.Client.find(cookie.params.client_id);

if (cookie.interaction.error === 'login_required') {
yield this.render('login', {
client,
cookie,
title: 'Sign-in',
debug: querystring.stringify(cookie.params, ',<br/>', ' = ', {
encodeURIComponent: value => value,
}),
interaction: querystring.stringify(cookie.interaction, ',<br/>', ' = ',
    {
encodeURIComponent: value => value,
}),
});
```

```
} else {
yield this.render('interaction', {
client,
cookie,
title: 'Authorize',
debug: querystring.stringify(cookie.params, ',<br/>', ' = ', {
encodeURIComponent: value => value,
}),
interaction: querystring.stringify(cookie.interaction, ',<br/>', ' = ',
    {
encodeURIComponent: value => value,
}),
});
}

yield next;
});

const body = bodyParser();

router.post('/interaction/:grant/confirm', body, function*
    submitConfirmationForm(next) {
const result = { consent: {} };
provider.interactionFinished(this.req, this.res, result);
yield next;
});

router.post('/interaction/:grant/login', body, function* submitLoginForm
    () {
const account = yield Account.findByLogin(this.request.body.login);

const result = {
login: {
account: account.accountId,
acr: 'urn:mace:incommon:iap:bronze',
amr: ['pwd'],
remember: !!this.request.body.remember,
ts: Math.floor(Date.now() / 1000),
},
consent: {},
};

provider.interactionFinished(this.req, this.res, result);
});

provider.app.use(router.routes());
})
```

```
129 .then(() => provider.app.listen(port))
130 .catch((err) => {
131 console.error(err);
132 process.exit(1);
133 });
```

Código Fonte B.1 – example/index.js

B.2 EXPRESS.JS

```
1  'use strict';
2
3  const Provider = require('../lib');
4  const express = require('express'); // eslint-disable-line import/no-
       unresolved
5
6  const app = express();
7
8  const provider = new Provider('http://localhost:3100/op');
9
10 provider.initialize().then(() => {
11 app.use('/op', provider.callback);
12 app.listen(3100);
13 });
```

Código Fonte B.2 – example/express.js

B.3 DEFAULT.JS

```
'use strict';

const url = require('url');
const LRU = require('lru-cache');
const epochTime = require('../helpers/epoch_time');
// const MemoryAdapter = require('../adapters/memory_adapter');
const MyAdapter = require('../../example/adapters/mongodb');

const cache = new LRU(100);

module.exports = {


/*
* acrValues
*
* description: list of the Authentication Context Class References that OP supports
* affects: discovery, ID Token acr claim values
*/
acrValues: ['0', '1', '2'],


/*
* claims
*
* description: list of the Claim Names of the Claims that the OpenID Provider MAY be able to
*   supply values for
* affects: discovery, ID Token claim names, Userinfo claim names
*/
claims: { acr: null, auth_time: null, iss: null, openid: ['sub'], politicas: null },


/*
* cookies
*
* description: options for https://github.com/pillarjs/cookies#cookiesset-name--value---options--
*   used by the OP to keep track of various User-Agent states
* affects: User-Agent sessions, passing of authorization details to interaction
*/
cookies: {
/*
```

```
42 * cookies.names
43 *
44 * description: cookie names used by the OP to store and transfer various
       states
45 * affects: User-Agent session, Session Management states and interaction
       cookie names
46 */
47 names: {
48 session: '_session', // used for session reference
49 interaction: '_grant', // interaction receives the details for it's path
       with this cookie
50 interactionResult: '_grant_result', // interaction results are expected
     in this cookie
51 resume: '_grant', // oidc-provider will store the original auth request
     parameters in this cookie for its resume path
52 state: '_state', // prefix for sessionManagement state cookies => _state
     .{clientId}
53 },
54
55 /*
56 * cookies.long
57 *
58 * description: options for long-term cookies
59 * affects: User-Agent session reference, Session Management states
60 */
61 long: { httpOnly: true, maxAge: (365.25 * 24 * 60 * 60) * 1000 }, // 1
     year in ms
62 /*
63 * cookies.short
64 *
65 * description: options for short-term cookies
66 * affects: passing of authorization details to interaction
67 */
68 short: { httpOnly: true, maxAge: (60 * 60) * 1000 }, // 60 minutes in ms
69 },
70
71
72 /*
73 * discovery
74 *
75 * description: pass additional properties to this object to extend the
     discovery document
76 * affects: discovery
77 */
78 discovery: {
79 claim_types_supported: ['normal', 'aggregated', 'distributed'],
80 claims_locales_supported: undefined,
```

```
display_values_supported: undefined,
op_policy_uri: undefined,
op_tos_uri: undefined,
service_documentation: undefined,
ui_locales_supported: undefined,
},


/*
* extraParams
*
* description: pass an iterable object (i.e. array or set) to extend the
    parameters recognized
*  by the authorization endpoint. These parameters are then available in
    ctx.oidc.params as well
*  as passed via the _grant cookie to interaction
* affects: authorization, interaction
*/
extraParams: [],


/*
* features
*
* description: enable/disable feature 'packs', see configuration.md for
   more details
*/
features: {
devInteractions: true,
backchannelLogout: false,
claimsParameter: false,
clientCredentials: false,
discovery: true,
encryption: false,
introspection: false,
alwaysIssueRefresh: false,
registration: false,
registrationManagement: false,
request: false,
requestUri: false,
revocation: false,
oauthNativeApps: false,
sessionManagement: false,
pkce: true,
},


```

```
125 /*
126 * prompts
127 *
128 * description: list of the prompt values that the OpenID Provider MAY be
        able to resolve
129 * affects: authorization
130 */
131 prompts: ['consent', 'login', 'none'],
132
133
134 /*
135 * responseTypes
136 *
137 * description: list of the OAuth 2.0 response_type values that OP
       supports
138 * affects: authorization, discovery, registration, registration
       management
139 */
140 responseTypes: [
141 'code id_token token',
142 'code id_token',
143 'code token',
144 'code',
145 'id_token token',
146 'id_token',
147 'none',
148 ],
149
150
151 /*
152 * routes
153 *
154 * description: routing values used by the OP
155 * affects: routing
156 */
157 routes: {
158 authorization: '/auth',
159 certificates: '/certs',
160 check_session: '/session/check',
161 end_session: '/session/end',
162 introspection: '/token/introspection',
163 registration: '/reg',
164 revocation: '/token/revocation',
165 token: '/token',
166 userinfo: '/me',
167 },
168
```

