# Provedores de Identidade Resilientes e Confiáveis

**Diego Kreutz[1], Eduardo Feitosa[2], Hugo Cunha[2]**

[1]LaSIGE/FCUL, Lisboa, Portugal

[2]IComp/UFAM, Manaus, Brasil

kreutz@computer.org, {efeitosa, hugo.cunha}@icomp.ufam.edu.br

***Resumo.*** *Este artigo apresenta o desenvolvimento de uma arquitetura para provedores OpenID resilientes e seguros, cujo objetivo é garantir propriedades essenciais como integridade, alta disponibilidade e confidencialidade de dados sensitivos. Para este fim são utilizados algoritmos para tolerar falhas arbitrárias, é proposto um componente seguro, com uma interface bem definida, e é detalhada a arquitetura que provê as propriedades almejadas ao sistema. A solução proposta supera trabalhos similares em diferentes aspectos, como vazão, latência, tolerância a falhas arbitrárias e confidencialidade (sem comprometer a escalabilidade do sistema). Os resultados demonstram que uma única instância do sistema suporta demandas de ambientes, como infraestruturas de rede e sistemas Web, com mais de 200k usuários.*

## 1. Introdução e Motivação

Provedores de identidade (IdPs) são empregados como fontes primárias de autenticação e/ou autorização em diferentes tipos de serviços e sistemas. Essencialmente, sistemas Web vem utilizando, em larga escala, provedores de identidade (externos) baseados em OpenID [Recordon and Reed 2006]. Isso significa que o usuário pode ter uma única identidade, em um provedor a sua escolha, para acessar serviços e sistemas de diferentes domínios. Adicionalmente, a utilização de IdPs também vem promovendo a criação de sistemas com suporte a Single-Sign-On (SSO), ou seja, o usuário autentica-se em um único serviço e passa automaticamente a ter acesso a outros serviços, os quais aceitam a mesma sessão de autenticação.

Como consequência, diferentes serviços e sistemas passam a confiar na autenticação e autorização de IdPs. No entanto, apesar da rápida expansão em termos de utilização, ainda persistem diferentes problemas relacionados com a disponibilidade e confiabilidade de provedores de identidade [Uruena et al. 2012, Kreutz et al. 2013b, Kreutz et al. 2013a, Kreutz et al. 2014]. Em termos práticos, a maioria dos IdPs não contempla aspectos de segurança e propriedades como confidencialidade, integridade e disponibilidade.

Existem alguns trabalhos que objetivam melhorar a capacidade de tolerar falhas e intrusões em serviços como Kerberos v5 [de Sousa et al. 2010], RADIUS [Malichevskyy et al. 2012] e provedores OpenID [Barreto et al. 2013], denominado aqui de OpenID-VR. Somente este último prove tolerância a intrusões em provedores OpenID. Entretanto, nenhum desses trabalhos explora como mitigar os diferentes tipos de falhas e ataques ou questões como componentes seguro escaláveis para provedores OpenID. O OpenID-VR tolera apenas intrusões relativas ao protocolo OpenID propriamente dito. Nesse sentido, a proposta aqui apresentada,

denominada de OpenID-PR, detalha uma arquitetura resiliente e confiável para IdPs, capaz de tolerar quaisquer falhas arbitrárias (não apenas do protocolo OpenID), prover propriedades como a confidencialidade de dados sensitivos e atender demandas de ambientes reais.

Os principais objetivos do trabalho são: (1) Identificar, analisar e discutir os principais blocos de construção para melhorar a resiliência e confiabilidade de IdPs OpenID, tomando como base os modelos e componentes anteriormente propostos [Kreutz et al. 2013b, Kreutz et al. 2013a, Kreutz et al. 2013c]; (2) suportar falhas arbitrárias em provedores OpenID, ou seja, não restringir-se ao protocolo OpenID; (3) apresentar e discutir os mecanismos para suportar falhas arbitrárias e intrusões em IdPs; (4) manter a compatibilidade com as infraestruturas OpenID existentes; (5) analisar o desempenho da solução em diferentes cenários, como uma única máquina física, múltiplas máquinas em um único data center e múltiplas máquinas em múltiplos data centers.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta alguns conceitos básicos e contextualiza o trabalho no âmbito do projeto SecFuNet [Kreutz et al. 2013b]. A Seção 3 descreve a arquitetura e os elementos de projeto do OpenID resiliente. Em seguida, as Seções 4 e 5 detalham a implementação e discutem os resultados experimentais da arquitetura proposta. Por fim, a Seção 6 apresenta as considerações finais.

## 2. Contexto e Trabalhos Relacionados

O trabalho insere-se no contexto do projeto SecFuNet, mais especificamente na parte de gestão de identidades através de provedores OpenID. Para assegurar a confidencialidade dos dados sensitivos, num servidor OpenID, são necessários elementos como componentes seguros. Problemas similares tem sido abordados em outros serviços, como Kerberous v5 e RADIUS, bem como com outras abordagens, como "replicação virtual", em provedores OpenID.

**Gestão de Identidade no contexto do SecFuNet**. Um dos pressupostos do projeto SecFuNet é a utilização de componentes seguros no processo de autenticação de usuários e sistemas. Entretanto, a maioria desses componentes, baseados em hardware, possui limitações de escalabilidade. Por exemplo, os smart cards disponíveis no mercado não podem ser comparados com unidades de processamento e armazenamento como CPUs x86 e discos SSD. Consequentemente, um dos desafios é projetar um sistema de identificação e autenticação mais seguro e disponível sem comprometer questões de escala, disponibilidade, integridade e confidencialidade.

A solução proposta para a implantação de IdPs mais confiáveis e resilientes baseia-se no OpenID para o estabelecimento de relações de confiança entre usuários, IdPs e provedores de serviços. Além disso, propõe o uso de componentes seguros, especificamente projetados para o OpenID, para isolar e proteger a confidencialidade de dados sensitivos, como chaves privadas e asserções de autenticação.

