UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Hugo Brito Lima

Uma Proposta de Protocolo Colaborativo e Seguro para Nuvens de Armazenamento de Dados

Belém

2017

Hugo Brito Lima

Uma Proposta de Protocolo Colaborativo e Seguro para Nuvens de Armazenamento de Dados

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação pela Universidade Federal do Pará.

Universidade Federal do Pará – UFPA

Instituto de Ciências Exatas e Naturais – ICEN

Faculdade de Ciência da Computação – FACOMP

Curso de Bacharelado em Ciência da Computação

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo

Belém

2017

Hugo Brito Lima

Uma Proposta de Protocolo Colaborativo e Seguro para Nuvens de Armazenamento de Dados

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação pela Universidade Federal do Pará.

Belém, 13 de março de 2017:

Prof. Dr. Roberto Samarone dos Santos Araújo
Orientador - UFPA

Prof. Dr. Josivaldo de Souza Araújo
Professor - UFPA

Prof. Dr. Raimundo Viégas Júnior
Professor - UFPA

Belém
2017

Agradecimentos

Agradeço primeiramente aos meus pais e a minha irmã, pelo carinho e por todos esses anos me apoiando de todas as formas.

Agradeço ao meu orientador, professor Samarone, pelo suporte, pela paciência, pela disposição e por todo o conhecimento repassado à mim.

Muito obrigado.

Resumo

Serviços de armazenamento em nuvem permitem um ambiente colaborativo no qual usuários podem compartilhar arquivos entre si. Para isso, geralmente, os usuários devem especificar quais arquivos eles querem compartilhar e com quem eles desejam compartilhar esses arquivos. Silva et al. (2014) introduziram uma ideia que desfaz a necessidade de que os usuários explicitamente informem com quem eles desejam compartilhar esses arquivos. O protocolo de Silva et al. (2014) permite que usuários possam compartilhar dados automaticamente pela identificação de interesses similares. Como resultado, os usuários não precisam conhecer informação alguma (e.g., endereço de email) sobre os outros usuários antes do compartilhamento. Todavia, a segurança dos dados armazenados não foi levada em consideração na construção do protocolo de Silva et al. (2014) e neste trabalho foi proposto um novo protocolo colaborativo que assegura os dados armazenados na nuvem enquanto mantém características-chave do protocolo anterior. Isso foi alcançado através da adoção de técnicas seguras, amplamente utilizadas na literatura, que garantem a segurança dos dados dos usuários.

Palavras-chave: caixas de interesses, deduplicação, criptografia, segurança, nuvens de armazenamento.

Abstract

Cloud storage services allow a collaborative environment in which users can share files between themselves. For that, users usually specify which files can be shared and which users can access those files. Silva et al. (2014) introduced an idea that does not require a previous statement specifying who may have access to those shared files. The protocol of Silva et al. (2014) allows users to share data automatically by means of their similar interests. As a result, users do not need to know any information about other users, such as email addresses, before sharing. However, the security of stored data was not approached in the scheme provided and in this present work it is proposed a new collaborative protocol that successfully secure data in the cloud, while it still maintains key-features from the previous protocol. This will be attained by the use of secure techniques, widely used in the literature, aiming at securing user's data.

Keywords: interest boxes, deduplication, cryptography, security, cloud storage.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AAtt	Autoridade de Atributos
ABE	Encriptação Baseada em Atributos
CE	Encriptação Convergente
CI	Caixa de Interesses
CIGer	Gerenciador de Caixas de Interesses
CSP	Serviço de Armazenamento em Nuvem
KP-ABE	Encriptação Baseada em Atributos com a Política na Chave Privada
PK	Chave Pública
SK	Chave Privada

Sumário

1 Introdução

Computação em Nuvem (*Cloud Computing*) é um termo que designa um modelo moderno de prestação de serviços computacionais pela web, que podem ser disponibilizados por demanda e de forma elástica; ela representa uma tendência para o modelo contemporâneo de computação. Segundo o NIST:

> "Computação em Nuvem é um modelo que permite o acesso ubíquo, conveniente e sob-demanda de recursos compartilhados e configuráveis (e.g., redes, servidores, dados, aplicações e serviços) que podem ser rapidamente alimentados e disponibilizados com o mínimo de esforço de gerenciamento ou interação do provedor de serviço"..." (NIST, 2011, tradução do autor).

Além da disponibilização de serviços computacionais, o modelo de computação em nuvem permite a colaboração entre usuários. De fato, diversos serviços de armazenamento permitem que usuários colaborem. Serviços de armazenamento em nuvem, tais como o Dropbox (DROPBOX, 2016) e o Microsoft Onedrive (ONEDRIVE, 2016), por exemplo, tornaram-se importantes ferramentas e fazem parte da rotina diária de trabalho. Esses serviços, além de permitirem que usuários armazenem arquivos nas nuvens de armazenamento, ainda disponibilizam um ambiente colaborativo no qual esses usuários podem compartilhar arquivos entre eles. Outros serviços, como os disponibilizados pela Google (DRIVE, 2016), permitem que usuários convidem outros colaboradores para editarem em tempo real documentos e compartilharem pastas *online*. Tais plataformas fazem parte do dia-a-dia de muitos usuários que precisam disponibilizar arquivos e colaborar com outros usuários.

Nos serviços anteriores (i.e.,Google Drive, Microsoft Onedrive e Dropbox), os usuários se conhecem por outros meios e adicionam manualmente outros colaboradores. Portanto, a colaboração não se estabelece de forma automática, pois usuários devem *explicitamente* indicar os usuários colaboradores. Geralmente o endereço de *email* é utilizado para registrar um novo colaborador que poderá desfrutar das informações disponibilizadas por outros usuários. Comumente as seguintes situações são encontradas nesses serviços de armazenamento: um usuário pode compartilhar um único arquivo que será partilhado com um outro usuário específico ou com vários outros usuários, outros serviços permitem que sejam criadas pastas onde todo conteúdo armazenado delas possa ser compartilhado com outros usuários especificados.