```
169
170 /*
171 * scopes
172 *
173 * description: list of the scope values that the OP supports
174 * affects: discovery, authorization, ID Token claims, Userinfo claims
175 */
176 scopes: ['address', 'email', 'offline_access', 'openid', 'phone', '
    profile'],
177
178
179 /*
180 * subjectTypes
181 *
182 * description: list of the Subject Identifier types that this OP
    supports. Valid types include
183 *   pairwise and public.
184 * affects: discovery, registration, registration management, ID Token
    and Userinfo sub claim
185 *   values
186 */
187 subjectTypes: ['public'],
188
189
190 /*
191 * pairwiseSalt
192 *
193 * description: Salt used by OP when resolving pairwise ID Token and
    Userinfo sub claim value
194 * affects: ID Token and Userinfo sub claim values
195 */
196 pairwiseSalt: '',
197
198
199 /*
200 * tokenEndpointAuthMethods
201 *
202 * description: list of Client Authentication methods supported by this
    OP's Token Endpoint
203 * affects: discovery, client authentication for token, introspection and
     revocation endpoints,
204 *   registration, registration management
205 */
206 tokenEndpointAuthMethods: [
207 'none',
208 'client_secret_basic',
209 'client_secret_jwt',
```

```
'client_secret_post',
'private_key_jwt',
],


/*
* ttl
*
* description: expirations (in seconds) for all token types
* affects: tokens
*/
ttl: {
AccessToken: 2 * 60 * 60, // 2 hours in seconds
AuthorizationCode: 10 * 60, // 10 minutes in seconds
ClientCredentials: 10 * 60, // 10 minutes in seconds
IdToken: 2 * 60 * 60, // 2 hours in seconds
RefreshToken: 30 * 24 * 60 * 60, // 30 days in seconds
},


/*
* postLogoutRedirectUri
*
* description: URL to which the OP redirects the User-Agent when no
    post_logout_redirect_uri
*   is provided by the RP
* affects: session management
*/
postLogoutRedirectUri: '/?loggedOut=true',


/*
* logoutSource
*
* description: HTML source to which a logout form source is passed when
    session management
*   renders a confirmation prompt for the User-Agent.
* affects: session management
*/
logoutSource(form) {
// this => koa context;
this.body = `<!DOCTYPE html>
<head>
<title>Logout</title>
</head>
<body>
${form}
```

```
255 Voce deseja fazer logout tambem do Provedor OpenID
256 <button type="submit" form="op.logoutForm" name="logout" value="yes">Sim
       </button>
257 <button type="submit" form="op.logoutForm">Nao, por favor</button>
258 </body>
259 </html>‘;
260 },
261
262
263 /*
264 * uniqueness
265 *
266 * description: function resolving whether a given value with expiration
       is presented first time
267 * affects: client_secret_jwt and private_key_jwt client authentications
268 */
269 uniqueness(jti, expiresAt) {
270 // this => koa context;
271 if (cache.get(jti)) return Promise.resolve(false);
272
273 cache.set(jti, true, (expiresAt - epochTime()) * 1000);
274
275 return Promise.resolve(true);
276 },
277
278
279 /*
280 * renderError
281 *
282 * description: helper used by the OP to present errors which are not
       meant to be ’forwarded’ to
283 *   the RP’s redirect_uri
284 * affects: presentation of errors encountered during authorization
285 */
286 renderError(error) {
287 // this => koa context;
288 this.type = ’html’;
289
290 this.body = ‘<!DOCTYPE html>
291 <head>
292 <title>Oops! Algo errado aconteceu</title>
293 </head>
294 <body>
295 <h1>Oops! Algo errado aconteceu</h1>
296 <pre>${JSON.stringify(error, null, 4)}</pre>
297 </body>
298 </html>‘;
```

```
},


/*
* interactionUrl
*
* description: helper used by the OP to determine where to redirect User-Agent for necessary
*   interaction
* affects: authorization interactions
* note: can return both absolute and relative urls
*/
interactionUrl(interaction) { // eslint-disable-line no-unused-vars
// this => koa context;
try {
return url.parse(this.oidc.urlFor('interaction', { grant: this.oidc.uuid })).pathname;
} catch (err) {
return `/interaction/${this.oidc.uuid}`;
}
},

/*
* interactionCheck
*
* description: helper used by the OP as a final check whether the End-User should be sent to
*   interaction or not, the default behavior is that every RP must be authorized per session
* how to: return false if no interaction should be performed
*         return an object with relevant error, reason, etc. when interaction should be requested
* affects: authorization interactions
* note: can return a Promise
*/
interactionCheck() {
// this => koa context;
if (!this.oidc.session.sidFor(this.oidc.client.clientId)) {
return {
error: 'consent_required',
error_description: 'client not authorized for End-User session yet',
reason: 'client_not_authorized',
};
}

return false;
},
```

```
341
342
343 /*
344 * findById
345 *
346 * description: helper used by the OP to load your account and retrieve
      it's avaialble claims
347 * affects: authorization, authorization_code and refresh_token grants,
      id token claims
348 * note: The whole method should return a Promise and #claims() can
      return a Promise too
349 */
350 findById(id) {
351 // this => koa context;
352 return Promise.resolve({
353 accountId: id,
354 claims() { return { sub: id }; },
355 });
356 },
357
358 /*
359 * unsupported
360 *
361 * description: fine-tune the algorithms your provider should support by
      further omitting values
362 *   from the respective discovery properties
363 * affects: signing, encryption, discovery, client validation
364 */
365 unsupported: {
366 idTokenEncryptionAlgValues: [],
367 idTokenEncryptionEncValues: [],
368 idTokenSigningAlgValues: [],
369 requestObjectEncryptionAlgValues: [],
370 requestObjectEncryptionEncValues: [],
371 requestObjectSigningAlgValues: [],
372 tokenEndpointAuthSigningAlgValues: [],
373 userinfoEncryptionAlgValues: [],
374 userinfoEncryptionEncValues: [],
375 userinfoSigningAlgValues: [],
376 },
377
378
379 /*
380 * adapter
381 *
382 * description: TODO
383 * -> see https://github.com/panva/node-oidc-provider/blob/master/docs/
```

```
   configuration.md#persistance
*/
// adapter: MemoryAdapter,

adapter: MyAdapter,
/*
* refreshTokenRotation
*
* description: configures if and how the OP rotates refresh tokens after
    they are used
* affects: refresh token rotation and adjacent revocation
* supported values:
*    'none' - refresh tokens are not rotated and their initial
   expiration date is final
*    'rotateAndConsume' - refresh tokens are rotated when used, current
   token is marked as
*                         consumed and new one is issued with new TTL,
   when a consumed refresh
*                         token is encountered an error is returned
   instead and the whole token
*                         chain (grant) is revoked.
*/
refreshTokenRotation: 'none',
};
```

Código Fonte B.3 – lib/helpers/default.js

B.4 SETTINGS.JS