**Componentes Seguros**. Componentes seguros podem ser implementados utilizando diferentes técnicas e tecnologias como o isolamento provido por *hypervisors* (e.g. máquinas virtuais isoladas, componentes de software no domínio do *hypervisor*), Trusted Platform Modules (TPMs), arquiteturas como Secure Virtual Machine (SVM) da AMD, tecnologias como Trusted Execution Technology (TXT) da Intel,

e soluções baseadas em FPGAs [Niedermayer et al. 2014, Viejo 2014].

A solução proposta considera duas formas de prover componentes seguros: (1) através do isolamento provido por *hypervisors* ; e (2) por meio de conceitos e premissas de TPMs, que pode ser aplicado a sistemas de autenticação para garantir a confidencialidade de dados sensitivos, sem comprometer a escalabilidade do sistema.

**Trabalhos Relacionados**. Até o presente momento, a tolerância a falhas e intrusões em IdPs tem sido pouco explorada na literatura. Os três trabalhos descritos a seguir são confrontados e comparados com a presente proposta.

Os dois primeiros exploram técnicas de tornar os serviços Kerberos v5 (denominado de Typhon) [de Sousa et al. 2010] e RADIUS [Malichevskyy et al. 2012] mais resilientes. Ambos utilizam técnicas e algoritmos tradicionais de replicação de máquinas de estado para tolerar falhas arbitrárias nos respectivos serviços. Adicionalmente, propõe componentes seguros, pouco escaláveis (se aplicados a componentes de hardware), para armazenar os dados sensíveis, como chaves secretas e tokens de sessão, e garantir a confidencialidade do sistema em caso de intrusão nas réplicas.

O terceiro trabalho propõe um IdP capaz de tolerar intrusões de operação do protocolo OpenID [Barreto et al. 2013]. O OpenID-VR utiliza uma abordagem baseada em "replicação virtual". Em outras palavras, o sistema assume que o *hypervisor* forma uma base de computação confiável. Sendo assim, as réplicas (máquinas virtuais sobre um mesmo *hypervisor*) utilizam uma área de memória compartilhada para executar protocolos de consenso, por exemplo. O pressuposto é que essa estratégia reduz a sobrecarga de comunicação imposta por abordagens baseadas em troca de mensagens (comunicação via rede), como é o caso do Typhon e do RADIUS resilientes. Além disso, o OpenID-VR utiliza um único serviço de autenticação, baseado em processadores seguros (grids de smart cards) para a autenticação dos usuários.

Em resumo, a abordagem do OpenID-VR possui pelo menos dois desafios: (1) a escalabilidade do sistema, uma vez que grids de smart cards geram uma sobrecarga significativa ao processo de autenticação, que pode ultrapassar a 2 segundos [Urien and Dandjinou 2006], além de impactar significativamente na vazão do sistema; (2) tolerar apenas instruções relativas ao protocolo OpenID, ou seja, falhas de infraestruturas (físicas e/ou lógicas) não são suportadas. Além disso, o serviço de autenticação é um ponto único de falha.

A Tabela 1 resume as principais características e componentes dos três sistemas, bem como da solução proposta. Nota-se que a solução OpenID-VR, por basear-se em máquinas virtuais e uma única máquina física, não é capaz de tolerar falhas físicas (e.g. quedas de energia, defeitos em discos rígidos, perdas de conectividade, entre outros), falhas lógicas (e.g. falhas de software, falhas de configuração de rede e/ou conectividade, entre outros) e nem ataques de exaustão de recursos, como os reportados em [Kreutz et al. 2013b]. Adicionalmente, uma falha do *hypervisor* ou do serviço de autenticação seguro compromete toda a operação do serviço OpenID, uma vez que ambos são únicos. Por outro lado, a solução proposta permite utilizar de 1 a $3f + 1$ componentes seguros (*# c. seguros*). Além disso, propõe um componente seguro para o desenvolvimento de subsistemas de autenticação escaláveis (*C. s. esc.*). Finalmente, a vazão do sistema mostra-se superior às demais soluções.

Resumidamente, os principais diferenciais da solução proposta, quando com-

**Tabela 1. Comparação dos trabalhos relacionados e abordagem proposta**

| Sistema | Falhas fís. | Falhas lóg. | Tipos de ataques | # réplicas | # c. seguros | C. s. esc. | Vaz. | Lat. |
|---|---|---|---|---|---|---|---|---|
| Typhon | sim | sim | quaisquer(*) | $3f+1$ | $3f+1$ | não | 280 | 5.7ms |
| RADIUS | sim | sim | quaisquer(*) | $3f+1$ | $3f+1$ | não | 100 | 200ms |
| OpenID-VR | não | sim/não | OpenID | $2f+1$ | 1 | não | — | 7ms |
| OpenID-PR | **sim** | **sim** | **quaisquer(*** | $3f+1$ | 1 a $3f+1$ | **sim** | **2440** | **7ms** |

(*) Estes sistemas toleram potencialmente qualquer tipo de falha ou ataque em até $f$ réplicas do sistema.

parada a solução OpenID-VR, são: (1) maior disponibilidade, resistindo a diferentes tipos de falhas físicas e lógicas; (2) resistência a ataques de exaustão de recursos, uma vez que as réplicas podem ser instanciadas em diferentes máquinas físicas e/ou domínios administrativos; (3) potencial suporte a quaisquer tipos de ataques e não apenas a ataques ao protocolo OpenID; (4) maior robustez no subsistema de autenticação, uma vez que pode ser composto por 1 a $3f + 1$ (até um por réplica) componentes seguros; (5) latência de autenticação equiparável, apesar de utilizar protocolos de troca de mensagens ao invés de memória compartilhada; (6) suporte a componentes seguros do tipo TPM, permitindo escalabilidade do subsistema de autenticação ao armazenar os dados sensíveis (e.g. chaves de sessão, dados de usuários, etc.) nas réplicas de forma protegida (i.e., cifrada); e (7) capacidade de tirar vantagem de ambientes multi-data center e/ou multi-cloud, usufruindo dos respectivos mecanismos de defesa dessas infraestruturas, como mecanismos de mitigação de ataques de negação de serviços [Prince 2013, Prince 2012].

## 3. Arquitetura do Sistema

A arquitetura do sistema proposto recorre a conceitos, técnicas e mecanismos que permitem construir sistemas baseados no conceito de segurança automática [Verissimo et al. 2006]. Em outras palavras, não corrige todos os problemas de segurança de IdPs, mas sim utiliza técnicas para tolerar falhas arbitrárias e intrusões.

