Modelagem e Detecção de Falhas em Soluções para Armazenamento Seguro em Nuvens usando Redes de Petri Coloridas: Um Estudo de Caso*

Carlos André Batista de Carvalho[1,2]†, Rossana Maria de Castro Andrade[1]‡, Miguel Franklin de Castro[1], Nazim Agoulmine[3]

[1]Mestrado e Doutorado em Ciência da Computação (MDCC),
Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat),
Universidade Federal do Ceará (UFC)

[2]Departamento de Computação, Universidade Federal do Piauí (UFPI)

[3]IBISC Laboratory, University of Evry (UEVE)

candrebc@ufpi.edu.br, rossana@ufc.br, miguel@ufc.br,

nazim.agoulmine@ufrst.univ-evry.fr

Abstract. *Some users and organizations resist to adopt solutions in clouds, due to concerns about data security and privacy. Cloud clients have a limited vision of the provider security, and they request security guarantees over data storage. This paper shows the results of the modeling, using Colour Petri nets, of an existing solution for secure storage. The results prove the viability of the use of formal methods to verify security properties of a system, and, by doing so, it was possible to identify scenarios in which security violations were not detected by this solution.*

Resumo. *Existem usuários e organizações que resistem em adotar soluções em nuvens, em virtude da preocupação com a segurança e privacidade no armazenamento dos dados. Clientes de serviços em nuvens possuem uma visão limitada da segurança do provedor e necessitam de garantias quanto a segurança dos dados armazenados. Este artigo apresenta resultados da modelagem em Redes de Petri Coloridas de uma solução existente para armazenamento seguro dos dados, identificando cenários em que violações de segurança não eram detectadas, e demonstrando a viabilidade do uso de métodos formais para verificar propriedades de segurança.*

1. Introdução

A computação em nuvem é um paradigma que permite a uma organização concentrar seus esforços em sua atividade fim, terceirizando seus recursos de Tecnologia da Informação (TI). É possível utilizar recursos computacionais de uma nuvem pública, reduzindo investimentos em infraestrutura e pagando apenas pelos recursos consumidos. Além de

*Este trabalho foi parcialmente financiado pelo programa de cooperação internacional STIC-AMSUD (Projeto SLA4CLOUD)

†CAPES/FAPEPI Doctoral Scholarship (MDCC/DC/UFC)

‡CNPq Research Scholarship

reduzir custos, o uso de serviços elásticos permite alocar ou desalocar recursos dinamicamente de acordo com as necessidades de um cliente [Costa et al. 2011]. A computação em nuvem é uma realidade, com investimentos de grandes empresas e a oferta de diversos serviços. Contudo, devido à perda de controle sobre a infraestrutura computacional, algumas organizações resistem em adotar soluções em nuvem, em virtude da preocupação com a segurança e privacidade dos seus dados [Sun et al. 2014, Subashini and Kavitha 2011].

Os provedores de computação em nuvem implementam controles de segurança para atender as necessidades dos seus clientes. Esses controles são baseados em *frameworks* e normas de segurança, elaboradas por órgãos de padronização tais como ISO (*International Organization for Standardization*), NIST (*National Institute of Standards and Technology*) e CSA (*Cloud Security Alliance*) [Luna et al. 2015]. No entanto, os clientes possuem apenas uma visão limitada da segurança de um provedor, requisitando mecanismos que ofereçam uma maior transparência e garantias quanto a segurança dos serviços contratados [Luna et al. 2015].

Nesse contexto, pode ser definido um acordo de nível de serviço (*Service Level Agreement* – SLA), para fornecer as garantias e a transparência desejada. Contudo, os aspectos de segurança não são tradicionalmente cobertos pelos SLAs [Rong et al. 2013]. O SLA da Amazon EC2, por exemplo, garante apenas a disponibilidade dos serviços em pelo menos 99.95% do tempo[1]. Assim, pesquisas têm sido realizadas buscando soluções para aumentar a confiança em provedores de computação em nuvem [Bamiah et al. 2014, Habib et al. 2011]. Em especial, pode-se destacar soluções de auditoria, direcionadas a demonstração da segurança e a detecção de eventuais violações [Popa et al. 2011, Hwang et al. 2014].

[Subashini and Kavitha 2011] descrevem diversos aspectos de segurança a serem abordados em soluções para segurança em nuvens, e [Rong et al. 2013] acreditam que a maior preocupação é referente à garantia da integridade e confidencialidade dos dados armazenados em nuvem. Além disso, [Popa et al. 2011] identificaram outras duas propriedades fundamentais a serem garantidas por soluções de armazenamento seguro: *write-serializability* e *freshness*. Os autores propõem ainda uma solução para detectar violações das propriedades de segurança, denominada *CloudProof* [Popa et al. 2011]. No entanto, é fundamental que soluções de auditoria sejam adequadamente analisadas e validadas. A validação de soluções para armazenamento seguro em nuvens é normalmente realizada apresentando teoremas, que nem sempre são provados formalmente [Wei et al. 2014, Popa et al. 2011, Hwang et al. 2014, Guan et al. 2015].

Por outro lado, métodos formais tem sido utilizados nas últimas décadas para modelagem de sistemas, permitindo a eliminação de ambiguidades e a validação por técnicas automáticas de verificação de modelos (*i.e. model checking*) [Clarke and Wing 1996]. Esses métodos, entretanto, não tem sido aplicados no projeto de soluções para armazenamento seguro. Todavia, os mesmos podem ser utilizados para demonstrar propriedades de segurança e encontrar falhas no projeto dessas soluções.