```
/* eslint-disable max-len */

'use strict';


const pkg = require('../package.json');

module.exports.config = {
acrValues: ['session', 'urn:mace:incommon:iap:bronze'],
cookies: {
long: { signed: true },
short: { signed: true },
},
discovery: {
service_documentation: pkg.homepage,
version: pkg.version,
},
claims: {
amr: null,
address: ['address'],
email: ['email', 'email_verified'],
phone: ['phone_number', 'phone_number_verified'],
profile: ['birthdate', 'family_name', 'gender', 'given_name', 'locale',
    'middle_name', 'name',
'nickname', 'picture', 'preferred_username', 'profile', 'updated_at', '
    website', 'zoneinfo'],
politicas: ['politicas'],
},
features: {
devInteractions: false,
claimsParameter: true,
clientCredentials: true,
encryption: true,
introspection: true,
registration: true,
registrationManagement: false,
request: true,
requestUri: true,
revocation: true,
sessionManagement: true,
backchannelLogout: true,
oauthNativeApps: true,
pkce: { skipClientAuth: true },
},
subjectTypes: ['public', 'pairwise'],
pairwiseSalt: 'da1c442b365b563dfc121f285a11eedee5bbff7110d55c88',
```

```
interactionUrl: function interactionUrl(interaction) { // eslint-disable-line no-unused-vars
// this => koa context;
return `/interaction/${this.oidc.uuid}`;
},
};

module.exports.clients = [{
client_id: 'oidcCLIENT',
client_secret: '91c0fabd17a9db3cfe53f28a10728e39b7724e234ecd78dba1fb05b909fb4ed98c476afc50a634d52808ad3cb2ea744bc8c3b45b7149ec459b5c416a6e8db242',
grant_types: ['client_credentials', 'refresh_token', 'authorization_code'],
redirect_uris: ['http://sso-client.dev/providers/7/open_id', 'http://sso-client.dev/providers/8/open_id'],
}];

module.exports.certificates = [{
d: 'VEZOsY07JTFzGTqv6cC2Y32vsfChind2I_TTuvV225_-0zrSej3XLRg8iE_u0-
3GSgiGi4WImmTwmEgLo4Qp3uEcxCYbt4NMJC7fwT2i3dfRZjtZ4yJwFl0SIj8TgfQ8
ptwZbFZUlcHGXZIr4nL8GXyQT0CK8wy4COfmymHrrUoyfZA154ql_OsoiupSUCRcKV
vZj2JHL2KILsq_sh_l7g2dqAN8D7jYfJ58MkqlknBMa2-zi5I0-1JUOwztVNml_zGr
p27UbEU60RqV3GHjoqwI6m01U7K0a8Q_SQAKYGqgepbAYOA-P4_TL15KC4-WWBZu_r
VfwgSENwWNEhw8oQ',
dp: 'E1Y-SN4bQqX7kP-bNgZ_gEv-pixJ5F_EGocHKfS56jtzRqQdTurrk4jIVpI
-ZITA881WAHxjD-OaoJUh9Jupd_lwD5Si80PyVxOMI2xaGQiF0lbKJfD38Sh8frRpg
elZVaK_gm834B6SLfxKdNsP04DsJqGKktODF_fZeaGFPH0',
dq: 'F90JPxevQYOlAgEH0TUt1-3_hyxY6cfPRU2HQBaahyWrtCWpaOzenKZnvGFZdg
-BuLVKjCchq3G_70OLE-XDP_ol0UTJmDTT-WyuJQdEMpt_WFF9yJGoeIu8yohfeLatU
-67ukjghJ0s9CBzNE_LrGEV6Cup3FXywpSYZAV3iqc',
e: 'AQAB',
kty: 'RSA',
n: 'xwQ72P9z9OYshiQ-ntDYaPnnfwG6u9JAdLMZ5o0dmjlcyrvwQRdoFIKPnO65Q8mh6F_
LDSxjxa2Yzo_wdjhbPZLjfUJXgCzm54cClXzT5twzo7lzoAfaJlkTsoZc2HFWqmcri0Buzm
TFLZx2Q7wYBm0pXHmQKF0V-C1O6NWfd4mfBhbM-I1tHYSpAMgarSm22WDMDx-WWI7TEzy2Q
haBVaENW9BKaKkJklocAZCxk18WhR0fckIGiWiSM5FcU1PY2jfGsTmX505Ub7P5Dz75Ygqr
utd5tFrcqyPAtPTFDk8X1InxkkUwpP3nFU5o50DGhwQolGYKPGtQ-ZtmbOfcWQ',
p: '5wC6nY6Ev5FqcLPCqn9fC6R9KUuBej6NaAVOKW7GXiOJA
q2WrileGKfMc9kIny20zW3uWkRLm-O-3Yzze1zFpxmqvsvCxZ5
ERVZ6leiNXSu3tez71ZZwp0O9gys4knjrI-9w46l_vFuRtjL6XEeFfH
EZFaNJpz-lcnb3w0okrbM',
q: '3I1qeEDslZFB8iNfpKAdWtz_Wzm6-jayT_V6aIvhvMj5mnU-Xpj75zLPQS
Ga9wunMlOoZW9w1wDO1FVuDhwzeOJaTm-Ds0MezeC4U6nVGyyDHb4CUA3ml2tz
t4yLrqGYMT7XbADSvuWYADHw79OFjEi4T3s3tJymhaBvy1ulv8M',
qi: 'wSbXte9PcPtr788e713KHQ4waE26CzoXx-JNOgN0iqJMN6C4_XJEX-
cSvCZDf4rh7xpXN6SGLVd5ibIyDJi7bbi5EQ5AXjazPbLBjRthcGXsIuZ3AtQy
```