### 3.1. Visão geral da arquitetura

As configurações arquiteturais ilustradas nas Figuras 1 e 2 são baseadas (e representam uma extensão) em um modelo funcional para serviços de autenticação e autorização, anteriormente proposto [Kreutz et al. 2013a]. Essencialmente, são cinco elementos principais: (a) cliente; (b) serviço e *relying party*; (c) gateway OpenID; (d) réplicas OpenID; e (e) componentes seguros. O conjunto dos gateways e réplicas OpenID, juntamente com o(s) componente(s) seguro(s), formam a infraestrutura do IdP. A função essencial do(s) componente(s) seguro(s) é garantir a confidencialidade dos dados sensitivos, como chaves privadas, certificados, chaves de sessão e asserções OpenID. Além disso, todas as operações que requerem acesso aos dados sensitivos são executadas pelo(s) componente(s) seguro(s).

O cliente representa um usuário ou dispositivo (e.g. máquina virtual) que quer acessar um serviço. Este serviço (que permite autenticação via OpenID) é vinculado a uma *relying party*, a qual redireciona o cliente para o seu próprio IdP.

O gateway é um elemento transparente para o usuário final e *relying party*, pois ele representa o servidor OpenID. As requisições que chegam ao gateway são simplesmente repassadas às réplicas OpenID. A função do gateway é mascarar a

replicação ativa, i.e., um pedido de um cliente é enviado para $3f_R + 1$ réplicas do sistema. O gateway mantém a compatibilidade com sistemas OpenID 2.0 existentes.

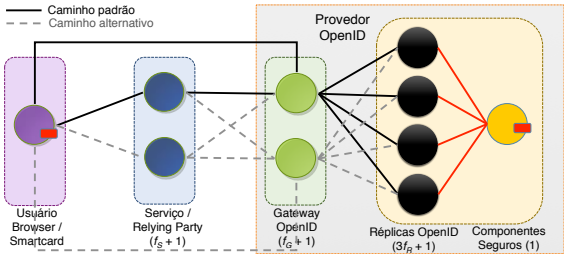


**Figura 1.** 1 **componente seguro**

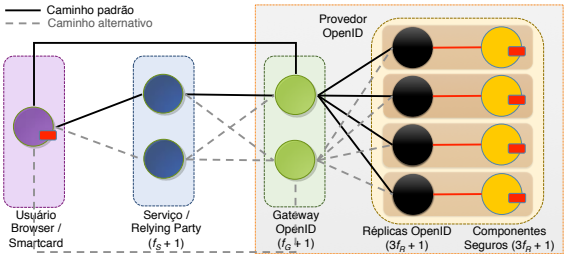


**Figura 2.** $3f_R + 1$ **comp. seguros**

O IdP propriamente dito é considerado a parte crítica do modelo funcional, pois é ele quem identifica e autentica usuários para os serviços. Portanto, requer níveis de segurança e disponibilidade mais elevados para assegurar que usuários inválidos e/ou sem permissão não sejam aceitos nos serviços. Um provedor de identidade pode ser parte integrante de um domínio local, bem como um serviço prestado por terceiros. Na prática, o caso mais comum tem sido os diferentes serviços aceitarem autenticações de usuários de IdPs externos. Devido às consequências que podem ser geradas por eventuais IdPs falhos ou comprometidos, é assumido que IdPs OpenID devem tolerar falhas arbitrárias, que podem ser causadas por comportamentos inesperados da infraestrutura ou do sistema, bem como por ataques. Adicionalmente, o IdP OpenID resiliente deve tolerar também intrusões. Portanto, são necessários componentes seguros para resguardar a confidencialidade de dados sensitivos.

Outro aspecto a observar nas Figuras 1 e 2 é o número de componentes seguros. No primeiro caso, as réplicas OpenID utilizam um único componente seguro, de forma compartilhada. Em termos práticos, o componente seguro irá responder à réplica solicitante somente após receber o mesmo pedido de $3f_R + 1 - f_R$ réplicas. Portanto, mesmo com um único componente seguro, não basta a um atacante comprometer apenas $f_R$ réplicas para conseguir fazer com que o componente seguro gere pedidos válidos a partir de dados de entrada forjados ou modificados. Este primeiro caso é similar a solução OpenID-VP, ou seja, as réplicas e o componente seguro podem estar a executar sobre uma mesma plataforma de hardware e *hypervisor*, por razões de desempenho (menor latência de comunicação e potencial maior vazão).

Por outro lado, a configuração apresentada na Figura 2 visa, além de manter a integridade e a confidencialidade, também a alta disponibilidade do sistema. Esta configuração é menos suscetível a ataques de exaustão de recursos, falhas físicas e lógicas, além de outros tipos de ataques, como DDoS de grandes proporções. Isso deve-se ao fato de esta configuração do sistema poder tirar proveito dos mecanismos de segurança e proteção de diferentes máquinas físicas e/ou ambientes administrativos. Além disso, essa segunda configuração pode ainda ser utilizada para aumentar o desempenho do sistema, em especial a capacidade de processar requisições de autenticação por unidade de tempo.

É importante ressaltar ainda que a pilha de protocolos não muda. As comunicações com o serviço OpenID continuam a ser via HTTP e/ou SSL (ou TLS para autenticação mútua). O que ocorre é que o gateway recebe as requisições

SSL/HTTP e as encapsula no protocolo de tolerância a falhas Bizantinas (BFT). Consequentemente, um IdP OpenID resiliente pode facilmente, e de forma transparente, substituir um IdP OpenID tradicional. O cliente, ou a *relying party*, não vai notar nenhuma diferença funcional ou operacional no sistema.

### 3.2. Tipos de falhas toleradas

A Figura 3 ilustra os mecanismos de detecção e/ou mascaramento de falhas entre os diferentes elementos. Para mitigar falhas, além de suportar falhas por parada, há mecanismos complementares de detecção de falhas entre o cliente, *relying party* e gateway. Exemplos incluem mensagens corrompidas e pacotes mal formados.