A necessidade de explicitar diretamente quais colaboradores poderão acessar um determinado conteúdo pode ser interpretada como uma limitação. Hipoteticamente, usuários que partilham de interesses parecidos e que não se conhecem, não poderão compartilhar

arquivos nesta situação. Haja vista que os métodos de colaboração convencionais mencionados anteriormente, não permitem que usuários que se desconhecem possam se encontrar automaticamente, apesar de seus interesses similares.

Como forma de eliminar a limitação mencionada, Silva et al. (2014) introduziram a ideia de *caixas de interesses*. Esse mecanismo permite a colaboração entre usuários baseando-se nos interesses em comum (e.g., pesquisa sobre a febre amarela). Para permitir a colaboração, cada usuário tem um diretório especial no serviço de nuvem. Quando um usuário quer compartilhar um arquivo, ele armazena o arquivo neste diretório especial. De acordo com os interesses do usuário, o serviço de nuvem identifica outros usuários com interesses similares e compartilha o arquivo entre esses outros usuários. Desta forma, o serviço de nuvem permite que usuários compartilhem arquivos sem a necessidade de se conhecerem previamente. Os usuários precisam apenas definir no serviço de nuvem os seus interesses.

A ideia introduzida por Silva et al. (2014) também utiliza a deduplicação de dados para encontrar outros possíveis colaboradores. O princípio é de que usuários que partilham de um mesmo arquivo, possivelmente também partilham interesses similares. Além da proposta de Caixas de Interesses, os autores também propuseram um protocolo que implementa essa ideia. No protocolo de Silva et al. (2014), cada interesse do usuário é modelado como atributo, e um diretório especial (i.e., caixa de interesses) é associado com os atributos do usuário. O serviço de nuvem usa esses atributos para compartilhar dados entre outros usuários.

A ideia da caixa de interesses introduziu um novo mecanismo de compartilhamento de arquivos na nuvem. No entanto, o protocolo proposto não implementa mecanismos de segurança. Especificamente, o protocolo não protege os arquivos armazenados de agentes maliciosos. Desta forma um adversário pode facilmente ler os arquivos armazenados nas caixas de interesses. Neste trabalho, será proposto um novo protocolo para o compartilhamento seguro de dados na nuvem, baseado em usuários com interesses similares. Este novo protocolo supera as fragilidades do protocolo original criado por (SILVA et al., 2014) e o utiliza como ponto de partida para o desenvolvimento desta pesquisa.

1.1 Motivação para o Trabalho

O protocolo de Silva et al. (2014) é uma proposta interessante, pois inova na forma como a colaboração em nuvem acontece (mais detalhes serão dados no capítulo seguinte), e o presente trabalho visa adequar o protocolo de Silva et al. (2014) para os riscos de um ambiente real ao propor um novo protocolo colaborativo e seguro. Como Chow et al. (2009) apresenta, a segurança é uma das principais dificuldades para adoção da computação em nuvem pelas empresas. E nas categorizações feitas por eles, o protocolo de Silva et al.

(2014) falha na seção de "segurança tradicional", pois faltam medidas que garantam: a segurança dos dados armazenados, a proteção dos dados do ataque de *insiders* (e.g., um hypervisor malicioso), por exemplo. O protocolo de caixas de interesses é uma abordagem nova e compatível com as necessidades modernas da computação e a colaboração em um ambiente seguro é uma qualidade compulsória no modelo contemporâneo de computação em nuvem.

1.2 Objetivos do trabalho

Objetivo geral:

- O objetivo geral deste trabalho é desenvolver um protocolo colaborativo de caixas de interesses com melhorias de segurança;

Objetivos Específicos:

- Estudar os principais mecanismos de segurança para nuvens de armazenamento de dados;
- Estudar o protocolo de caixas de interesses;
- Verificar a segurança do protocolo de caixas de interesses;
- Desenvolver um protocolo de caixas de interesses com melhorias de segurança;
- Verificar a segurança do novo protocolo;

1.3 Trabalhos Relacionados

Diversos serviços viabilizam o compartilhamento de dados utilizando nuvens de armazenamento. Citam-se alguns serviços bem conhecidos como o *Drive* da Google, o *Onedrive* da Microsoft, o *Dropbox*, entre outros. Nenhum desses, todavia, permite o compartilhamento de dados por nomeação implícita, ou seja, sem deliberar explicitamente quais arquivos deseja-se compartilhar e com quais usuários.

Além dos serviços anteriores, existem outras propostas como a de Koulouzis et al. (2012), que estabelece redes colaborativas através de federações, deixando vaga também a questão da segurança dos dados compartilhados. O trabalho de Chard et al. (2012) aborda o conceito de computação social, onde usuários de uma rede social (e.g., facebook) compartilham laços de amizade que podem ser utilizados para indicar possíveis colaboradores. Contudo, tal abordagem ainda limita o alcance desta rede de colaboração, visto que ainda é necessário pre-existir algum tipo de laço explícito entre dois usuários.

A proposta de Li et al. (2014) utiliza uma abordagem distribuída das chaves dos arquivos como forma de reduzir os custos (i.e., *overhead*) do gerenciamento das chaves dos arquivos. Eles alegam que uma deduplicação segura de blocos (e.g., encriptação convergente com deduplicação de blocos), que divide um arquivo em n blocos de tamanho fixo (e.g., 4 KB por bloco), gerará uma quantidade muito grande de chaves. Visto que cada bloco terá uma chave diferente, o custo (i.e., *overhead*) será muito grande: um arquivo de 1 TB gerará *8 GB de chaves*, se cada chave tiver o tamanho de um *hash* SHA-256 (LI et al., 2014). Para recuperar cópias dos arquivos se faz necessário passar por um número mínimo de servidores, eliminando a necessidade de rearmazenar novas chaves para cada novo usuário de um mesmo arquivo. Todavia, no presente trabalho aborda-se a deduplicação em nível de arquivo e portanto a quantidade de chaves geradas será muito menor. Além disso o objetivo da proposta de Li et al. (2014) é a colaboração tradicional: deliberação explícita de quais arquivos e para quais usuários, diferentemente da encontrada no protocolo de caixas de interesses.