```
88  ROCEWNSdM7EyM5TRdyZQ9kftfz9nI03guW3iKKASETqX2vh0Z8XRjyU',
89  use: 'sig',
90  }, {
91  d: 'QCRapETx5p-iZ2eg7TCK9LWmIB38CZvbiDwjaDZPGM-hOvgYiD5HdsKmKy
92  c40R6XWduAZgINQ8WfPQ2ms6Xfdnwdy7u0scy0lhDQlvWrF-FU1SQ3t4jEzI0
93  2zDZ3EeZOg3car5XpP5PRVBkC9yxmVYpQN1MRdNCwfkeeMSJtUf9LwVZXdfkk6
94  VN0i_FjXH77cDhoPk4WEayWWSKLNeq1Hu9bhkXwakDoveHtpdgVoWR6H8RVZc
95  IndzFS-tKxRUkUum6cr4V9zrcH7vI28W9M0LT2EU5omrs0MNIBjBzsxH_osYOo
96  l0EawTPIjejhnQSM6akQlKWzQEv6iPXKvHF5',
97  dp: 'qNx5nc47pEfF_hfn72Xh3Tn_cQFIC-3dsWUQkZt9P7ve4Ue30S-sICXb9sEa
98  eJ3oF2G_jGW6sLu9o7pkyKBQnaoBbZrDfCrPqF_5wFkR14SSV6PteFCMuooGeyrC
99  5RNnoziFAX0roV8mpIyUlr1_NuItMqIBcKprvIUaCPQbaGs',
100 dq: 'HUUQCZO4Xqe-0tknlk0RUzqFOKvdHTmMeAAKb74_aG5R9_YmfE-U_JO5SwSzer
101 0BPfbIQ8w6NsU1MTiptW9DPkkzsZg1xjuYWvdfjsfuJUN0pq1QNxotEWna5YpZP1n
102 XBXO68LbBU84KU7RMgOd8mSGI9zANoCOPgEzWP5FiY-k',
103 e: 'AQAB',
104 kty: 'RSA',
105 n: 'mXauIvyeUFA74P2vcmgAWSCMw6CP6-MJ6EvFuRARfLLJEi49AzQvJl_
106 4pwDvLkZcCqS7OqPE1ufNyDH6oQPEc7JuukHMY02EgwqHjJ6GG6FQqJuiWlKB_
107 l-7c9y9r4bh4r58xdZc6T5dFVSNT2VcIVoSjq9Vmzw
108 paTKCUyVeZYHZhnLfWMm9rKU5WSz75siG-_jbudItsfhEwA59kvi4So2IV9Tx
109 HwW50i4IcTB1gXwG1olNgiX3-Mq1Iw5VGPzMo2hQXI3q1y-
110 ZjhSwhvG5dje9J8htBEWdVYk4f6cv19IE9gEx7T-2
    vIVw5FCpAmmfFuRebec49c7zjfr0EyTI4w',
111 p: '65DqEX3ClDSQnH2mGDcnOWgy2M66bNqGA8DSPxHiNc_alLPD_
112 3PJh0Q4PVIZeqi5LBxaHtEl28EFCaPP_2rfM22Ki32jY8WVSdoCMT5
113 sD8tOGCF24zChGLHTRKlq1N7xffiUUUrzFTJE8_23r1GoGxRU4b81eiIU7iLxjepYv88',
114 q: 'psaPv_vFHchNJ-225
    VnGwvfGYYKCoNgFBSFlLO4KvjPHVwcLDLuFlw1DJMS361s1cDhngglNvl
115 -dgATfav0fJlYiVoh3Q0qHjsAW7Zq07nPeN4IW-n0tWcLBccLPj
116 -S4oNiGWmax2IvpEdHMkY-AhpGbgOQEz9FXYLyHa6vStq0',
117 qi: 'bqrEvOsv-
    dzuhh6dbQfxxq6aQn_hnYi89pFcThfkCEQmHEFqOoxR121QlqQmVPoZoqN7kzxvf
118 _qFEyjvW9I-JuUlz4CbknhwHQrniUf3YwFtJ3SouqmAEL01c5pcB52Bh40J_xmd
119 -Z7whjnwNMCDPQXTEtQZh4BXh4TWrq1N4No',
120 use: 'enc',
121 }, {
122 crv: 'P-256',
123 d: 'K9xfPv773dZR22TVUB80xouzdF7qCg5cWjPjkHyv7Ws',
124 kty: 'EC',
125 x: 'FWZ9rSkLt6Dx9E3pxLybhdM6xgR5obGsj5_pqmnz5J4',
126 y: '_n8G69C-A2Xl4xUW2lF0i8ZGZnk_KPYrhv4GbTGu5G4',
127 }, {
128 crv: 'P-384',
129 d: 'qdPGmba6NZMJYCF1UGOfHCKbc0CBtGss0zPVegBdw0sky6H7FamJcckfwItfS0HT',
130 kty: 'EC',
131 x: 'ij8LkaIQ-QkODmWucHJ7PWEtnnqlyd-iQU6fZcLoEEh-ScWULv4ggleNCWHdULtZ',
```

```
132 y: 'tTq_5IMhNcPR6L4W7T0ATPofr0wNRpHOZEIcTLk6DBqb5o0ZLo3g0r1ZUxNdAU3W',
133 }, {
134 crv: 'P-521',
135 d: 'P9pF8q_vq97UloR9C4d05mGeCN3cQ4AP9p3kMubrAVuzUi
136 eeNFLEjRseWmXftsk4sVFxnM9Roxt5Sy1fN5VgeWc',
137 kty: 'EC',
138 x: 'AamstoAouLxrWi6WHt903QR7NMpK4NszB5mNEFqLqaCxRXhPwrq_BG5R-7
    UP41cUIF38TQCePJpGLnoC5amCJNy3',
139 y: 'AUguNqeqkhVSrmolR58H4J26S58XinSN3kcnoIl75iHMKRM
140 JDXBI9J4lBHALVn6i0zc9N9ucQAb8kmOXfObga_9J',
141 }];
142
143 module.exports.integrityKeys = [{
144 kty: 'oct',
145 k: 'TQcYPUBhQvg68X_7hwRyVyRNxpdkN-xsrOqVjgaCi-
    PsbeBrPHOYHcc4lPDIzdfEtdxpQ7pOMJn5__pW-NJx9w',
146 alg: 'HS512',
147 }];
```

Código Fonte B.4 – example/settings.js

B.5 ACCOUNT.JS

```
'use strict';

const store = new Map();
const logins = new Map();
// const logins = require('./adapters/mongodb');
const uuid = require('uuid');


class Account {
constructor(id) {
this.accountId = id || uuid();
store.set(this.accountId, this);
}

claims() {
return {
address: {
country: '000',
formatted: '000',
locality: '000',
postal_code: '000',
region: '000',
street_address: '000',
},
politicas: {
PI_SI_PP_FI: '1',
PCP_SI_PP_FI: '1',
LO_SI_PP_FI: '1',
AH_SI_PP_FI: '1',
RS_SI_PP_FI: '1',
PI_SC_PP_FI: '1',
PCP_SC_PP_FI: '1',
LO_SC_PP_FI: '1',
AH_SC_PP_FI: '1',
RS_SC_PP_FI: '1',
PI_CO_PP_FI: '1',
PCP_CO_PP_FI: '1',
LO_CO_PP_FI: '1',
AH_CO_PP_FI: '1',
RS_CO_PP_FI: '1',
PI_SI_SP_FI: '1',
PCP_SI_SP_FI: '1',
LO_SI_SP_FI: '1',
AH_SI_SP_FI: '1',
RS_SI_SP_FI: '1',
PI_SC_SP_FI: '1',
```