Entre o gateway e as réplicas existem mecanismos de replicação ativa, utilizando máquina de estados replicada, para mascarar falhas arbitrárias de até $f_R$ réplicas. Em outras palavras, qualquer tipo de comportamento, como atrasos, respostas mal formadas e mensagens corrompidas, em até $f_R$ réplicas, irá ser mascarado sem afetar a operação do sistema. Por exemplo, pacotes mal formados ou respostas divergentes, de uma réplica falha, serão simplesmente descartadas.

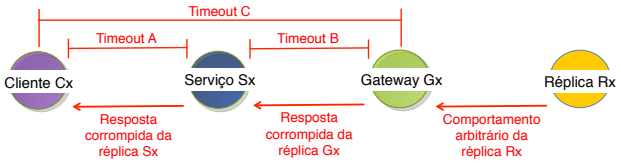


**Figura 3. Detecção e mascaramento de falhas entre os elementos**

### 3.3. Componentes seguros para provedores de identidade

O componente seguro deve ser mínimo e prover métodos como os discriminados na Tabela 2. Neste caso, as informações de associação e os *handlers* são armazenados no sistema de arquivos da réplica, contudo, cifrados e/ou assinados para assegurar a confidencialidade e integridade dos dados. Portanto, o componente seguro segue o modelo de comportamento de uma TPM, armazenando internamente informações mínimas necessárias para garantir algumas das propriedades do sistema.

**Tabela 2. Interface do componente seguro**

| Método | Protocolos | Entrada | Saída |
|---|---|---|---|
| `VerifySignRSA` | TLS/SSL | Pacote com a assinatura a ser verificada. | Ack ou Nack de verificação da assinatura. |
| `SignRSA` | TLS/SSL | Dados a assinar. | Assinatura RSA para os dados de entrada, utilizando a chave ($Pr_S$). |
| `GenAssociation` | OpenID | Dois números inteiros e chave pública do cliente. | Informação de associação e a chave pública do serviço OpenID. |
| `GenerateNonce` | OpenID | — | Um *nonce* pseudo aleatório. |
| `SymmetricCipher` | TLS/SSL | Dados a cifrar. | Dados de entrada cifrados. |

O componente seguro necessita armazenar somente quatro valores ($K_{User}$, $Pr_S$, $Pu_{CA}$ e $K_{Assoc}$) para garantir as propriedades de integridade e confidencialidade do sub sistema de autenticação. Utilizando a chave $K_{Assoc}$ e um *cipher* simétrico (`SymmetricCipher`), as informações de associação podem ser armazenadas, cifradas, na réplica sem comprometer a confidencialidade dos dados no caso de uma intrusão. Sem conhecer a chave secreta ou quebrar o algoritmo de cifra, um atacante não é capaz de ler as informações de associação do provedor de identidade.

O segredo $K_{User}$ é utilizado para verificar a identidade e os dados do usuário. Cada entrada na tabela de usuários do sistema contém um MAC gerado com o segredo $K_{User}$. Portanto, um atacante não consegue gerar ou modificar entradas da tabela sem conhecer a chave secreta utilizada pelo componente seguro. Em outras palavras, apenas o componente seguro consegue gerar MACs válidos para as entradas da tabela de usuários. Alternativamente, um administrador do sistema também poderia conhecer a chave secreta para poder gerenciar (adicionar, modificar, remover) a tabela de usuários do sistema.

Já o certificado $Pu_{CA}$ é necessário para verificar a identidade de um usuário através do respectivo certificado assinado pela CA. O sistema assume que usuários válidos são somente aqueles que possuem um certificado válido e assinado com a chave privada da CA. Portanto, a chave pública permite ao componente verificar a validade da identidade de um usuário.

A chave privada do serviço ($Pr_S$) é necessária para o caso de autenticação mútua, fim-a-fim, entre o cliente e o serviço OpenID, através de protocolos como o EAP/TLS. Esta chave é utilizada para verificar a assinatura (`VerifySignRSA`) da mensagem enviada pelo cliente e/ou *relying party*.

Finalmente, o método `GenerateNonce` tem como função criar um carimbo temporal e um número pseudo aleatório. O *nonce* tem como finalidade evitar ataques de *replay* junto às respostas de autenticação do OpenID.

## 4. Implementação

Para a implementação do serviço OpenID resiliente foi utilizada a biblioteca *openid4java* [OpenID4Java 2013] (versão 0.9.8), que suporta as versões 1.0 e 2.0 do OpenID. O protótipo desenvolvido utiliza a especificação 2.0 do OpenID.

A replicação ativa é realizada através da biblioteca BFT-SMaRt [Bessani et al. 2013]. Esta biblioteca prove um conjunto de módulos e protocolos de comunicação como o Mod-SMaRt, o VP-Consensus e Canais Confiáveis para comunicação entre as réplicas. Tanto Mod-SMaRt quanto VP-Consensus utilizam canais confiáveis em sua comunicação entre réplicas e componentes do sistema.

O componente seguro, que é um elemento independente do sistema, foi implementado como um módulo Java, com uma interface bem definida. Ele é utilizado pelas réplicas para executar algumas operações essenciais do protocolo, como detalhado na Figura 4. Adicionalmente, quando há um componente seguro por réplica, o Mod-SMaRt é utilizado também para comunicação entre os componentes seguros. Essa comunicação é necessária para chegar a um consenso entre os componentes seguros antes de responder à réplica. Os componentes seguros podem ser executados no mesmo ambiente das réplicas ou em máquinas virtuais isoladas.

**Componentes do sistema**. Tanto o cliente quanto a *relying party* podem ser aplicações (e.g. *browser*) ou serviços existentes. Para fins de avaliação experimental do protótipo, foram desenvolvidos um cliente e uma *relying party* em Java.

**OpenID Gateway**. É um elemento projetado para manter compatibilidade com infraestruturas OpenID existentes. Assim, recebe conexões TCP/IP (com HTTP e/ou HTTP sobre SSL) dos clientes e *relying parties* OpenID e simplesmente repassa os pacotes recebidos a interface BFT proxy do BFT-SMaRt. A biblioteca encarrega-se de enviar uma cópia dos pacotes para todas as réplicas. De maneira similar,

pacotes vindos das réplicas são enviados aos clientes e/ou *relying parties* através dos canais TCP/IP previamente estabelecidos.