Evidentemente, o trabalho de Silva et al. (2014) também pode ser citado como trabalho relacionado, visto que o protocolo de (SILVA et al., 2014) foi utilizado como ponto de partida para o presente trabalho. O protocolo de Silva et al. (2014) aborda amplamente o conceito de redes colaborativas, deduplicação e a relação implícita entre os usuários associados através dos interesses similares. No entanto, tal protocolo não aborda questões fundamentais de segurança para os dados armazenados pelos seus usuários. Desta forma, faz-se necessário propor um novo protocolo, com as características principais de (SILVA et al., 2014), mas com as adequações necessárias visando a implementação em um ambiente real e potencialmente inseguro.

1.4 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma: o Embasamento Teórico no Capítulo 2 aborda as técnicas utilizadas neste trabalho para o pleno funcionamento do novo protocolo proposto, tais como: a Criptografia Simétrica (Seção 2.2, Criptografia Assimétrica (Seção 2.3), Deduplicação de Dados (Seção 2.4), Deduplicação Segura (Seção 2.5), Encriptação Baseada em Atributos com a Política na Chave Privada (Seção 2.6) e O Protocolo de Caixas de Interesses (Seção 2.7). No próximo capítulo é introduzido o novo "Protocolo Colaborativo e Seguro para Nuvens de Armazenamento de Dados", proposta desse trabalho (Capítulo 3). Uma análise do novo protocolo é feita na Seção 4. Finalizando, a Conclusão e Trabalhos Futuros no Capítulo 5.

2 Embasamento Teórico

Neste capítulo serão introduzidos os tópicos que servem como técnicas-base para este trabalho. Primeiramente serão feitas considerações sobre funções hash, criptografia simétrica e assimétrica. Após, serão abordados os conceitos deduplicação de dados e deduplicação segura, a primeira foi utilizada por Silva et al. (2014) no protocolo proposto originalmente por eles, e a deduplicação segura será utilizada neste trabalho com o objetivo de incrementar a segurança do protocolo proposto, ao prover a versão segura desta técnica. Em seguida será introduzida a "Encriptação Baseada em Atributos com a Política na Chave Privada"(KP-ABE), mais uma técnica importante adotada neste trabalho que se relaciona bem com a ideia de "usuários x interesses"utilizada por Silva et al. (2014). Posteriormente será introduzido o protocolo proposto por Silva et al. (2014) com o objetivo de solidificar o entendimento sobre a ideia de Caixas de Interesses e após, algumas considerações sobre o protocolo de Silva et al. (2014), que justificam detalhadamente a necessidade deste trabalho.

2.1 Funções Hash

Uma função *hash* é uma função que mapeia uma entrada de tamanho variado em uma sequência com tamanho fixo e reduzido (e.g., MD5 = 128 bits, SHA-1 = 160 bits, SHA-3 = 512 bits (KATZ; LINDELL, 2014)). Se H é uma função *hash* e recebe como entrada um texto M, de tamanho variável, então H retorna o *hash* $h = H(M)$. Uma "boa"função *hash* recebe um bloco grande como entrada e distribui de forma uniforme e aparentemente aleatória as saídas (STALLINGS, 2011). Essas funções podem ser utilizadas para a verificação de integridade de uma mensagem (i.e., *checksums*), ou para a verificação de dígitos, e também para a geração de números aleatórios etc.

Na criptografia, as funções *hash* podem ser utilizadas em assinaturas digitais, códigos de autenticação de mensagens e outras formas de autenticação. Essas aplicações contam com a propriedade de funções de sentido único, presente nas funções hash. Uma função de sentido único é definida como a impraticabilidade de se recuperar o texto de entrada utilizando o texto *hash* de saída. Formalmente, uma função de sentido único f é uma função que, para um valor aleatório x, é extremamente difícil de encontrar a pré-imagem de $f(x)$. Uma função $f : \{0,1\}^n \rightarrow \{0,1\}^m$ é de sentido único se para todo algoritmo A de complexidade $\leq t$, observamos:

$\mathbb{P}_{x\sim\{0,1\}^n}[A(f(x)) = x' : f(x) = f(x')] \leq \epsilon$, onde ϵ é um valor muito pequeno (TREVISAN, 2009);

As funções *hash* possuem alguns requisitos, como os listados por Stallings (2011). São eles: tamanho variável de entrada (i.e., as funções *hash* podem receber como entrada qualquer tamanho de dados), tamanho fixo de saída (i.e., a função *hash* retorna como saída uma sequência de tamanho *fixo*), eficiência (i.e., um *hash* deve ser computacionalmente fácil de ser obtido) (STALLINGS, 2011).