```
47 PCP_SC_SP_FI: '1',
48 LO_SC_SP_FI: '1',
49 AH_SC_SP_FI: '1',
50 RS_SC_SP_FI: '1',
51 PI_CO_SP_FI: '1',
52 PCP_CO_SP_FI: '1',
53 LO_CO_SP_FI: '1',
54 AH_CO_SP_FI: '1',
55 RS_CO_SP_FI: '1',
56 PI_SI_TP_FI: '1',
57 PCP_SI_TP_FI: '1',
58 LO_SI_TP_FI: '1',
59 AH_SI_TP_FI: '1',
60 RS_SI_TP_FI: '1',
61 PI_SC_TP_FI: '1',
62 PCP_SC_TP_FI: '1',
63 LO_SC_TP_FI: '1',
64 AH_SC_TP_FI: '1',
65 RS_SC_TP_FI: '1',
66 PI_CO_TP_FI: '1',
67 PCP_CO_TP_FI: '1',
68 LO_CO_TP_FI: '1',
69 AH_CO_TP_FI: '1',
70 RS_CO_TP_FI: '1',
71 PI_SI_PP_CO: '1',
72 PCP_SI_PP_CO: '1',
73 LO_SI_PP_CO: '1',
74 AH_SI_PP_CO: '1',
75 RS_SI_PP_CO: '1',
76 PI_SC_PP_CO: '1',
77 PCP_SC_PP_CO: '1',
78 LO_SC_PP_CO: '1',
79 AH_SC_PP_CO: '1',
80 RS_SC_PP_CO: '1',
81 PI_CO_PP_CO: '1',
82 PCP_CO_PP_CO: '1',
83 LO_CO_PP_CO: '1',
84 AH_CO_PP_CO: '1',
85 RS_CO_PP_CO: '1',
86 PI_SI_SP_CO: '1',
87 PCP_SI_SP_CO: '1',
88 LO_SI_SP_CO: '1',
89 AH_SI_SP_CO: '1',
90 RS_SI_SP_CO: '1',
91 PI_SC_SP_CO: '1',
92 PCP_SC_SP_CO: '1',
93 LO_SC_SP_CO: '1',
```

```
94  AH_SC_SP_CO: '1',
95  RS_SC_SP_CO: '1',
96  PI_CO_SP_CO: '1',
97  PCP_CO_SP_CO: '1',
98  LO_CO_SP_CO: '1',
99  AH_CO_SP_CO: '1',
100 RS_CO_SP_CO: '1',
101 PI_SI_TP_CO: '1',
102 PCP_SI_TP_CO: '1',
103 LO_SI_TP_CO: '1',
104 AH_SI_TP_CO: '1',
105 RS_SI_TP_CO: '1',
106 PI_SC_TP_CO: '1',
107 PCP_SC_TP_CO: '1',
108 LO_SC_TP_CO: '1',
109 AH_SC_TP_CO: '1',
110 RS_SC_TP_CO: '1',
111 PI_CO_TP_CO: '1',
112 PCP_CO_TP_CO: '1',
113 LO_CO_TP_CO: '1',
114 AH_CO_TP_CO: '1',
115 RS_CO_TP_CO: '1',
116 PI_SI_PP_SA: '1',
117 PCP_SI_PP_SA: '1',
118 LO_SI_PP_SA: '1',
119 AH_SI_PP_SA: '1',
120 RS_SI_PP_SA: '1',
121 PI_SC_PP_SA: '1',
122 PCP_SC_PP_SA: '1',
123 LO_SC_PP_SA: '1',
124 AH_SC_PP_SA: '1',
125 RS_SC_PP_SA: '1',
126 PI_CO_PP_SA: '1',
127 PCP_CO_PP_SA: '1',
128 LO_CO_PP_SA: '1',
129 AH_CO_PP_SA: '1',
130 RS_CO_PP_SA: '1',
131 PI_SI_SP_SA: '1',
132 PCP_SI_SP_SA: '1',
133 LO_SI_SP_SA: '1',
134 AH_SI_SP_SA: '1',
135 RS_SI_SP_SA: '1',
136 PI_SC_SP_SA: '1',
137 PCP_SC_SP_SA: '1',
138 LO_SC_SP_SA: '1',
139 AH_SC_SP_SA: '1',
140 RS_SC_SP_SA: '1',
```

```
141 PI_CO_SP_SA: '1',
142 PCP_CO_SP_SA: '1',
143 LO_CO_SP_SA: '1',
144 AH_CO_SP_SA: '1',
145 RS_CO_SP_SA: '1',
146 PI_SI_TP_SA: '1',
147 PCP_SI_TP_SA: '1',
148 LO_SI_TP_SA: '1',
149 AH_SI_TP_SA: '1',
150 RS_SI_TP_SA: '1',
151 PI_SC_TP_SA: '1',
152 PCP_SC_TP_SA: '1',
153 LO_SC_TP_SA: '1',
154 AH_SC_TP_SA: '1',
155 RS_SC_TP_SA: '1',
156 PI_CO_TP_SA: '1',
157 PCP_CO_TP_SA: '1',
158 LO_CO_TP_SA: '1',
159 AH_CO_TP_SA: '1',
160 RS_CO_TP_SA: '1',
161 PI_SI_PP_MI: '1',
162 PCP_SI_PP_MI: '1',
163 LO_SI_PP_MI: '1',
164 AH_SI_PP_MI: '1',
165 RS_SI_PP_MI: '1',
166 PI_SC_PP_MI: '1',
167 PCP_SC_PP_MI: '1',
168 LO_SC_PP_MI: '1',
169 AH_SC_PP_MI: '1',
170 RS_SC_PP_MI: '1',
171 PI_CO_PP_MI: '1',
172 PCP_CO_PP_MI: '1',
173 LO_CO_PP_MI: '1',
174 AH_CO_PP_MI: '1',
175 RS_CO_PP_MI: '1',
176 PI_SI_SP_MI: '1',
177 PCP_SI_SP_MI: '1',
178 LO_SI_SP_MI: '1',
179 AH_SI_SP_MI: '1',
180 RS_SI_SP_MI: '1',
181 PI_SC_SP_MI: '1',
182 PCP_SC_SP_MI: '1',
183 LO_SC_SP_MI: '1',
184 AH_SC_SP_MI: '1',
185 RS_SC_SP_MI: '1',
186 PI_CO_SP_MI: '1',
187 PCP_CO_SP_MI: '1',
```

```
188 LO_CO_SP_MI: '1',
189 AH_CO_SP_MI: '1',
190 RS_CO_SP_MI: '1',
191 PI_SI_TP_MI: '1',
192 PCP_SI_TP_MI: '1',
193 LO_SI_TP_MI: '1',
194 AH_SI_TP_MI: '1',
195 RS_SI_TP_MI: '1',
196 PI_SC_TP_MI: '1',
197 PCP_SC_TP_MI: '1',
198 LO_SC_TP_MI: '1',
199 AH_SC_TP_MI: '1',
200 RS_SC_TP_MI: '1',
201 PI_CO_TP_MI: '1',
202 PCP_CO_TP_MI: '1',
203 LO_CO_TP_MI: '1',
204 AH_CO_TP_MI: '1',
205 RS_CO_TP_MI: '1',
206 DISSEMINACAO: '1',
207 CRIPTOGRAFIA: '1',
208 },
209 birthdate: '1987-10-16',
210 email: 'teste@teste.com',
211 email_verified: false,
212 family_name: 'Teste',
213 gender: 'male',
214 given_name: 'Teste',
215 locale: 'en-US',
216 middle_name: 'Teste',
217 name: 'Teste',
218 nickname: 'teste',
219 phone_number: '+49 000 000000',
220 phone_number_verified: false,
221 picture: 'http://lorempixel.com/400/200/',
222 preferred_username: 'Jdawg',
223 profile: 'https://teste.com',
224 sub: this.accountId,
225 updated_at: 1454704946,
226 website: 'http://teste.com',
227 zoneinfo: 'Europe/Berlin',
228
229 };
230 }
231
232 static findByLogin(login) {
233 if (!logins.get(login)) {
234 logins.set(login, new Account());
```