**Réplicas OpenID**. A implementação atual do protótipo suporta OpenID 2.0 sobre HTTP. Uma réplica consiste em uma implementação do OpenID 2.0 a executar com o suporte da biblioteca BFT-SMaRt, que fornece os protocolos para replicação de máquina de estados. Com isso, todas as réplicas OpenID, corretas, atendem todas as requisições de autenticação na mesma ordem. Fazem parte dos trabalhos futuros o suporte a HTTPS e EAP-TLS para autenticação mútua entre: (a) cliente e *relying party*; (b) *relying party* e serviço OpenID; e (c) cliente e serviço OpenID. Vale a pena ressaltar que a implementação desse protocolos pode tirar partido da implementação EAP-TLS disponível no componente seguro do RADIUS [Malichevskyy et al. 2012].

**Componente seguro**. O componente seguro foi implementado em Java com a API de criptografia BouncyCastle [of the Bouncy Castle Inc. 2014]. O principal desafio de implementação foi o determinismo requerido pelas réplicas. Em outras palavras, os componentes seguros de cada réplica devem gerar dados de saída de maneira determinística. Para resolver esse problema foi utilizada a mesma solução da implementação do RADIUS resiliente, ou seja, uma adaptação da função de geração de números pseudo-aleratórios (PRF) do TLS [Malichevskyy et al. 2012].

### 4.1. Interação entre os elementos do sistema

A Figura 4 ilustra os passos de comunicação entre os diferentes elementos do OpenID resiliente. No passo 1, o usuário requisita acesso a um serviço através da *relying party*. No passo 2, a *relying party* apresenta ao usuário um formulário para ele inserir o seu identificador específico (URL de identificação, no passo 3). No passo 4, a *relying party* realiza a operação de descoberta em busca da informação necessária para continuar o processo de autenticação. A requisição de descoberta é encaminhada para o serviço OpenID replicado.

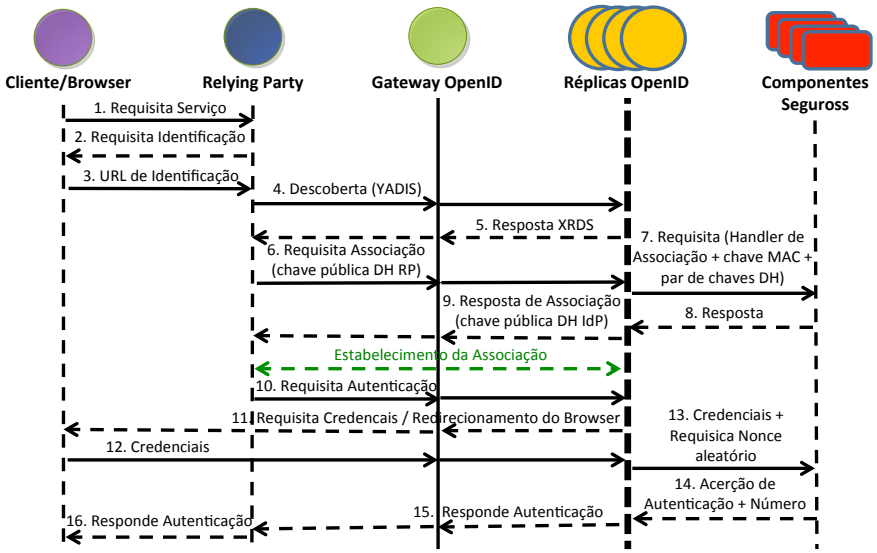


**Figura 4. Visão geral das interações entre os componentes do sistema (OpenID 2.0)**

O serviço OpenID responde à *relying party* (passo 5) com um documento XRDS que contém, entre outras coisas, uma lista de URLs (ponteiros para o provedor

OpenID) com as respectivas prioridades. Esta lista pode ser utilizada pela *relying party* nos próximos passos.

A seguir, a *relying party* tenta conectar-se com cada servidor OpenID (URL) contida na lista, de acordo com a ordem de prioridade pré-definida, e estabelecer uma associação protegida. A primeira conexão bem sucedida é utilizada para requisitar a associação (passo 6). Esta requisição contém dados como a respectiva URL e os dados Diffie-Hellman (DH) [Rescorla 1999] (chave pública, número primo, número gerador) da *relying party* para manter a associação segura.

No servidor OpenID, a requisição de associação é encaminhada ao componente seguro, que gerará o *handler* de associação e a chave MAC (passo 7). Na sequência, o servidor OpenID solicita ao componente seguro uma par de chaves DH, enviando os dados de entrada recebidos da *relying party* (número primo e número gerador). Depois disso, o componente seguro gera e retorna (passo 8) a informação requisitada (*handler* de associação, chave MAC e par de chaves DH do OpenID). A resposta de associação, contendo a chave pública DH do OpenID (entre outras informações), é enviada para a *relying party* (passo 9), completando a associação.

Assim que a associação entre a *relying party* e o servidor OpenID é estabelecida, a autenticação é iniciada. No passo 10, a *relying party* envia uma requisição de autenticação ao servidor OpenID, que é encaminhada ao browser do cliente como um pedido de credenciais (passo 11). O cliente envia as suas credenciais ao servidor OpenID (passo 12), que em seguida requisita a verificação das credenciais ao componente seguro. Adicionalmente, ele requisita também um *nonce* aleatório (passo 13). O componente seguro responde com a asserção de autenticação e o *nonce* gerado (passo 14). Depois disso, a resposta de autenticação é enviada a *relying party* (passo 15), que realiza as ações necessárias. Finalmente, a *relying party* envia uma resposta de autenticação ao cliente, completando o processo de autenticação (passo 16).

### 4.2. Configuração de elementos do sistema

O protótipo desenvolvido suporta duas configurações distintas, similares as apresentadas nas Figuras 1 e 2. Na primeira existe um único componente seguro, utilizado por todas as réplicas. Tanto as réplicas quanto o componente seguro podem estar a executar sobre uma mesma máquina física, em diferentes máquinas virtuais, por razões de desempenho. Neste caso, o componente seguro pode realizar uma votação majoritária simples, a partir dos pedidos das réplicas, sem precisar estabelecer comunicação com componentes seguros replicados.