No estudo de caso, apresentado nesse artigo, o *CloudProof* foi modelado e erros de projeto foram detectados. Com o uso de Redes de Petri Coloridas, foi possível identificar cenários em que violações de segurança não eram detectadas pela solução proposta

[1] http://aws.amazon.com/ec2/sla/

inicialmente. Assim, este artigo demonstra a viabilidade do uso de métodos formais para modelagem e detecção de eventuais falhas no projeto de soluções para armazenamento seguro. Além disso, a modelagem permite uma análise detalhada, observando aspectos fundamentais para o funcionamento correto de uma solução. O *CloudProof* foi selecionado como estudo de caso desta pesquisa, por ser uma das primeiras iniciativas propostas para detecção de violações de propriedades de segurança, e por ser referência em pesquisas sobre o tema.

O restante do artigo está organizado da seguinte forma. Na Seção 2, é discutido sobre o armazenamento seguro em nuvens, descrevendo o *CloudProof*. Os uso de métodos formais para especificação e validação de sistemas são expostos na Seção 3. Trabalhos relacionados, sobre a verificação de propriedades de segurança, são abordados na Seção 4. Na Seção 5, o resultado da modelagem e análise do *CloudProof* é detalhado. Por fim, as considerações finais são apresentadas.

2. Armazenamento seguro em nuvem

Ao utilizar ambientes terceirizados de computação em nuvem, a segurança dos dados armazenados e processados na nuvem pode ser comprometida. As soluções propostas na literatura abordam diferentes aspectos, incluindo: privacidade; prova de recuperabilidade e localização [Albeshri et al. 2012]; compartilhamento e controle de acesso [Yu et al. 2010]; e remoção segura de dados [Vanitha and Kavitha 2014]. Existem ainda abordagens diferenciadas, tais como operações com dados cifrados [Samanthula et al. 2012] e a manipulação de dados anônimos [Zhang et al. 2014] para prover a privacidade dos dados.

No âmbito desta pesquisa, deseja-se prover confiança em serviços de computação em nuvem, por meio de mecanismos de auditoria que detectem eventuais violações de propriedades de segurança. [Popa et al. 2011] identificaram as seguintes propriedades de segurança desejáveis para armazenamento seguro de dados: confidencialidade; integridade; *write-serializability*; e *freshness*. A confidencialidade e integridade são propriedades de segurança frequentemente citadas na literatura, referente a garantias quanto a leitura e alteração de dados apenas por usuários autorizados. Ao prover *write-serializability* e *freshness*, um provedor garante, respectivamente, a consistência quanto a ordem das operações de escrita e a leitura de dados atualizados.

[Popa et al. 2011] descrevem ainda um sistema, denominado *CloudProof*, que permite que um cliente possa detectar e provar a ocorrência da violação de alguma propriedade. Infelizmente, esse sistema não detecta violações de confidencialidade, apesar de permitir que os usuários criptografem os dados a serem armazenados. Nesse sistema, o cliente (*i.e. owner*) contrata um serviço de armazenamento de um provedor de nuvem, disponibilizado a usuários autorizados. Esses usuários realizam operações de escrita e leitura de blocos por meio dos comandos *get* e *put* de uma interface. O bloco é a unidade básica de armazenamento manipulada pelo sistema.

No *CloudProof*, um cliente deve determinar os usuários que podem acessar o sistema, configurando o controle de acesso aos dados. Além disso, um cliente pode realizar a auditoria do sistema, comprovando a conformidade do mesmo com as propriedades citadas. O controle de acesso é baseado nas listas de controle de acesso (*Access Control List* - ACL) de cada bloco. Uma ACL determina quem pode ler e/ou escrever em um

bloco. É valido destacar que as ACLs podem ser atualizadas, permitindo revogar o acesso a determinado usuário ou liberar o acesso a novos usuários. A leitura de um bloco é feita com uma chave secreta compartilhada por um sistema eficiente de distribuição, baseado nas técnicas de *Broadcast Encryption* e *Key Rotation*.

Os usuários com permissão de escrita devem assinar um bloco sempre que modificá-lo. Com essa assinatura, o provedor pode verificar se a atualização de um bloco foi feita por um usuário autorizado, garantindo a integridade do sistema. Se o provedor permitir a escrita de blocos com assinaturas inválidas, o mecanismo de atestações irá detectar a violação. Sempre que um usuário autorizado ler um dado, o provedor envia um atestado de que está enviando o dado correto. Esse atestado contém a assinatura do bloco a ser verificada pelo usuário. Sempre que o usuário escrever um bloco, é enviado também um atestado de escrita, solicitando a atualização dos dados e o provedor responde com outro atestado, informando que o bloco foi atualizado. A Figura 1 apresenta as estruturas desses atestados.

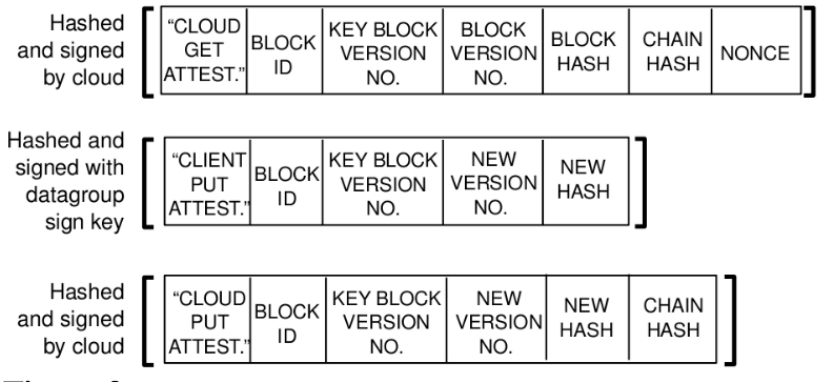

Figura 1. Estrutura dos atestados [Popa et al. 2011]

A versão do bloco e o *hash* é utilizado para garantir a *write-serializability* e a *chain hash* para garantir a *freshness*. O cálculo da *chain hash* é feito com os dados atuais da atestação e a *chain hash* da atestação anterior. O *nounce*, enviado pelo usuário ao solicitar a leitura de um bloco, é usado para evitar que o provedor envie um atestado antigo. É importante ressaltar que um usuário não pode enganar um provedor com falsas acusações.