```
235 }
236
237 return Promise.resolve(logins.get(login));
238 }
239
240 static findById(id) {
241 if (!store.get(id)) new Account(id); // eslint-disable-line no-new
242 return Promise.resolve(store.get(id));
243 }
244 }
245
246 module.exports = Account;
```

Código Fonte B.5 – example/account.js

C Artigo

Implementação de Sticky Policies no protocolo *OpenID Connect*

Lucas M. Monfardine, Thiago A. Avila

Curso de Bacharelado em Sistemas de Informação – Departamento de Informática e Estatística– Universidade Federal de Santa Catarina (UFSC) – 88040-900 - Florianópolis – SC – Brasil

lucasmmonfardine@gmail.com, jakavila@gmail.com

Abstract. *The identity of an organization or person is given through individual characteristics which each is known. Given their sensitivity and need for safety, greater care must be taken while dealing with this information, especially when it includes financial or medical data. Through this paper, the extension to the OpenID Connect protocol was performed on an existing implementation in node.js, allowing to send privacy policies in the context of the service use, attached to the basic information contained in the token. For this, the Sticky Policies context was used, which adds the privacy policies to the user data normally sent by the OpenID protocol.*

Resumo. *A identidade de uma organização ou pessoa consiste em características individuais, através das quais essa pessoa ou organização são conhecidas. Devido à sua sensibilidade e necessidade de segurança, deve-se ter maior cuidado no manuseio destas informações, principalmente quando incluem dados financeiros ou médicos. Através deste trabalho, foi realizada a extensão do protocolo OpenID Connect em uma implementação já existente em node.js, permitindo o envio de políticas de privacidade no contexto da utilização do serviço, juntamente com as informações básicas contidas no token. Para tal, foi utilizado o contexto de Sticky Policies, que agrega as políticas de privacidade aos dados do usuário normalmente enviados pelo protocolo OpenID Connect.*

1. Introdução

Com o crescente mercado de sistemas web, deve crescer também o cuidado das organizações com os dados sensíveis dos utilizadores de seus serviços, em especial dados de identificação, também chamados de PII (personally identifiable information), enviados pelos usuários para as suas aplicações. Quando usuários compartilham seus dados com um serviço, devem ter controle sobre o destino destes e certeza que este destino será cumprido, além das regras quanto ao seu uso ou divulgação a terceiros.

As políticas de privacidade são os documentos que se destinam a ajudar o usuário a entender sobre o tratamento dos dados após a coleta. Através delas, o usuário é questionado ou informado sobre as informações coletadas, o uso dessas informações, o

tempo de retenção ou da divulgação das mesmas. O *OpenID Connect*, *Open Source* e baseado no protocolo OAuth 2.0, visa garantir autenticação e autorização entre um usuário e o serviço desejado. Para tanto, fornece as informações sobre o usuário final na forma de um token, que contém as informações básicas de perfil sobre o usuário.

O objetivo geral deste artigo consiste no levantamento de características sobre o padrão de implementação *OpenID Connect* e posterior elaboração de protótipos de Cliente e Provedor de Identidades que utilizam este formato, com extensão dos *Tokens* de ID, para que, com a utilização de *Sticky Policies*, sejam agregadas informações das políticas de privacidade do usuário.

2. Gerenciamento de Identidades

Para Stallings (2011), gerenciamento de identidades é uma abordagem centralizada e automatizada para prover acesso a recursos para usuários autorizados. O foco do gerenciamento de identidade é definir uma identidade para cada usuário (humano ou processo), associando atributos à identidade e reforçando a forma que a identidade de um usuário pode ser verificada.

O conceito central de um sistema de gerenciamento de identidades é o uso do *Single Sign-On* (SSO), que permite a um usuário acessar todos os recursos da rede depois de uma única autenticação. Um sistema de gerenciamento de identidades pode incorporar às identidades digitais atributos, além das informações de identificação e autorização. Um serviço de atributos gerencia a criação e manutenção destes atributos, como por exemplo, o endereço residencial de um usuário. O gerenciamento de identidades permite que o usuário informe este dado apenas uma vez, sendo este mantido em um único lugar e liberado aos recursos que solicitam esta informação de acordo com a autorização do usuário e de políticas de privacidade.

Uma federação de identidades divide o sistema em vários domínios administrativos. Cada domínio possui ao menos um provedor de identidades e vários provedores de serviço. Um usuário utiliza a identidade fornecida por seu provedor de identidades para acesso aos serviços fornecidos pelos provedores de serviço daquele domínio, além de serviços de domínios externos, desde que seja estabelecida uma relação de confiança prévia entre os provedores de identidades destes domínios. Assim, a identidade gerada por um provedor de identidades é reconhecida por outros provedores de identidades. [Bertino e Takahashi, 2011].

O objetivo é prover o compartilhamento de identidades digitais de forma que o usuário possa se autenticar apenas uma vez e então acessar aplicações e recursos entre múltiplos domínios. Como esses domínios são relativamente autônomos ou independentes, não há um controle central. Ao invés disso, organizações que cooperam entre si devem formar uma federação baseada em padrões acordados e múltiplos níveis de confiança para compartilhar identidades digitais de forma segura [Stallings, 2011].

3. *OpenID Connect*

De acordo com a documentação da ferramenta, o *OpenID Connect* 1.0 é definido como uma camada simples de identidade implementada sobre o protocolo OAuth 2.0. Ela permite que provedores de serviço, na condição de clientes, verifiquem a identidade de

um usuário final baseando-se na autenticação executada por um servidor de autorização, bem como obter informações básicas sobre o perfil do usuário que está se autenticando no sistema de uma forma interoperável utilizando o conceito de *REpresentational State Transfer* (REST) [*OpenID Connect*, 2016]. Sua implementação é dividida em módulos, cada um com uma finalidade específica.