A segunda configuração oferece uma plataforma mais robusta. Neste caso, há um componente seguro por réplica. Portanto, as réplicas podem ser instanciadas em uma única máquina física, em múltiplas máquinas físicas ou ainda em domínio distintos (e.g. múltiplos data centers). Nessa configuração até $f_R$ componentes seguros podem falhar sem comprometer a operação do sistema.

Além disso, como o gateway e o OpenID replicado foram implementados em Java, ambos podem ser instanciados em diferentes sistemas operacionais, o que aumenta a robustez do sistema pelo fato de evitar vulnerabilidades comuns existentes num mesmo sistema. Outra característica interessante desta configuração é com relação à independência de *hypervisors*. Como o sistema baseia-se no paradigma de troca de mensagens, diferentes máquinas virtuais (réplicas) podem ser instanciadas

em distintos *hypervisors*, aumentando a robustez contra vulnerabilidades comuns tanto no sistema operacional quanto no sistema de virtualização.

## 5. Avaliação e Resultados Experimentais

Esta seção tem por objetivo discutir e avaliar a solução proposta em relação a diferentes ataques e tipos de falhas. Além disso, também são apresentadas avaliações de desempenho em três cenários distintos.

### 5.1. Ataques ao OpenID

O framework OpenID possui diferentes problemas de segurança, tanto de especificação quanto de implementação [Uruena et al. 2012, Kreutz et al. 2013b, Kreutz et al. 2013a]. A Tabela 3 resume algumas dos ataques conhecidos, bem como aponta as soluções adotadas no protótipo desenvolvido, como forma de mitigar os ataques.

**Tabela 3. Análise dos Ataques ao OpenID**

| Ataque | Problema | Soluções |
|---|---|---|
| *Man-in-the-middle* | Interceptação de comunicações. | Segredo compartilhado entre RP e IdP, usando Diffie-Hellman, e protocolos como TLS para autenticação mútua entre usuário e servidor OpenID. |
| *Negação de serviço* | Potenciais ataques de DoS em RPs e IdPs. | (1) Limitar o número de requisições/s nos gateways (e.g. `iptables`). (2) No IdP, negar requisições com base nos valores das mensagens *openid.realm* e *openid.return_to*. (3) Utilizar mecanismos baseados em timeouts, verificação de mensagens corrompidas e pacotes mal formados. |
| *Replay* | No OpenID 2.0 é opcional a assinatura do *nonce*. | (1) Incluir o *nonce* na lista de informações assinadas. (2) Rejeitar *nonces* expirados. |

### 5.2. Falhas arbitrárias e falhas por parada

O OpenID resiliente é comparado com uma versão livremente disponível e utilizada em IdPs OpenID, o JOIDS [openid-server Community 2010]. Este representa o caso comum de implementações OpenID disponíveis para criação de IdPs.

**Falha por parada**. Para provocar uma falha por parada é o suficiente matar o respectivo processo no sistema operacional. O OpenID resiliente não é afetado, i.e., continua a funcionar normalmente, sem perdas de desempenho, com falhas por parada de até $f_R$ réplicas. Similarmente, falhas em até $f_G$ gateways não afetam a operação do sistema, uma vez que as *relying parties* irão tentar estabelecer conexão com todos os gateways da lista. Por outro lado, em um sistema baseado no JOIDS, uma falha por parada interrompe o serviço.

**Falhas arbitrárias**. Servidores OpenID com o JOIDS não toleram falhas arbitrárias (e.g. bugs em diferentes camadas de software, problemas de configuração dos sistemas, atraso de mensagens intencionalmente provocadas por um atacante, e assim por diante). Como resultado, o serviço pode negar a autenticação de usuários legítimos e garantir asserções de associação a usuários não autorizados, por exemplo. Por outro lado, o OpenID resiliente tolera até $f_R$ falhas arbitrárias nas réplicas OpenID. Isso incluí falhas de sistemas operacionais, falhas de bibliotecas e componentes de software, entre outras falhas físicas e lógicas, além de intrusões.

### 5.3. Impacto de falhas periódicas e ataques

**Parando periodicamente até $f_R$ réplicas**. Para avaliar o sistema sobre circunstâncias adversárias, como falhas periódicas por parada, foi utilizado o ambiente

UFAM-VMs (detalhado na seção seguinte) com 20 e 40 clientes. Foi implementado um *script* para matar e re-iniciar uma das réplicas a cada 10 segundos durante a execução do sistema. Com periodicamente uma réplica a menos no sistema, houve um ligeiro aumento no desempenho do sistema, passando de 867.73 para 1009.86 autenticações/s com 20 clientes e de 984.59 para 1145.98 com 40 clientes. Isso é devido ao fato de menos réplicas no sistema ($3f_R + 1 - f_R$) gerar uma menor sobrecarga de comunicação nos protocolos utilizados para replicação de máquina de estados.

**Ataque constante de DoS em até $f_R$ réplicas**. Foi utilizado o comando `hping3` para gerar um ataque de DoS constante, utilizando as flags SYN e ACK do TCP, na porta de comunicação de uma das réplicas. Esta réplica começou a ficar mais lenta e não receber todas as mensagens do gateway por causa do ataque. Consequentemente, as outras réplicas consideraram a réplica como comprometida e continuaram a manter o sistema em correta operação. Novamente, pode ser observado um pequeno aumento no desempenho do sistema, indo de 867.73 a 956.46 autenticações/s com 20 clientes, e de 984.59 para 1005.54 com 40 clientes.