A integridade dos dados armazenados é facilmente comprovada por meio das assinaturas dos blocos. Após um determinado período, denominado época, uma auditoria deve ser realizada para garantir que as outras propriedades não foram violadas. No processo de auditoria, o cliente ordena as transações de determinado bloco, e verifica suas versões ao longo de toda cadeia de atestados, detectando eventuais violações. Os atestados são enviados, pelos usuários, ao cliente, e ao final da auditoria podem ser descartados. Se necessário, o provedor deve fornecer algum atestado pendente e pode ser penalizado, caso o provedor não colabore com a auditoria. O *CloudProof* foi implementado na nuvem da Microsoft e [Popa et al. 2011] apresentam uma avaliação do sistema, demonstrando sua eficiência e escalabilidade.

3. Métodos formais

A confiabilidade e robustez de sistemas computacionais são influenciadas pela ausência de erros nesses sistemas e conformidade com os requisitos dos mesmos. Nesse contexto, métodos formais podem ser utilizados na especificação e validação de um sistema [Clarke and Wing 1996, Ardis 1997]. A especificação de um sistema é baseada em linguagens formais, que possuem sintaxe e semântica definidas matematicamente, proporcionando a eliminação das inconsistências existentes na descrição de sistemas. As ferramentas e técnicas desenvolvidas para linguagens formais permitem validar um sistema, antes de sua implementação. É possível realizar simulações em um sistema modelado formalmente, mas não é possível simular todos os cenários possíveis, garantindo que determinadas propriedades sejam satisfeitas. Então, duas abordagens são normalmente utilizadas no processo de validação de um sistema: verificação de modelos e prova de teoremas [Clarke and Wing 1996].

Redes de Petri, Estelle, LOTOS e SDL são exemplos de linguagens formais utilizadas para especificação de sistemas [Ardis 1997]. As Redes de Petri Coloridas (*Coloured Petri Nets* - CPNs) são amplamente utilizadas para especificação e verificação de protocolos de segurança [Pommereau 2010, Seifi 2014, Bouroulet et al. 2008]. CPN é uma linguagem que permite a modelagem gráfica de sistemas, com auxílio de uma linguagem de programação (*e.g.* CPN ML, usada na ferramenta *CPN Tools*) [Jensen and Kristensen 2009]. Nesta pesquisa, a *CPN Tools* foi utilizada para a modelagem do *CloudProof*, em virtude da facilidade no uso dessa ferramenta, da ampla documentação existente sobre CPNs, e por ser adequada para especificação de protocolos de segurança.

A modelagem de um sistema em redes de Petri é feita descrevendo um grafo composto por vértices que indicam os lugares e as transições do sistema. Arcos direcionados conectam lugares à transições ou vice-versa, mas nunca dois lugares ou duas transições. Os lugares de uma rede de Petri contêm fichas, que podem habilitar transições a serem disparadas. Ao disparar uma transição, fichas dos lugares de origem são consumidas, e novas fichas são adicionadas nos lugares de saída, conforme os pesos informados nos arcos. A escolha pela transição a ser disparada é não-determinística. A disposição inicial das fichas nos lugares é denominada de marcação inicial, e cada marcação durante o funcionamento de um sistema representa um estado do mesmo.

A rede de Petri colorida é uma extensão da rede de Petri, que permite atribuir cores ou valores às fichas. De modo semelhante às linguagens de programação é possível definir que tipo de informação pode ser armazenada em cada lugar e, ainda, atribuir ou verificar os valores das fichas. Assim, o tamanho dos modelos criados é reduzido. Existem ainda outras extensões, incluindo as redes de Petri hierárquicas e as temporizadas, que possibilitam respectivamente dividir um sistema em módulos e adicionar temporizadores nas transições. Esse tópico é aprofundado em [Jensen and Kristensen 2009], com detalhes sobre os tipos de redes de Petri e definições formais.

Como exemplo, pode-se destacar o trabalho de [Jensen et al. 2007], que descreve a modelagem, utilizando a *CPN Tools*, de um protocolo de comunicação segundo a estratégia *stop-and-wait* (vide Figura 2). Nesse trabalho, é possível compreender o funcionamento da ferramenta, bem como os aspectos essenciais para modelagem de CPNs. Nessa ferramenta, simulações automáticas ou manuais podem ser executadas para verifi-

car o funcionamento de uma CPN, e os resultados podem ser visualizados em diagramas de sequência (Message Sequence Charts - MSC). Ao usar simulações, é possível detectar erros, mas não é possível provar que eles não existem, por não ser viável verificar todos os cenários existentes. Assim, para validar um modelo é necessário analisar o espaço de estados (*space state*) de uma CPN. O espaço de estado é um grafo direcionado que representa todos os fluxos de execução possíveis, em que cada arco indica uma transição do sistema, alterando a marcação de uma CPN.