As especificações do módulo *Core* do protocolo *OpenID Connect* 1.0 definem as funcionalidades principais do protocolo: autenticação construída sobre o protocolo OAuth 2.0 e o uso de requisições para a troca de informações sobre o usuário final. Além disso, nesse módulo são descritas as considerações sobre segurança e privacidade na utilização do protocolo *OpenID Connect* 1.0.

O protocolo *OpenID Connect* segue o fluxo detalhado na Figura 1:

1. A RP envia uma solicitação de autenticação para o OP;
2. O OP autentica o usuário final e obtém informações de autorização;
3. O OP responde com um *Token* ID e, ocasionalmente, um *token* de acesso;
4. A RP pode requisitar ao OP informações sobre o usuário.
5. O OP retorna informações sobre o usuário.

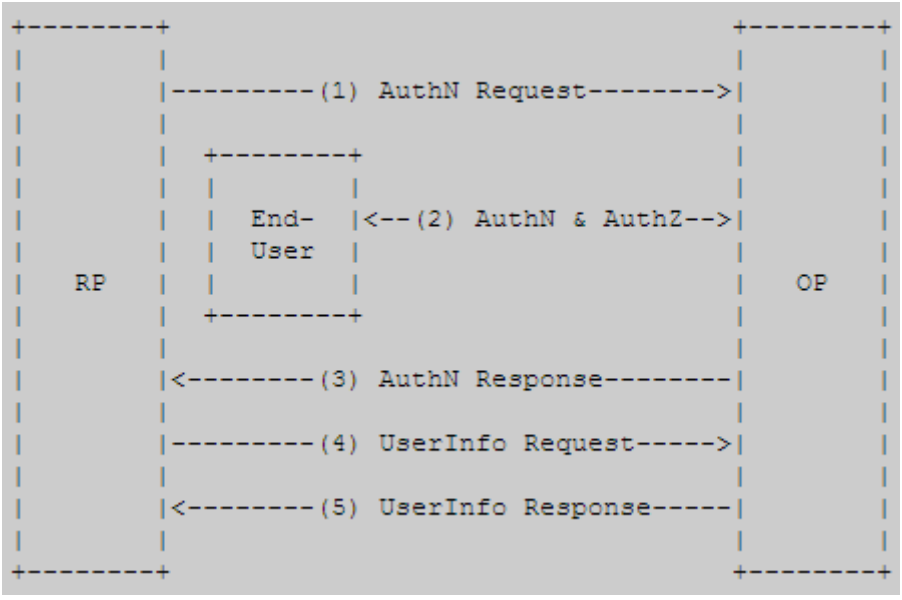

Figura 1: Fluxo do *OpenID Connect* (*OpenID Connect*)

4. Políticas de Privacidade

De acordo com Caramujo e Silva (2015), políticas de privacidade representam os termos que um usuário precisa aceitar para utilizar os serviços providos por uma organização. É definido por um conjunto de atributos e composto por um ou mais elementos chamados *Statements*, que são as afirmações que detalham as escolhas do usuário sobre o destino e tratamento dos dados. Este documento deve identificar quais os tipos de informações dos usuários serão gerenciadas e podem ser expostas.

Villarreal et. al. (2017) estabelecem as regras de política de privacidade, definindo inicialmente tuplas contendo os itens: Tipo de Dados (dT), propósito de uso para o dado (pR), tempo em horas de conservação dos dados (tM), o contexto da utilização dos atributos (cN), informação sobre a notificação do usuário quanto ao uso do dado (nT), informação sobre criptografia dos dados (cP). Esta tupla é montada tendo como destino um Provedor de Serviço específico. Para tanto, apresenta-se da seguinte forma:

SP.Regra = [dT, pR, tM, cN. nT, cP]

onde:

dT - tipo de dados (nome, cpf, endereço)
pR - propósito (melhoria de serviço, científico, comercial, governamental)
tM - tempo de conservação dos dados
cN - contexto (Financeiro, Compras online, Saúde, Militar)
nT - notificação (1 para notificar, 0 para não)
cP - criptografia(1 para cifrar, 0 para não)

Exemplo 1:

Loja1.Regra = [nome,cpf;pagamento;48;compras;1;1]

No exemplo 1, os tipos de dados (nome e CPF) devem ser utilizados apenas para fins de pagamento, pela Loja1 (Prestador do Serviço), mantendo esses dados por 48 horas, sendo utilizados apenas para compras online, notificando o usuário quanto ao uso e transmitindo-os de forma cifrada.

Para uma melhor apresentação e padronização, Villarreal et. al. (2017) definem categorias e valores padrão para cada uma delas. Além disso, estende a estrutura, para o refinamento das preferências pessoais dos usuários, adicionando beneficiários:

Tipo de Dados são divididos em:

- Informação Pessoal (PI)
- Preferências e Caracterısticas Pessoais (PCP)
- Localizacao (LO)
- Atividades e Hábitos (AH)
- Relacoes (RS)

Propósito de uso são divididos em:

- Melhoria de Serviço (SI)
- Cientıfico (SC)
- Comercial (CO)

Beneficiários, divididos em:

- PII Principal (PP)
- Provedor de Serviço (SP)
- Terceira Parte (TP)

Contexto, divididos em:

- Financeiro (FI)
- Compras online (CO)
- Saúde (SA)
- Militar (MI)

Disseminação de dados: 1 ou 0

Criptografia: 1 ou 0

5. *Sticky Policies*

As *Sticky Policies*, em seu contexto original, são as políticas de privacidade que são compartilhadas, quando estas são anexadas juntamente com os dados e que descrevem como estes devem ser tratados. Ou seja, são as restrições ou condições que definem a autorização dos usuários quanto à utilização destes dados [Pearson, Mont, 2011].

Em *Privacy Patterns* (2017), tem-se que as *Sticky Policies* permitem a propagação das políticas entre organizações de confiança, mantendo a correta aplicação das mesmas, além de prover rastreabilidade. Por outro lado, *Privacy Patterns* (2017) declara como problemático o aumento no tamanho dos dados e como principal obstáculo para a implementação, a dificuldade na atualização das políticas.

6. Modelo Inicial

De acordo com Stallings (2011), o gerenciamento de identidades usa padrões que são essenciais para uma troca segura de identidades através de diferentes domínios e sistemas heterogêneos. Um dos desafios do gerenciamento de identidades é garantir aos usuários que os dados providos por eles a um serviço serão corretamente tratados pelo serviço e pelas outras partes.

Para que isso ocorra, é utilizado o conceito de políticas de privacidade e as definições que dela decorram para determinar padrões a serviços a fim de prover um serviço seguro [Stallings 2011]

A criação das políticas de privacidade se dá através da combinação de todos os tipos de dados, todos os propósitos de uso, todos os beneficiários e todos os contextos descritos anteriormente. Na figura 2, são apresentadas de algumas destas combinações incluídas em um *token* ID.

Figura 2. Exemplo de política de privacidade (VILLARREAL et. al., 2017)

7. Desenvolvimento

A proposta inicial deste trabalho compreende os seguintes pontos:

- Criação de um perfil de usuário e suas políticas de privacidade, compreendendo todos os tipos de dados, propósitos, contextos de uso, beneficiários, definição sobre a disseminação e criptografia dos dados;
- Definição e criação da estrutura para envio das políticas de privacidade do usuário definidas previamente para os provedores de serviço, juntamente com os dados de identificação;
- Implementação dessa estrutura em um provedor de dados baseado no protocolo *OpenID Connect*, permitindo o envio das políticas com o *Token* ID;
- Implementação de modificações em um modelo de cliente baseado em *OpenID Connect*, permitindo que este receba um *token* contendo, além das informações da estrutura original, as políticas de disseminação definidas pelo usuário.

Para o cumprimento destes pontos, foram utilizadas como base para o provedor de identidades e do cliente *OpenID Connect*, as implementações de Skokan (2017), em Node.js. Após a implantação de ambos os sistemas foram realizadas diversas alterações no código original de Skokan para realizar o proposto neste artigo.