### 5.4. Análise de Desempenho

Foram utilizados três ambientes distintos para avaliação do protótipo, conforme resumido na Tabela 4. Em cada ambiente foram instanciadas 5 máquinas virtuais (MVs), sendo uma para executar um gateway e os clientes e outras 4 para executar as réplicas e componentes seguros. No primeiro ambiente, UFAM-VMs, as máquinas virtuais (Debian Linux) foram suportadas pelo *hypervisor* Xen. O segundo ambiente, Amazon-EC2, é representado por nós de computação `m3.xlarge` [Amazon Web Services, Inc. 2014] no datacenter da Amazon de N. Virginia, rodando Ubuntu Server 13.10. O terceiro ambiente também utilizou nós de computação `m3.xlarge`, entretanto, distribuídos em três data centers da Amazon, N. Virginia (Ubuntu Server 13.10), N. Califórnia (Amazon Linux AMI ) e Oregon (Ubuntu Server 12.04 LTS).

**Tabela 4. Ambientes e configuração das máquinas virtuais**

| Environment | vCPUs | ECUs | MEM | Disk | Network |
|---|---|---|---|---|---|
| UFAM-VMs | 2 | — | 2GB | 20GB | Gigabit Ethernet |
| Amazon-EC2 | 4 | 13 | 15GB | 2x40GB SSD | High Speed Gigabit |
| Amazon-DCs | 4 | 13 | 15GB | 2x40GB SSD | WAN pública |

**Vazão do sistema**. A Tabela 5 resume os resultados para os três ambientes de teste. Foram utilizados 20, 40, 80 e 100 clientes simultâneos, executados na mesma máquina do gateway. Cada cliente foi configurado para realizar 2.000 requisições de autenticação. Uma autenticação é composta por 5 mensagens, de acordo com a especificação do OpenID 2.0, o que totaliza 10.000 mensagens por cliente. Devido a limitações de tempo e alocação de recursos, a média de autenticações por segundo foi calculada a partir de 5 execuções para cada configuração de cliente (e.g. 20). Apesar de representar um número relativamente baixo de execuções para cada configuração, durante os testes e experimentos foi observado um comportamento razoavelmente estável, sem variações significativas entre uma e outra execução.

O número de autenticações por segundo varia de 860 (20 clientes) a 995 (80 clientes) no ambiente UFAM-VMs. Por outro lado, com 100 clientes há uma queda

**Tabela 5. Número de autenticações por segundo com 20, 40, 80 e 100 clientes**

| Ambiente | 20 clientes | 40 clientes | 80 clientes | 100 clientes |
|---|---|---|---|---|
| UFAM-VMs | 867.73 | 984.59 | 995.12 | 960.11 |
| Amazon-EC2 | 1969.17 | 2166.58 | 2244.30 | 2244.04 |
| Amazon-DCs | 26.66 | 50.72 | 92.42 | 114.05 |

no desempenho. Ela deve-se às limitações do ambiente, que entra num estado de *thrashing* (i.e. sobrecarga de escalonamento, acesso concorrente de I/O, etc.) com um número elevado de clientes. Um comportamento similar é observado no ambiente Amazon-EC2. Portanto, para executar mais clientes seriam necessárias mais VMs, como foi analisado e constatado na prática.

É interessante observar que ambos os ambientes (UFAM-VMs e Amazon-EC2) são similares, exceto pelo fato das VMs do Amazon-EC2 terem um poder de computação superior às do UFAM-VMs. Portanto, essa disparidade justifica a diferença de desempenho. A exemplo, se pegarmos o caso de 80 clientes, o Amazon-EC2 suporta uma carga de autenticações/s 2.25 vezes maior que o UFAM-VMs.

Diferentemente dos dois primeiros ambientes, o Amazon-DCs apresenta um desempenho significativamente menor. A principal razão para esse desempenho está na latência da rede (WAN) entre os datacenters, como discutido a seguir. Com 20 clientes chega-se a uma vazão de 26.66 autenticações/s, enquanto que com 100 clientes é possível atingir-se uma média de 114.05 autenticações/s.

**Latência do ambiente e sistema**. A latência da rede foi medida utilizando o comando `ping` para gerar 100 pacotes de 512 bytes. Portanto, a latência representa a média de 100 mensagens ICMP. O tamanho de 512 bytes deve-se ao fato de a maioria das mensagens de autenticação do OpenID girar em torno desse tamanho.

A latência entre datacenters, no ambiente Amazon DCs, varia significativamente. Ela vai de 32.10ms entre Oregon e N. California até 87.34ms entre Oregon and N. Virginia, o que representa uma diferença de 2.72 vezes. Portanto, uma das coisas a observar, na escolha dos datacenters, é a latência da rede. Inter-conexões privadas e/ou de menor latência irão gerar ganhos significativos no desempenho geral do sistema. Por outro lado, as latências de rede dos ambientes UFAM-VMs e Amazon-EC2 ficaram em uma média de aproximadamente 0.075ms e 0.185ms, respectivamente. Como observado, a latência entre os datacenters é superior às demais, o que justifica os números de vazão do sistema no ambiente multi-datacenter.

Adicionalmente, a média de latência das autenticações (média de 9.000) ficou em 7.16ms (UFAM-VMs), 6.22ms (Amazon-EC2) e 857.93ms (Amazon-DCs). Como esperado, a latência de autenticação no ambiente Amazon-DCs é significativamente alta. Comparando os ambientes UFAM-VMs e Amazon-EC2, novamente tem-se um desempenho melhor do segundo em relação ao primeiro. Entretanto, observa-se que a diferença é muito menor quando equiparada a diferença de vazão do sistema, o que significa que o maior impacto na vazão do sistema está mais relacionado com a capacidade de computação do que com latência de uma autenticação.

**Capacidade e dimensionamento do sistema**. Coletando os dados de um ambiente universitário real, com aproximadamente 11.5k usuários, pôde ser constatado que: (1) o número máximo de autenticações, na hora mais crítica num período de 7

dias, chega a 143907 autenticações, o que representa uma média de 39.97 autenticações por segundo; e (2) o maior pico de autenticações, somando o pior segundo dos dois sistemas de autenticação, o Active Directory e o LDAP, chega a 118 autenticações por segundo. Entretanto, mais de 99.99% do tempo o número de autenticações por segundo fica abaixo de 100. Cabe ressaltar ainda que esses números representam as demandas de autenticação de praticamente todos os sistemas da instituição, incluindo login em computadores de laboratórios ou de uso pessoal, logins de acesso à rede (e.g. autenticações 802.1x), autenticações em sistemas Web online, verificações de usuários nos servidores SMTP, entre outros.