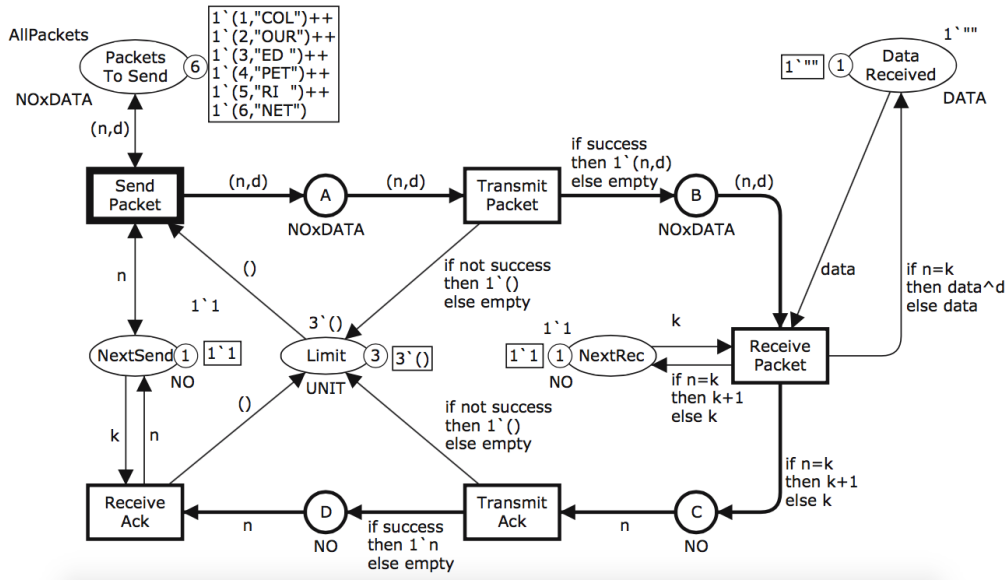

Figura 2. Modelagem do protocolo *stop-and-wait* [Jensen et al. 2007]

No entanto, é possível que uma CPN não seja representada por um espaço de estados finito. No protocolo *stop-and-wait*, por exemplo, o emissor pode enviar diversas vezes a mesma mensagem, devido a possibilidade de perda da mesma ou de sua confirmação. Assim, para evitar a retransmissão de infinitas mensagens e consequentemente a explosão de estados, foi limitada, em três unidades, a quantidade de mensagens e/ou confirmações transitando simultaneamente na rede, conforme visualizado na CPN da Figura 2. Além do espaço de estados, a *CPN Tools* gera um relatório com diversas informações, incluindo as marcações possíveis de cada lugar.

Para verificar se um sistema modelado satisfaz determinada propriedade e validar uma CPN, deve ser realizada uma verificação de modelos, por meio de funções em CPN ML ou em lógicas temporais [Jensen et al. 2007]. Ao usar uma função em CPN ML, todas as marcações do sistema são analisadas, observando as fichas armazenadas em cada lugar para detectar alguma violação. O caminho executado até a violação pode ser analisado para propor correção do modelo.

Para validação do protocolo *stop-and-wait*, [Jensen et al. 2007] definiram uma função para verificar se é possível reenviar mensagens antigas já recebidas e confirmadas corretamente. Desse modo, por exemplo, a primeira mensagem não pode ser reenviada após a segunda mensagem ser transmitida. A função criada compara, em cada marcação do espaço de estados, o número de sequência da próxima mensagem a ser enviada com o número de sequência esperado da mensagem a ser recebida. Apesar de não alterar a men-

sagem recebida ao final da transmissão, a CPN modelada não respeita essa propriedade da estratégia *stop-and-wait*. Então, é necessária uma alteração no modelo, de modo que, ao receber uma confirmação, o número de sequência da próxima mensagem a ser enviada seja atualizado pelo maior valor entre o número de sequência da confirmação recebida e o o número de sequência da última mensagem enviada.

O processo de verificação de modelos pode ainda ser realizado utilizando lógicas temporais, como a *Linear Temporal Logic* (LTL) e a *Computational Tree Logic* (CTL) [Clarke et al. 1986]. Ao utilizar uma lógica temporal pode-se verificar o comportamento de um sistema ao longo dos fluxos/caminhos de execução. É possível, por exemplo, definir fórmulas lógicas para verificar se todos os caminhos atendem determinada condição ou se existe algum caminho em que uma condição qualquer é satisfeita. Assim, poder-se-ia desenvolver uma fórmula lógica, com base nas mensagens enviadas, que detectasse o mesmo erro exposto anteriormente. Um manual, detalhando o uso de fórmulas em CTL na *CPN Tools*, está disponível em [Christensen and Mortensen 1996].

4. Trabalhos relacionados

A literatura descreve estudos em que métodos formais são utilizados para demonstrar a segurança em protocolos, APIs (*Application Programming Interfaces*) e processos de negócios [Armando et al. 2014]. Pesquisas detalham falhas, especialmente, em protocolos de autenticação, tais como: baseados em *Single Sign-On* (SSO) [Armando et al. 2013], Kerberos [Panti et al. 2002], *Kao-Chow* (KC) [Bouroulet et al. 2008] e *Needham-Schroeder* (NS) [Pommereau 2010, Seifi 2014]. Nessas pesquisas, as violações de autenticidade foram detectadas em CPNs [Pommereau 2010, Seifi 2014, Bouroulet et al. 2008], ou usando os verificadores SATMC [Armando et al. 2013] e NuSMV [Panti et al. 2002].

O protocolo de autenticação NS é baseado em criptografia assimétrica e sua falha foi descoberta por [Lowe 1995], que apresentou ainda uma correção. Contudo, não foi realizada uma avaliação formal do protocolo, descrita em outros trabalhos [Pommereau 2010, Seifi 2014]. [Seifi 2014] propõe uma metodologia para a modelagem de protocolos de segurança utilizando redes de Petri. Nessa metodologia, uma CPN hierárquica é projetada com base nas mensagens do protocolo, usando uma abordagem *top-down*. Em seguida esse modelo preliminar é avaliado e só então o atacante é modelado. Por fim, as propriedades são traduzidas em fórmulas lógicas ou funções para validação de CPNs.