Outra propriedade importante de uma função *hash* é a resistência a colisões, ou seja, duas entradas que mapeiem na mesma saída. Uma função H é resistente a colisões se for impraticável encontrar um algoritmo, que em tempo polinomial encontre colisões em H. A probabilidade de se encontrar colisões em uma função *hash* é regulada pelo chamado "paradoxo do aniversário", e ilustra-se da seguinte forma: se há 23 pessoas em uma sala, então existe a possibilidade de mais de 50% que duas pessoas na sala possuam o aniversário no mesmo dia (TREVISAN, 2009), ou seja: $1 - S!/((S-n)! \times S^n)$, onde S é o espaço amostral (e.g., 365 dias) e n é o tamanho da amostra (e.g., 23 pessoas na turma). Portanto, quanto maior o espaço amostral, mais difícil de serem encontradas colisões. Para aumentar a segurança no uso das funções hash, o *National Institute of Standards and Technology* (NIST) dos Estados Unidos, publicou a nova função *hash* criptográfica SHA-3, que visa substituir os esquemas utilizados atualmente (e.g., SHA-1, SHA-2). A função SHA-3 recebe uma entrada de tamanho qualquer e retorna uma sequência de 512 bits de tamanho, diminuindo a possibilidade de colisões e de ataques de força bruta (i.e., ataques que testam todas as combinações de *hash* possíveis). Tal adoção se faz necessária devido ao crescimento do poder de computacional.

Outra propriedade importante é a de avalanche definida por Feistel (1973), e denota que uma simples alteração no texto de entrada, de até mesmo 1 bit, altera o *hash* resultante de forma significativa. Por exemplo, o *hash* do tipo SHA-3 (National Institute of Standards and Technology, 2015) da palavra "crypto"resulta em:

601057e930458c0ad2deb75f514fa6e31149877287495c1959e1528bc686071dde5e99ec31a2445
2b2731409bc3f7521a4d5da00b116e21f5c88ef53765a8ee5

Se alterarmos a *string* para "crypt" e a introduzirmos na mesma função SHA-3, obtemos:

8ef0885a4cd511761d15f8f554de91acd731ee2ac60f34a7abb5091470a230791f2a90d8c3c5d830b
9d7738f336052e4bca2e586e9d2d02ca90035ff2d2b5c64

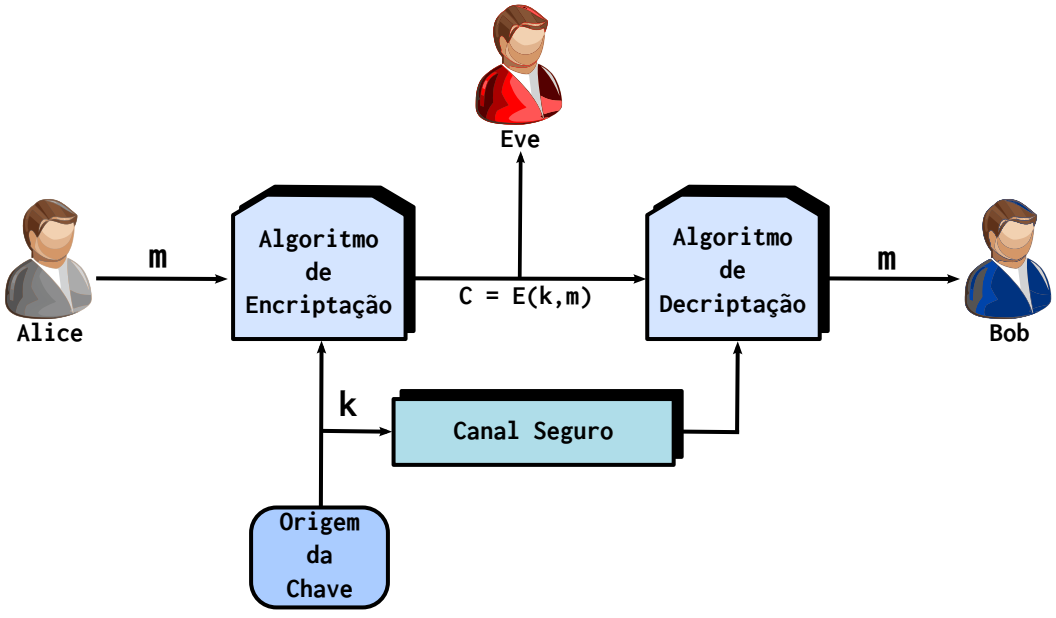
A facilidade na geração do *hash* e a extrema dificuldade na recuperação do texto de entrada, tornam essa ferramenta útil para o desenvolvimento de aplicações que requeiram a confirmação da veracidade de uma informação, verificação da integridade de uma mensagem, ou até mesmo a geração de uma sequência única e pseudo-aleatória. As funções *hash* foram utilizadas amplamente neste trabalho, como por exemplo na deduplicação segura de dados (Seção 2.4).

2.2 Criptografia Simétrica

A criptografia simétrica, também conhecida como encriptação convencional, refere-se a encriptação que utiliza a mesma chave secreta na cifragem e decifragem (e.g., AES). A segurança da chave criptográfica neste criptosistema é muito importante, visto que se um adversário conseguir obtê-la, ele poderá decifrar todas as mensagens que foram cifradas com a chave que ele conseguiu. Um criptosistema simétrico possui cinco componentes principais: ele possui um texto claro m que se deseja proteger o conteúdo; possui um algoritmo de encriptação E que executa transformações e substituições no texto claro m; possui uma chave *secreta* k que influencia a saída do algoritmo de encriptação, ou seja, para cada chave diferente será produzida uma saída diferente pelo algoritmo de encriptação de forma pseudo-aleatória; um texto cifrado c, que é um texto ininteligível produzido pelo algoritmo de encriptação na forma: $c \leftarrow E(k, m)$; possui também um algoritmo de decriptação D que executa o inverso do algoritmo de encriptação, ou seja, ele transforma o texto cifrado c no texto claro m da seguinte forma: $m \leftarrow D(k, c)$.

Na figura 1, Alice deseja enviar uma mensagem m para Bob, e decide fazê-lo em segredo, de forma que Eve não possa descobrir o conteúdo da mensagem. Alice encripta a mensagem m utilizando um algoritmo de encriptação simétrica $E(k, m)$. Ela escolhe uma chave secreta k e encripta a mensagem m em um texto cifrado c da seguinte forma: $c \leftarrow E(k, m)$. Ao receber o texto cifrado c, Bob utiliza a chave secreta k para decriptar o texto cifrado e recuperar a mensagem enviada por Alice.