A partir dos números apresentados, pode ser inferido que a solução desenvolvida é capaz de suportar a demanda de ambientes com algumas dezenas de milhares de usuários. Considerando o pior caso, de 118 autenticações por segundo, o ambiente UFAM-VMs comporta aproximadamente 97k usuários. Já o ambiente Amazon-EC2 comporta uma instituição, ou sistema, com mais de 218k usuários. Por outro lado, o ambiente Amazon-DCs comporta uma infraestrutura com apenas cerca de 11k usuários. Mesmo assim, considerando a robustez provida pelo ambiente, os números são interessantes, mostrando que a solução é aplicável a ambientes reais. Cabe ressaltar que esses números representam uma única instância do sistema. Com múltiplas instâncias, e distribuição de carga entre elas, a capacidade do sistema deve potencialmente aumentar de forma linear.

## 6. Conclusão

O artigo detalhou uma arquitetura para o desenvolvimento de provedores de identidade mais resilientes e seguros. Entre os componentes e técnicas fundamentais estão protocolos para tolerar falhas arbitrárias, diversidade de sistemas operacionais e *hypervisors*, componentes seguros para garantir a confidencialidade de dados sensitivos e um novo elemento, denominado de gateway, para manter compatibilidade com infra-estruturas OpenID existentes.

Um protótipo foi implementado, como prova de conceito, seguindo a especificação do OpenID 2.0. A base de desenvolvimento foi a linguagem Java e as bibliotecas *openid4java*, BFT-SMaRt e BouncyCastle.

No que refere-se aos resultados, são discutidos os mecanismos e recursos adotados para evitar e/ou mitigar diferentes tipos de ataques, bem como tolerar falhas arbitrárias no serviço OpenID. Adicionalmente, a avaliação de desempenho do sistema foi baseada em três ambientes distintos, sendo um deles multi-data center. Os resultados de vazão e latência demonstram que o sistema é capaz de atender demandas de ambientes reais, como instituições de ensino, com mais de 200k usuários.

## Agradecimentos

Agradecemos aos revisores anônimos pelos comentários. Este trabalho é suportado pela FCT, através do Programa Multianual (LaSIGE) e projeto TRONE (CMU-PT/RNQ/0015/2009), pela Comissão Européia, através do projeto SecFuNet (FP7-ICT-STREP-288349), e pelo CNPq, através dos processos de número 590047/2011-6 e 202104/2012-5.

## Referências

Amazon Web Services, Inc. (2014). Amazon EC2 Pricing. `http://aws.amazon.com/ec2/pricing/`.

Barreto, L., Siqueira, F., da Silva Fraga, J., and Feitosa, E. (2013). Gerenciamento de Identidades Tolerante a Intrusões. In *Anais do XXXI SBRC*. SBC.

Bessani, A., ao Sousa, J., and Alchieri, E. (2013). State Machine Replication for the Masses with BFT-SMaRt. Technical report, DI/FCUL. `goo.gl/XPEHMm`.

de Sousa, J. C., Bessani, A., and Sousa, P. (2010). Typhon: um serviço de autenticação e autorização tolerante a intrusões. In *INForum*. `http://goo.gl/mgPT7Y`.

Kreutz, D., Feitosa, E., Malichevskyy, O., Barbosa, K. R. S., and Cunha, H. (2013a). Um modelo funcional para serviços de identificação e autenticação tolerantes a faltas e intrusões. In *SBSeg*. SBC. `goo.gl/yDet4k`.

Kreutz, D., Malichevskyy, O., Feitosa, E., Barbosa, K. R. S., and Cunha, H. (2014). System design artifacts for resilient identification and authentication infrastructures. In *Proceedings of the Tenth International Conference on Networking and Services*. To appear.

Kreutz, D., Niedermayer, H., Feitosa, E., da Silva Fraga, J., and Malichevskyy, O. (2013b). Architecture components for resilient networks. Technical report, SecFuNet Consortium. `goo.gl/xBHCNb`.

Kreutz, D., Ramos, F. M., and Verissimo, P. (2013c). Towards secure and dependable software-defined networks. In *SIGCOMM HotSDN*, pages 55–60.

Malichevskyy, O., Kreutz, D., Pasin, M., and Bessani, A. (2012). O vigia dos vigias: um serviço RADIUS resiliente. In *INForum*. `http://goo.gl/GDUYBw`.

Niedermayer, H., Kreutz, D., Feitosa, E., Malichevskyy, O., Bessani, A., Fraga, J., Cunha, H. A., and Kinkelin, H. (2014). Trustworthy and resilient authentication service architectures. Technical report, SecFuNet Consortium.

of the Bouncy Castle Inc., L. (2014). The Legion of the Bouncy Castle. `https://www.bouncycastle.org`.

openid-server Community (2010). JOIDS. `goo.gl/nkyZIj`.

OpenID4Java (2013). OpenID 2.0 Java libraries. `goo.gl/c3kFV`.

Prince, M. (2012). Ceasefires Don't End Cyberwars. `goo.gl/GI506`.

Prince, M. (2013). The DDoS that almost broke the internet. `goo.gl/g5Qs1`.

Recordon, D. and Reed, D. (2006). Openid 2.0: A platform for user-centric identity management. In *ACM WDIM*, DIM '06, pages 11–16. ACM.

Rescorla, E. (1999). Diffie-Hellman Key Agreement Method. `goo.gl/Ta2jhI`.

Urien, P. and Dandjinou, M. (2006). Introducing smartcard enabled radius server. In *International Symposium on CTS*, pages 74–80.

Uruena, M., Munoz, A., and Larrabeiti, D. (2012). Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites. *Multimedia Tools and Applications*.

Verissimo, P. E., Neves, N. F., Cachin, C., Poritz, J., Powell, D., Deswarte, Y., Stroud, R., and Welch, I. (2006). Intrusion-tolerant middleware: The road to automatic security. *Security & Privacy, IEEE*, 4(4):54–62.

Viejo, A. (2014). Microsemi Enables FPGA-Based Root-of-Trust Solution for Embedded Systems with Introduction of Secure Boot Reference Design. `goo.gl/OYCDNT`.