Nesse tipo de protocolo, o comportamento de um atacante é tradicionalmente definido segundo o modelo *Dolev-Yao* (DY) [Dolev and Yao 1983], em que o atacante pode: i) copiar ou bloquear mensagens; e ii) alterar ou criar mensagens, limitado por restrições criptográficas. O atacante não pode quebrar a criptografia, mas, caso conheça as chaves, pode decifrar mensagens e armazená-las em uma base de dados, junto com mensagens capturadas.

[Seifi 2014] descreve ainda estratégias para facilitar a modelagem e evitar a explosão de estados. É possível: i) definir lugares para indicação de erros e, assim, interromper a criação do espaço de estado ao atingir uma marcação inesperada; ii) construir modelos parametrizados, permitindo alteração dos mesmos por meio de constantes; iii) controlar a execução, usando fichas para habilitar ou desabilitar transições; ou ainda iv)

manipular as fichas que podem ser armazenadas em cada lugar para evitar que infinitas marcações representem um mesmo estado, contendo, por exemplo, fichas duplicadas.

5. Modelagem e análise do *CloudProof*

Neste artigo, a viabilidade do uso de métodos formais no projeto de soluções de armazenamento seguro em nuvens é demonstrada com a especificação e verificação formal do *CloudProof*, proposto por [Popa et al. 2011]. Além de indicar alguns pontos que necessitam de esclarecimentos, a análise realizada expõe falhas a serem corrigidas em uma nova versão dessa solução. Em virtude da integridade ser uma propriedade amplamente discutida na literatura e do *CloudProof* seguir procedimentos padronizados para assinar os dados, a pesquisa realizada foi restrita às propriedades *write-serializability* e *freshness*.

5.1. Modelagem do protocolo original

Algumas simplificações foram realizadas durante a modelagem, removendo aspectos que aumentariam a complexidade da CPN e, até mesmo, causariam a explosão de estados. Essas simplificações, no entanto, não alteram o comportamento do protocolo proposto e permitem a verificação das propriedades citadas. Nesse contexto, os campos **BlockID**, **KeyBlockVersion** e *Hash* (*i.e.* **BlockHash** ou **NewHash**) foram desconsiderados da estrutura dos atestados (vide Figura 1).

O campo *BlockID* não é necessário pois as operações de escrita e/ou leitura são realizadas em um único bloco. O gerenciamento de chaves não foi avaliado nesse momento e a chave utilizada não impacta nas propriedades analisadas. Assim, a versão da chave não precisa ser considerada. Na realidade, para facilitar a compreensão da modelagem, o conteúdo dos blocos é representado como texto em claro, sem prejuízo para o resultado da análise. Por fim, supõe-se que todas as mensagens enviadas são assinadas e o controle de acesso é adequadamente realizado, podendo ser omitido o *Hash* dessas mensagens.

O protocolo modelado é constituído das mensagens **get**, **getAck**, **put** e **putAck**. O **get** é uma mensagem apenas com o *nounce*, por não necessitar indicar o bloco a ser lido. O comando **getAck** contém o atestado de leitura do provedor e o bloco, representado por seu conteúdo e sua versão. Pelo comando **put**, um usuário solicita a gravação de um bloco, enviando o novo conteúdo e atestado de escrita do usuário, e recebe o atestado do provedor pela mensagem **putAck**. É importante destacar que usuário deve ter conhecimento da versão do bloco, pois o número da mesma é um dos metadados do bloco assinado pelo usuário.

A Figura 3 expõe a modelagem do *CloudProof* de acordo com as especificações em [Popa et al. 2011] e as simplificações realizadas. Para evitar a explosão de estados, três novos blocos e cinco *nounces* diferentes foram utilizados para compor as transações de uma época. O número da versão do bloco deve ser atualizado ao processamento das confirmações recebidas pelo usuário, para reconhecer as operações de escrita dos usuários. Outro detalhe é a representação do *hash* da cadeia de atestados por meio de uma tupla, composta pelo número de versão do bloco e de sequência. Apesar do protocolo prevê a interação de diversos usuários, em virtude do comportamento análogo de cada usuário, a CPN da Figura 3 detalha o funcionamento de um único usuário. Entretanto, simulações e a validações do protocolo foram realizadas também em modelos com dois usuários.