Figura 1 – Ilustração do criptosistema simétrico.



Fonte – Elaborado pelo autor

O algoritmo de encriptação simétrica precisa ser robusto. Ainda que um adversário conheça o algoritmo e possua acesso a um ou mais textos cifrados, ele deve ser incapaz de decifrar o texto cifrado ou descobrir a chave secreta. Outro importante requisito tem a ver com a proteção da chave secreta. Remetente e destinatário da mensagem devem obter cópias da chave secreta através de um canal seguro (i.e., um canal de comunicação onde não exista a possibilidade de monitoramento por um adversário) (STALLINGS, 2011).

A criptografia simétrica adota a abordagem "computacionalmente segura". Que significa dizer que o algoritmo garante segurança prática baseada na limitação de poder computacional do adversário. Algoritmos como o AES, utilizam essa abordagem ao possibilitar a adoção chaves de tamanhos: 128, 192, e 256 bits. Uma chave de 128 bits de tamanho, por exemplo, permite até 2^{128} combinações possíveis, um número considerado grande (KATZ; LINDELL, 2014).

Desta forma, a encriptação simétrica é um criptosistema útil e prático para a proteção dos dados da possível análise de um adversário.

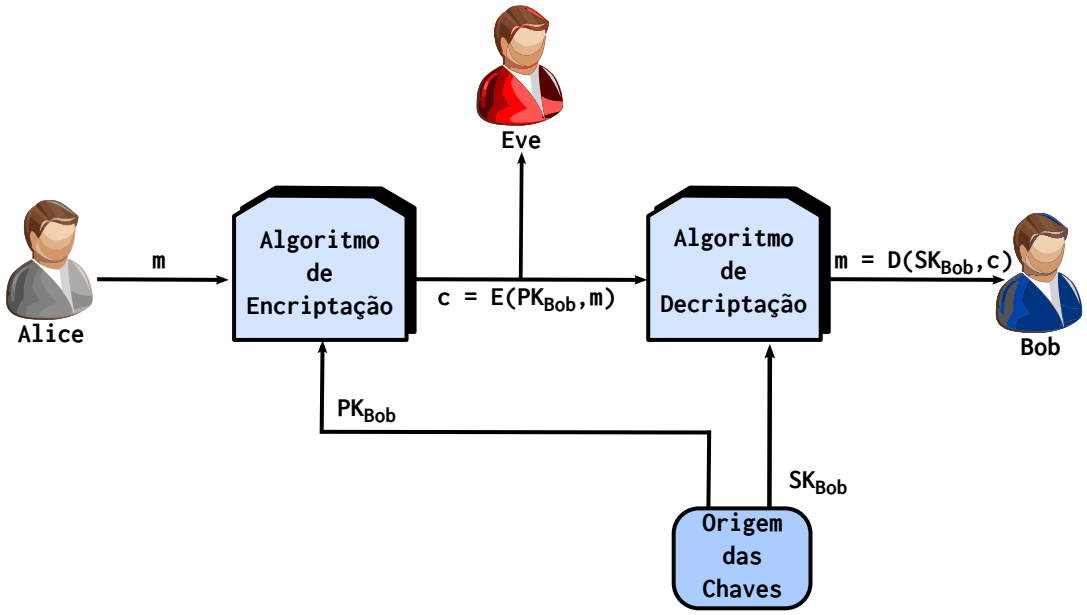
2.3 Criptografia Assimétrica

A criptografia assimétrica, também chamada de "criptografia de chave pública" é um criptosistema no qual a encriptação e a decriptação são efetuadas utilizando chaves diferentes. Neste esquema uma chave é pública e a outra é privada. Ela surgiu para sanar dois dos principais problemas da criptografia simétrica: a distribuição de chaves e o da assinatura digital. Na criptografia simétrica a distribuição das chaves de decriptação requer que dois comunicantes compartilhem a mesma chave que de alguma forma foi seguramente distribuída para eles ou tenham-nas recebido via centro de distribuição de chaves. Tais características vão contra a essência da criptografia: a possibilidade de manter segredo total sobre a própria comunicação (STALLINGS, 2011). O segundo problema foi o da assinatura digital. Whitfield Diffie, um dos criadores da criptografia de chave pública argumentou que se a criptografia fosse utilizada de forma ampla, inclusive para questões comerciais, então deveria ser possível assinar documentos eletrônicos. Uma alternativa equivalente à assinatura usada em documentos de papel deveria ser concebida (DIFFIE, 1988; DIFFIE; DIFFIE; HELLMAN, 1976).

A encriptação assimétrica pode ser usada para confidencialidade ou para autenticação (STALLINGS, 2011). Essa variação depende da ordem utilizada das chaves privadas e públicas. Visto que uma chave pública é conhecida por todos, um remetente pode encriptar uma mensagem com a chave pública do destinatário, e o destinatário pode decriptar a mensagem com a chave privada dele (Figura 2). Formalmente, se Alice deseja enviar uma mensagem m em segredo para Bob, Alice usa a chave pública PK_{Bob} de Bob e encripta o texto claro m: $c \leftarrow E(PK_{Bob}, m)$. Bob usa a chave privada dele SK_{Bob} para decriptar a

mensagem enviada por Alice: $m \leftarrow D(SK_{Bob}, c)$.

Figura 2 – Ilustração do criptosistema assimétrico para a encriptação.