Na classe app.js do cliente implementado por Skokan (2017), é definida a requisição de autorização que será enviada ao provedor de identidades. Esta requisição está descrita na figura 1, como o passo (1). Nela, são adicionadas as reivindicações ou *Claims* que serão tratadas pelo cliente quando este receber a resposta. Uma das adaptações desenvolvidas neste trabalho consiste na adição de um objeto *claim* “políticas”, como um dos *claims* constantes nas informações de usuário (*UserInfo*).

Na classe client.js, do cliente de Skokan, está o trecho de código que define o passo (4) da figura 1. Nesta etapa é montada a requisição para *UserInfo* ou informações sobre o usuário. Uma das atividades no desenvolvimento deste trabalho foi a adição de

código para incluir na requisição a *claim* de políticas, necessárias para que o cliente solicite ao provedor de identidades *OpenID* informações sobre este campo.

Na classe defaults.js, do provedor de identidades implementado por Skokan, são definidos os nomes das *claims* ou reivindicações habilitadas no provedor de identidades OpenID e que podem ser fornecidas aos seus clientes. Como desenvolvimento das modificações propostas neste trabalho, está a adição do objeto políticas a esta linha, permitindo aos clientes que as solicitem a este provedor e tornando o provedor apto a entregá-las.

O provedor de identidades OpenID deve de alguma maneira persistir as contas de usuários cadastrados para uso posterior. Esta conta deve ter uma propriedade *accountId*, bem como a função *claims*(), para retornar um objeto com reivindicações que correspondem às reivindicações que ele suporta.

As contas de usuários podem estar salvas em um banco de dados ou em classes com as informações em memória. Para o desenvolvimento deste trabalho, foi utilizada uma classe account.js, para definir as informações do usuário de teste. Nele, foram registradas as informações pessoais do usuário, bem como as políticas hipoteticamente definidas por ele.

A figura 3 refere-se à resposta recebida pelo cliente com a *UserInfo Response*, que contém as informações pessoais do usuário. Neste mesmo espaço estão as políticas de privacidade que são foco deste trabalho, enviadas juntamente com o Token ID. Na imagem, são apresentados os mesmos dados e políticas definidos em account.js. As políticas são suprimidas para melhor apresentação. Apesar disso, na execução, todas as 182 políticas definidas foram mostradas e apresentadas do lado do cliente.

Figura 3. Userinfo Response (Adaptado de Skokan, 2017)

8. Trabalhos Futuros

Ao longo da elaboração deste trabalho foram identificados alguns itens que podem estender os resultados obtidos:

- Implementação de métodos que permitam ao provedor de identidades *OpenID Connect* do protótipo elaborado cadastrar usuários diversos, permitindo ainda que cada um defina suas políticas de privacidade e estes dados sejam salvos no banco de dados, além da identificação de usuário e senha na tela de autorização.
- Implementação das modificações que permitem a transferência de *Sticky Policies* em outros modelos de provedor e cliente *OpenID Connect*, garantindo intercompatibilidade entre sistemas e linguagens.
- Elaboração de documentação e casos de uso para submissão ao OpenID Foundation, entidade mantenedora do protocolo *OpenID Connect*, sugerindo a adesão do modelo de *Sticky Policies* como funcionalidade oficial e suportada pelo protocolo.
- Estudo de modelo conceitual para implantação em clientes *OpenID Connect* que certifiquem o usuário final de que suas definições quanto à propagação e disseminação de seus dados serão cumpridas pelo cliente que as recebe.

9. Conclusão

De acordo com Kallela (2008), a possibilidade de um usuário interagir com diversos serviços utilizando a mesma credencial, utilizando o conceito de Identidade Federada, tem se difundido nos últimos anos. Isso reforça a necessidade já fundamentada de adaptar as tecnologias existentes para que estas forneçam suporte aos novos métodos de autenticação, garantindo opções modernas e seguras para o uso de redes federadas de autenticação.

Ao mesmo tempo, a privacidade é apontada como a questão mais preocupante por consumidores de lojas online nos Estados Unidos, estando à frente de itens como spam e segurança. A divulgação das práticas de privacidade por um site reduz significativamente as preocupações de seus usuários, tornando o ambiente mais confiável para eles [Benassi, 1999].

Considerando a evidente necessidade de privacidade e controle sobre os dados disseminados no ambiente de nuvem, tecnologia que vem se ampliando exponencialmente, acredita-se que a necessidade de se pensar em formas de garantir ao usuário final que este defina qual destino dar a suas informações se torna essencial, e a utilização do modelo de *Sticky Policies* é uma ótima forma de prover as garantias de segurança e privacidade que o usuário busca.

Referências

Benassi, P. TRUSTe: An Online Privacy Seal Program. ACM, New York, NY, p 56-59, 1999.

Bertino, E.; Takahashi, K. Identity Management: Concepts, Technologies, and Systems. Artech House, Boston, MA, 2011.

Caramujo, J.; da Silva, A. R. Analyzing Privacy Policies based on a Privacy-Aware Profile: The Facebook and LinkedIn case studies. 2015 IEEE 17th Conference on Business Informatics, 2015.

Kallela, J. Federated Identity Management Solutions. Seminar on Internetworking, TKKT-110.5190, 2008.

OpenID Connect. *OpenID Connect*. 2016 Disponível em: <http://openid.net/connect/>. Acesso em 29 de maio de 2016.

Pearson, S.; Mont, M. C. Sticky Policies: An Approach for Managing Privacy across Multiple Parties. 2011. Disponível em: <http://bit.ly/2rdyvbf> Acesso em 12 de Maio de 2017.

Privacy Patterns. Sticky Policies. Disponível em: <http://bit.ly/2qCSmSp> Acesso em 11 de Maio de 2017.

Skokan, F. Identity, *OpenID Connect*, OAuth 2.0, SSO, Authorization, Node.js, Ruby. Repositório GitHub. 2017 Disponıvel em: <https://github.com/panva/> Acesso em 12 de Maio 2017

Stallings, W. Network Security Essentials 4ª Edição. Pearson Education, Inc. 2011

Villarreal, M. E.; Villarreal, S. R.; Westphal, C. M.; Werner, J. Privacy Token: A Mechanism for User's Privacy Specification in Identity Management Systems for the Cloud. ICN 2017 - The Sixteenth International Conference on Networks, 2017 .