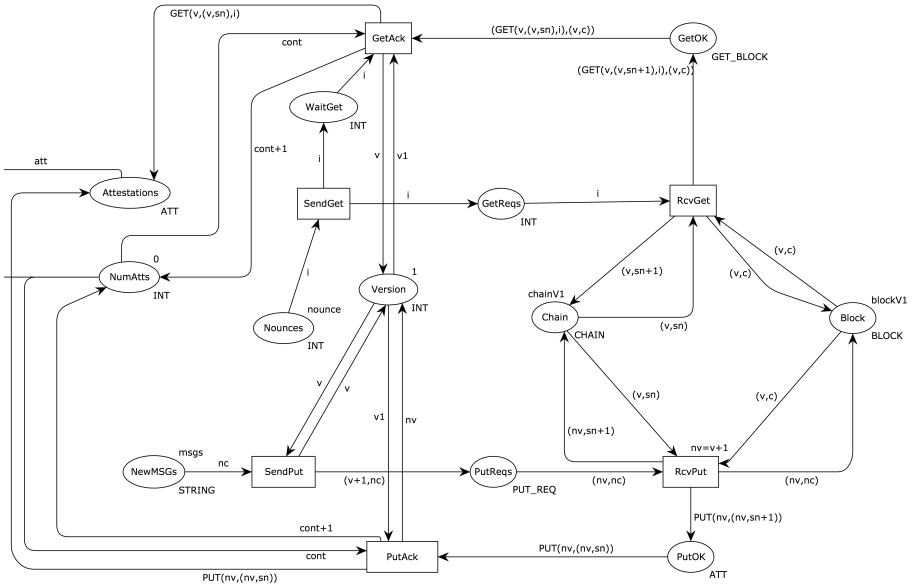

Figura 3. Modelagem do protocolo *CloudProof*

Simulações foram realizadas para demonstrar o funcionamento adequado do protocolo. Na validação do modelo, pôde-se perceber que existe a possibilidade do número de versão do bloco, gerenciado pelo usuário, esteja desatualizado. Esse fato ocorre, por exemplo, quando um usuário solicita a leitura de um dado e a escrita de um novo bloco. O provedor pode executar o comando de leitura, seguido pelo de escrita. Contudo, o usuário pode processar a confirmação da escrita antes da leitura. Assim, após o processamento da confirmação da leitura, o número da versão do bloco gerenciado pelo usuário estará desatualizado. Nesse caso, o modelo pode ser corrigido de modo que o maior número de versão fosse armazenado após o processamento da confirmação de leitura.

Por outro lado, nesse exemplo, o atestado de leitura não pode ser descartado, apesar da informação recebida está desatualizada. Esse descarte iria comprometer o processo de auditoria descrito posteriormente nesta seção. Nesse contexto, entende-se que a melhor solução seria o controle das solicitações por parte do usuário, permitindo uma única operação por vez. Com esse controle, uma nova operação de leitura ou escrita só poderia ser realizada após processar a confirmação da mensagem anterior. Esse controle facilitaria ainda a análise com a redução do espaço de estados.

No entanto, a implementação desse controle requer uma nova alteração no protocolo original, pois o mesmo não prevê que o provedor informe a um usuário que uma escrita não foi efetivada por fornecer um bloco com o número de versão desatualizado. Esse fato é factível, pois em um cenário com vários usuários, existe a possibilidade de dois usuários solicitarem concorrentemente a escrita de novos blocos, em virtude do controle das operações ser realizado pelos usuários. Nesse caso, apenas uma escrita seria efetivada e o outro usuário esperaria indefinidamente pela resposta. Ao fornecer uma confirmação negativa da operação de escrita, o provedor pode informar a versão atual do bloco, permitindo ao usuário criar um novo bloco com o número de versão correto. Outra opção seria solicitar previamente ao provedor o número de versão atualizado, contudo o número de mensagens aumentaria sendo necessária uma análise de desempenho que foge do escopo

deste artigo.

5.2. Processo de auditoria e análise de segurança

Além do protocolo, é necessário modelar o processo de auditoria para verificar as propriedades *write-serializability* e *freshness*. Violações de integridade podem ser detectadas em tempo real, analisando a assinatura das mensagens recebidas. Contudo, outras violações são detectadas apenas na auditoria, utilizando a cadeia de atestados. Em um cenário com vários usuários, por exemplo, não tem como um usuário saber se a informação recebida é ou não a mais atual.

A auditoria é realizada após a coleta, pelo cliente, dos atestados de todos os usuários. Em seguida os atestados são ordenadas, e declara-se que a segurança do provedor foi violada se a ordenação não for possível. Na modelagem da auditoria apresentada na Figura 4, a ordenação é baseada no número de sequência, e uma violação ocorre quando existem números de sequências iguais ou ausentes. Após a ordenação, é verificado se houve violações, comparando, com o valor esperado, as versões informadas nos atestados da última até a primeira operação. Uma violação de *freshness* (ou *write-serializability*) é detectada caso uma divergência seja encontradas em um atestado de leitura (ou escrita).

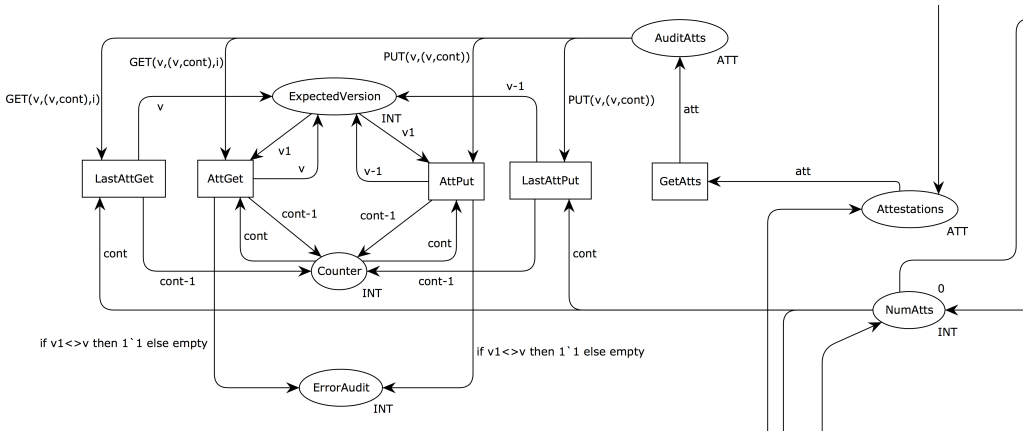

Figura 4. Modelagem do Processo de Auditoria do *CloudProof*

Nenhuma violação foi detectada, nas simulações realizadas, e fichas no lugar *ErrorAudit* não foram encontrados no espaço de estados. Além disso, foi desenvolvida funções para verificar se os números de séries dos blocos lidos e escritos caracterizam violações por parte do provedor. A auditoria exposta analisa as transações de uma única época, mas é importante ainda verificar as versões iniciais e finais de um bloco em épocas consecutivas.