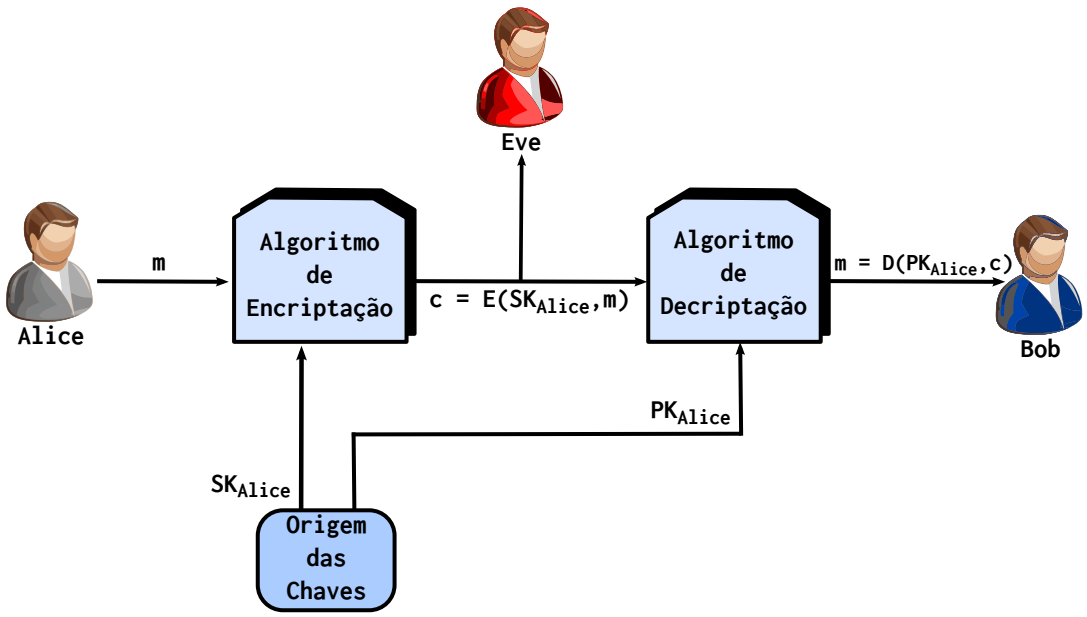
Fonte – Elaborado pelo autor

Utilizando a operação inversa, ou seja, encriptando com a chave privada e decifrando com a chave pública é possível criar um esquema de autenticação, visto que somente o real dono da chave pública poderia ter encriptado usando a chave privada dele. Se Alice deseja se autenticar para Bob, ou seja, confirmar que ela é realmente Alice, ela pode encriptar uma mensagem com a chave privada dela e Bob decifra o texto cifrado com a chave pública de Alice (Figura 3). Ou seja, Alice encripta a mensagem m com a chave secreta dela SK_{Alice}: $c \leftarrow E(SK_{Alice}, m)$. Bob recebe c e o decifra com PK_{Alice}: $m \leftarrow D(PK_{Alice}, m)$. Partindo do princípio que somente Alice poderia ter encriptado aquele texto, dado que a chave pública decifrou com sucesso, Bob pode ter certeza de que está falando com Alice.

A Criptografia Assimétrica possui alguns requisitos para o funcionamento ideal do criptosistema (STALLINGS, 2011):

- É computacionalmente fácil para um destinatário gerar um par de chaves pública (PK) e privada (SK).
- É computacionalmente fácil para um remetente, conhecendo a chave pública do destinatário, encriptar uma mensagem m: $c \leftarrow (PK, m)$.
- É computacionalmente fácil para o destinatário decifrar o texto cifrado c e recuperar a mensagem m: $m \leftarrow D(SK, c)$.

Figura 3 – Ilustração do criptosistema assimétrico para a autenticação.



Fonte – Elaborado pelo autor

- É impossível computacionalmente, um adversário descobrir uma chave secreta SK a partir da chave pública SK.
- É impossível computacionalmente para um adversário, conhecendo a chave pública PK e um texto cifrado c, recuperar a mensagem m.

O criptosistema assimétrico mais utilizado é o RSA (STALLINGS, 2011), e a dificuldade em atacar o RSA é baseada na dificuldade em fatorar números primos grandes (mais informações em (RIVEST; SHAMIR; ADLEMAN, 1978)).

O sistema de chave pública é utilizado nesse trabalho para alcançar alguns dos requisitos deste trabalho: como a segurança dos dados armazenados (i.e., utilizando a encriptação) e a confidencialidade das informações armazenadas nas caixas de interesses (e.g., atributos) (ver seção 2.7).

2.4 Deduplicação de Dados

A deduplicação permite a eliminação de dados redundantes. Ela é amplamente utilizada em nuvens de armazenamento (Dropbox (HACKERNEWS, 2010), Google Drive (DRIVE, 2016) e Microsoft OneDrive (ONEDRIVE, 2016)). Pois permite tanto a economia de espaço em disco quanto reduções no consumo de banda, pois estas nuvens também podem empregar análises de redundância ainda no computador do cliente, ou seja,

antes dos dados serem enviados. A Tabela 1 exemplifica os custos[1] de armazenamento de 10TB de dados por um mês associados a alguns serviços comerciais de armazenamento de dados.

Tabela 1 – Armazenamento de 10TB

Nuvem	Serviço	Armazenamento
Amazon Web Service (AWS)	S3	R$ 1085,79
Google Cloud Platform	Standard Storage	R$ 955,80
Microsoft Azure	Storage	R$ 1182,15

Fonte – (AMAZON WEB SERVICES (AWS), 2016; DRIVE, 2016; ONEDRIVE, 2016)

2.4.1 Definição

A Deduplicação é uma técnica de eliminação de dados redundantes. Ela visa reduzir o espaço ocupado em discos de armazenamento a partir da eliminação de cópias de um determinado dado, ou seja, ela elimina padrões de dados repetidos por comparações e cria n *links* de referência para as n cópias descartadas. Desta forma a redundância pode ser completamente eliminada e, dado que existam "u" usuários que armazenam um mesmo arquivo "M", os custos associados anteriormente a $\mathcal{O}(u * |M|)$ tornam-se $\mathcal{O}(u + |M|)$ (BELLARE; KEELVEEDHI; RISTENPART, 2013).