No contexto do *CloudProof*, os ataques são caracterizados por provedores maliciosos que fornecem blocos desatualizados ou escrevem dados fora de ordem. Assim, CPNs diferentes foram modeladas para representar comportamentos maliciosos de um provedor. Para violar a *freshness*, o provedor foi projetado para armazenar todos os blocos gravados e escolher aleatoriamente aquele que deve ser retornado a um usuário. Um usuário pode até detectar uma violação em tempo real, se o provedor enviar um bloco antigo de acordo com o número de versão mantido pelo usuário, mas em um ambiente com vários

usuários, não é possível manter esse valor sempre atualizado. A violação, então, só é comprovadamente detectada durante a auditoria.

Os autores do *CloudProof* argumentam que o *nounce* é fundamental para evitar que um provedor elabore previamente uma atestação por desconhecer o *nounce* a ser utilizado pelo usuário. Assim, esperava-se que o *nounce* fosse suficiente para detectar uma violação de *freshness* em tempo real. Nossos testes provaram que o *nounce* não é necessário para identificar violações realizadas pelo provedor. Por outro lado, o *nounce* é fundamental para evitar ataques de reprodução (*replay*) em conformidade com o modelo *Dolev-Yao*. Porém, a ausência do *nounce* inviabiliza identificar se há um atacante externo ou um provedor malicioso. Atacantes externos tem suas ações limitadas, pois as mensagens transmitidas são assinadas e não podem ser alteradas sem serem detectadas. Em uma operação de escrita, ataques de *replay* são evitados ao não permitir a gravação de blocos que possuem números de versão antigos.

Em virtude de o usuário não saber previamente qual é a versão atualizada de bloco, é possível que solicite a escrita de um bloco com o número de versão incorreto. Um provedor malicioso pode efetuar a gravação, porém a violação da *write-serializability* é detectada na auditoria. No entanto, existe um cenário em que a violação não foi detectada pela auditoria. Um usuário pode solicitar uma escrita e um provedor malicioso, que informa a gravação do bloco, sem efetuar a mesma em disco. Se a próxima transação for outra escrita e todas as outras transações forem processadas corretamente, a violação não será detectada. Isso ocorre pelo fato da auditoria ser baseada na cadeia de atestados, que será ordenada e validada sem erros.

Teoricamente, é inviável que um provedor saiba previamente se serão realizadas duas gravações seguidas. Além disso, salvo em um sistema com controle de versão, o bloco antigo não será solicitado por um usuário posteriormente. Contudo, essa pode ser considerada uma falha no projeto do protocolo. É possível corrigir essa falha analisando todas as versões gravadas no sistema de controle de versão ou alterar o protocolo evitando que duas gravações seguidas possam ser realizadas, exigindo, por exemplo, uma leitura antes de toda gravação. Uma abordagem semelhante foi sugerida previamente, ao alterar o protocolo para incluir uma mensagem para verificar qual o número atualizado da versão de um bloco. Todavia é necessário ainda uma análise mais completa para avaliar o impacto da adição de novas mensagens.

Em uma outra análise, o provedor foi modelado para processar corretamente as operações de um usuário, construindo uma cadeia de atestados com base apenas nas operações desse usuário. As operações de um segundo usuário são respondidas, mas não são efetivadas no provedor. Nenhuma violação será detectada caso, o segundo usuário não envie os atestados para a auditoria. Esse usuário pode estar indisponível, ser malicioso ou, até mesmo, estar sob ataque e informar que não executou transações. O uso de um *broker* ou um terceiro confiável para centralizar as solicitações dos usuários é uma abordagem interessante, permitindo tratar esse tipo de situação. Além disso, essa abordagem permitiria a detecção de violações em tempo real.

O *roll-back attack*, descrito por [Hwang et al. 2014], ocorre quando um provedor descarta uma série de transações após um incidente ou ataque, e restaura o sistema para um estado consistente. Nesse cenário, o cliente pode detectar violações, pois os usuários

mantêm todos os atestados, incluindo os das transações descartadas. O *CloudProof* foi desenvolvido para identificar provedores maliciosos. Apesar da auditoria detectar violações de segurança, foram encontrados comportamentos maliciosos que não são detectados, comprometendo a segurança do processo.

A abordagem descrita neste artigo pode ser utilizada para verificar outras propriedades de segurança em protocolos para armazenamento seguro. Contudo, é necessário destacar que não é possível usar a mesma para avaliar a segurança de algoritmos criptográficos isoladamente. Assim, a verificação, por exemplo, da confidencialidade é restrita a observar quais as informações podem ser obtidas por um atacante, e se as chaves criptográficas são atualizadas adequadamente.

Por fim, alguns aspectos do protocolo não foram detalhados adequadamente em [Popa et al. 2011], como a adição e remoção de blocos. É possível, ainda, supor que existem duas cadeias de atestados diferentes, pois os autores usam, em um determinado momento, o termo cadeia de escrita (*i.e. write chain*). Contudo, testes comprovaram que deve ser utilizada uma única cadeia de atestados, demonstrando as vantagens do uso de métodos formais para a eliminação de dúvidas e ambiguidades. O armazenamento, pelo provedor, apenas do último atestado de escrita enviado é outro aspecto a ser esclarecido, pois os blocos são assinados e só poderiam ser criados por usuários autorizados. Uma análise mais completa também é desejável, avaliando a segurança do controle de acesso e gerenciamento das chaves.

6. Conclusão e trabalhos futuros

Métodos formais são amplamente utilizados na literatura para especificação e verificação de sistemas. No entanto, a validação de soluções para segurança em nuvem não tem sido apresentada com o formalismo adequado. Esta pesquisa apresenta a modelagem de uma solução de armazenamento seguro, proposta por [Popa et al. 2011]. No *CloudProof*, a auditoria é realizada para detecção de violações, mas a análise aqui realizada expõe cenários em que violações não são detectadas pelo sistema. Os resultados, apresentados neste artigo, demonstram as vantagens e a viabilidade do uso de métodos formais oferecer garantias quanto a segurança em ambientes de computação em nuvem.

Em virtude dos problemas destacados neste artigo, é necessário, como trabalho futuro, atualizar o *CloudProof* ou modelar uma nova solução, satisfazendo as propriedades de segurança desejadas. A modelagem em redes de Petri e a verificação de modelos pode ser utilizada para validar uma solução, e, como resultado, essas propriedades podem ser incluídas em um SLA. É possível ainda considerar aspectos negligenciados no protocolo modelado. O uso de chave compartilhada, por exemplo, não permite verificar a propriedade de irretratabilidade. Além disso, pode-se considerar métricas, como a repetição de chaves, na análise da confidencialidade. Um *broker* deve viabilizar o gerenciamento adequado de chaves e a detecção de violações em tempo real.

Diversas modelagens foram realizadas para representar comportamentos maliciosos de um provedor. Assim, um modelo, semelhante ao *Dolev-Yao*, pode ser proposto para caracterizar todos os comportamentos possíveis de um provedor. Nesse modelo, é possível encontrar comportamentos maliciosos não abordados neste artigo. Uma metodologia genérica pode ainda ser elaborada para ser usada na modelagem e validação de outras soluções sobre o tema. Por fim, novas pesquisas devem ser realizadas, identificando

limitações e novos horizontes para o uso de métodos formais no projeto e validação de soluções para segurança em computação em nuvem.

Referências

Albeshri, A., Boyd, C., and Gonzalez Nieto, J. (2012). Geoproof: Proofs of geographic location for cloud computing environment. In *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW'12*, pages 506–514.

Ardis, M. A. (1997). Formal methods for telecommunication system requirements: A survey of standardized languages. *Annals of Software Engineering*, 3(1):157–187.

Armando, A., Carbone, R., and Compagna, L. (2014). Satmc: A sat-based model checker for security-critical systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 31–45. Springer.

Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Pellegrino, G., and Sorniotti, A. (2013). An authentication flaw in browser-based single sign-on protocols: Impact and remediations. *Computers & Security*, 33:41–58.

Bamiah, M. A., Brohi, S. N., Chuprat, S., and lail Ab Manan, J. (2014). Trusted cloud computing framework for healthcare sector. *Journal of Computer Science*, 10(2):240–240.

Bouroulet, R., Devillers, R., Klaudel, H., Pelz, E., and Pommereau, F. (2008). Modeling and analysis of security protocols using role based specifications and petri nets. In *Applications and Theory of Petri Nets*, pages 72–91. Springer.

Christensen, S. and Mortensen, K. H. (1996). Design/cpn ask-ctl manual. *University of Aarhus*.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263.

Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643.

Costa, R., Brasileiro, F., Lemos, G., and Mariz, D. (2011). Sobre a amplitude da elasticidade dos provedores atuais de computação na nuvem. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 221–234.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208.

Guan, C., Ren, K., Zhang, F., Kerschbaum, F., and Yu, J. (2015). Symmetric-key based proofs of retrievability supporting public verification. In *Computer Security–ESORICS 2015*, LNCS, pages 203–223. Springer.

Habib, S. M., Ries, S., and Mühlhäuser, M. (2011). Towards a trust management system for cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 933–939.

Hwang, G.-H., Huang, W.-S., Peng, J.-Z., and Lin, Y.-W. (2014). Fulfilling mutual nonrepudiation for cloud storage. *Concurrency and Computation: Practice and Experience*.

Jensen, K. and Kristensen, L. M. (2009). *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media.

Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254.

Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol. *Information processing letters*, 56(3):131–133.

Luna, J., Suri, N., Iorga, M., and Karmel, A. (2015). Leveraging the potential of cloud security service-level agreements through standards. *IEEE Cloud Computing Magazine*, 2(3):32 – 40.

Panti, M., Spalazzi, L., and Tacconi, S. (2002). Using the nusmv model checker to verify the kerberos protocol.

Pommereau, F. (2010). Algebras of coloured petri nets. *LAP LAMBERT Academic Publishing*.

Popa, R. A., Lorch, J. R., Molnar, D., Wang, H. J., and Zhuang, L. (2011). Enabling security in cloud storage slas with cloudproof. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11*.

Rong, C., Nguyen, S. T., and Jaatun, M. G. (2013). Beyond lightning: a survey on security challenges in cloud computing. *Computers and Electrical Engineering*, 39(1):47–54.

Samanthula, B. K., Howser, G., Elmehdwi, Y., and Madria, S. (2012). An efficient and secure data sharing framework using homomorphic encryption in the cloud. In *1st International Workshop on Cloud Intelligence (Cloud-I)*.

Seifi, Y. (2014). *Formal Analysis of Security Properties in Trusted Computing Protocols*. PhD thesis, Queensland University of Technology.

Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11.

Sun, Y., Zhang, J., Xiong, Y., and Zhu, G. (2014). Data security and privacy in cloud computing. *International Journal of Distributed Sensor Networks*, 2014:1–9.

Vanitha, M. and Kavitha, C. (2014). Secured data destruction in cloud based multi-tenant database architecture. In *International Conference on Computer Communication and Informatics*.

Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y., and Vasilakos, A. V. (2014). Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258:371–386.

Yu, S., Wang, C., Ren, K., and Lou, W. (2010). Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9.

Zhang, X., Liu, C., Nepal, S., Yang, C., and Chen, J. (2014). Privacy preservation over big data in cloud systems. In *Security, Privacy and Trust in Cloud Systems*, pages 239–257. Springer Berlin Heidelberg.