Esta tecnologia tornou-se comum a partir de meados dos anos 2000, primeiramente em dispositivos de backup. Estes dispositivos traziam nativamente a deduplicação como forma de armazenar a maior quantidade possível de dados nos discos de armazenamento. O uso da deduplicação ampliou seu escopo e hoje é utilizado com muita frequência em grandes *sets* de dados tais como em nuvens de armazenamento.

Além dos ganhos de espaço através da deduplicação de dados redundantes, a economia no tráfego de rede também podem ser obtida. Bellare e Keelveedhi (2015) enfatizam que transferir 1GB de dados para um servidor custaria o equivalente ao armazenamento destes dados por um mês. Portanto, a deduplicação é um mecanismo relevante para a utilização em nuvens de armazenamento. Para isso, implementam-se no lado do cliente artifícios que são utilizados para detectar se um cliente está enviando ou não uma cópia de um dado existente em uma nuvem de armazenamento. Esta identificação é comumente feita a partir da obtenção do *hash* do arquivo a ser enviado. Formalmente, digamos que Alice deseja enviar um arquivo f para seu diretório de arquivos em uma nuvem de armazenamento. Primeiramente é calculado o *hash* de f, que resulta h e este *hash* é enviado previamente para a nuvem que serve Alice para confirmar se h existe nesse serviço de armazenamento. Caso h, que representa o arquivo f já esteja registrado, a nuvem evita o

[1] Cotação do dólar: $1 = R$3,59 em 02/06/2016

envio do arquivo e cria um *link* de acesso para Alice. Caso não seja encontrado o *hash*, Alice prossegue com o procedimento normal: o arquivo é enviado e o *hash* do novo arquivo é registrado.

Há de se salientar que usuários partilhando do mesmo arquivo coexistem de forma independente, ou seja, um usuário pode não ter a mínima ciência de que está utilizando dados compartilhados. Neste sentido entra a questão da transparência de dispositivos distribuídos; visto que uma nuvem de armazenamento é também um sistema distribuído (MARINESCU, 2013) e busca tornar transparente a forma com a qual seus usuários a utilizam. Outro ponto importante, dá-se ao fato de que não há a necessidade da autorização prévia do usuário que enviou primeiramente um determinado dado para que este dado possa ser ou não deduplicado. A deduplicação, portanto, é primariamente um *interesse de quem fornece serviços de armazenamento* e muito menos do interesse dos próprios usuários, donos dos arquivos armazenados nestas nuvens.

2.4.2 Tipos de Deduplicação

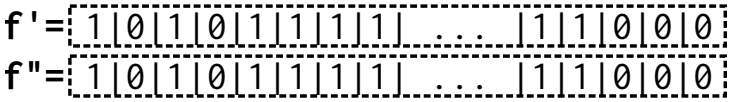
Como observado anteriormente, a deduplicação é uma técnica importante que visa a eliminação de dados redundantes. Contudo, existem diversos tipos de deduplicação, como por exemplo a deduplicação de arquivos e a deduplicação de blocos (e esta dividia em outros modos), tais técnicas serão introduzidas em seguida.

Deduplicação de Arquivos. A deduplicação de arquivos ocorre a partir da comparação de um arquivo com os arquivos de uma base de dados. De modo formal, se Bob envia a um servidor um arquivo f e posteriormente Alice envia o mesmo arquivo f para o mesmo serviço de armazenamento, o serviço de armazenamento pode reconhecer que os dois arquivos são idênticos e resolver armazenar somente uma cópia, eliminando a cópia redundante e criando um *link* de acesso de forma transparente para Alice. Desta forma ambos possuirão acesso ao mesmo arquivo de forma independente.

A comparação pode ser efetuada a partir da comparação bit-a-bit dos arquivos, mas geralmente implementa-se a comparação via cálculo do respectivo *hash* (e.g., SHA3). Os hashes calculados ficam registrados em uma tabela na nuvem de armazenamento e são relacionados aos arquivos que representam. Caso um novo arquivo seja enviado, imediatamente é calculado o *hash* deste arquivo e verificado se o valor já existe na tabela, a partir disso é decidido se a deduplicação será efetuada. Caso dois arquivos comparados confirmem-se idênticos (Figura 4), ou seja os bits de f' sejam iguais aos bits de f'', o *hash* resultante será o mesmo, executa-se em seguida a deduplicação.

Caso os arquivos difiram, os hashes calculados também serão diferentes e portanto significa dizer que o arquivo não está armazenado na nuvem de dados. A deduplicação não será aplicada e este novo arquivo será armazenado de forma convencional.

Figura 4 – Comparação entre arquivos iguais.



Fonte – Elaborado pelo autor

É necessário, todavia, apontar que caso existam dois arquivos f' e f'', e a diferença entre esses arquivos seja muito pequena, até mesmo de 1 bit (Figura 5), o *hash* calculado também será diferente, devido ao efeito avalanche das funções *hash* (FEISTEL, 1973). Apesar de grande parte do arquivo f' ser exatamente a mesma de um arquivo f'' já armazenado (i.e., grande parte dos bits de f' são idênticos aos valores de f'') , o sistema de armazenamento identificará dois arquivos diferentes e consequentemente guardará os dois arquivos.

Figura 5 – Comparação entre arquivos diferentes.

Fonte – Elaborado pelo autor

Deduplicação de Blocos. O modo de deduplicação de blocos consiste na divisão do arquivo em blocos; a partir desses blocos executam-se as devidas comparações necessárias para a deduplicação. Este tipo de deduplicação é útil, por exemplo, para o caso de arquivos modificados, pois as modificações tendem a alterar apenas alguns blocos, mantendo partes da versão antiga intactas. Desta forma permite-se que os blocos intactos possam ainda ser deduplicados.

Neste modo de deduplicação a quantidade de comparações é muito maior, dado que um arquivo possui tamanho l KB e utiliza, por exemplo, 4 KB de tamanho de chunk, um sistema de armazenamento efetuará até $l/4$ comparações para a deduplicação.

Dentro da deduplicação em blocos existem diversas práticas, ou modos. Cada modo utiliza alguma estratégia diferente para a divisão dos blocos e a forma como esses blocos serão divididos.

Modo Estático ou Divisão em Blocos Fixos: O "Modo Estático"ou "Divisão em Blocos Fixos"(MEYER; BOLOSKY, 2012) de deduplicação é um algoritmo linear que consiste na divisão de um arquivo em diversos blocos de tamanho fixo t (p.ex. 4KB) , e em seguida efetuam-se comparações. Um arquivo f possui f/t blocos e cada bloco tem seu valor de *hash* calculado e armazenado em uma tabela para agilizar comparações futuras.

A deduplicação ocorrerá quando um segundo usuário enviar um arquivo e, ao dividir esse arquivo em blocos de mesmo tamanho t, será possível identificar blocos idênticos aos registrados previamente na nuvem de armazenamento.

Figura 6 – Deduplicação em blocos de tamanho fixo.

Fonte – Elaborado pelo autor

Apesar da deduplicação em blocos no modo estático já garantir um desempenho melhor que o modo de arquivo, ainda existem situações onde a flexibilização do tamanho do bloco possa ser desejada, como o caso da figura 7.

Figura 7 – Deduplicação em blocos de tamanho fixo.

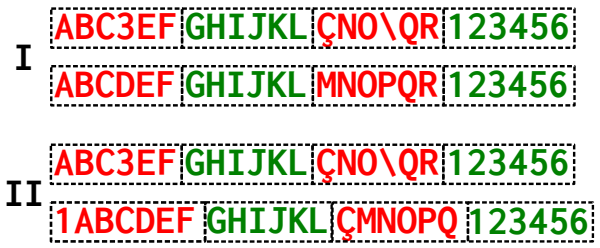
Fonte – Elaborado pelo autor

Na figura 7 pode-se notar que ao se deslocarem os blocos no modo fixo em um caractere, nenhuma redundância pôde ser encontrada. Em seguida será abordado um outro modo de deduplicação de blocos que poderá resolver o problema mencionado previamente.

Rabin Karp e Chunk variável: Para resolver o problema do modo de deduplicação de blocos de tamanhos fixos, no qual uma simples troca de posições (Figura 7) acomete na impossibilidade de se aplicar a deduplicação, é necessário tratar a divisão dos blocos de acordo com o conteúdo do fluxo de dados. Para tal, é necessário utilizar algum método que possibilite identificar um padrão dentro deste fluxo o qual possa ser dividido.

O algoritmo de "Rolling Hash"(KARP; RABIN, 1987) é utilizado para procurar por um padrão. Na Figura 8 A cor verde indica os blocos deduplicados. A primeira situação (I) indica dois arquivos que tiveram seus blocos deduplicados. A segunda situação (II) mostra esses mesmos dois arquivos, e apesar do segundo arquivo ter sido modificado, a deduplicação ainda pôde ser efetuada. Em linhas gerais, define-se uma janela (i.e., uma sequência delimitada do fluxo total de dados) de tamanho N. Depois calcula-se o *hash* dessa janela e procura-se no fluxo de dados por *fingerprints* que correspondam ao *hash* da janela. Se o *fingerprint* corresponder, então o algoritmo marca o enésimo elemento como limite para a divisão do bloco. Se o *fingerprint* não corresponder, então move-se a janela

Figura 8 – Ilustração do funcionamento do algoritmo Rabin-Karp.



Fonte – Elaborado pelo autor

em um elemento e recalcula-se o *rolling hash*. Repete-se o processo até o final do fluxo de dados.

2.5 Deduplicação Segura

Nesta seção é introduzida a noção de deduplicação segura. Em linhas gerais ela busca combinar os atributos da deduplicação convencional (ver 2.4) com a garantia de segurança para os dados armazenados. Para isso, técnicas de criptografia (e.g., hash, criptografia simétrica e assimétrica) são utilizadas na criação dos protocolos de deduplicação segura. Um algoritmo de deduplicação segura exige o armazenamento seguro de dados e, ao mesmo tempo, a deduplicação desses dados. A deduplicação segura será utilizada para substituir a deduplicação convencional adotada por Silva et al. (2014).

Técnicas convencionais de encriptação (e.g., encriptação simétrica) são incompatíveis com a deduplicação. Isto porque, digamos que Alice cifre um arquivo m com uma chave k', resultando em um texto cifrado c'; caso Bob resolva enviar o mesmo arquivo m e o encripte utilizando uma chave k'', o texto cifrado resultante será c'', que é diferente do texto cifrado armazenado previamente por Alice. Portanto, mesmo armazenando essencialmente o mesmo arquivo, *sob uma chave diferente*, obtêm-se textos cifrados diferentes, e isto impossibilita a deduplicação.

Para contornar o problema anterior, neste trabalho serão utilizadas as primitivas criptográficas *Message-Locked Encryption* (MLE) (BELLARE; KEELVEEDHI; RISTENPART, 2013), que permitem que a chave para a encriptação seja gerada deterministicamente da mensagem que se deseja encriptar, permitindo que dois usuários detentores de um mesmo arquivo, possam cifrar seus arquivos, de forma independente, com a mesma chave. Resultando, naturalmente, em cifras idênticas, esta característica desta primitiva é atrativa do ponto de vista da deduplicação, haja posto que para ser possível efetuar a deduplicação, é necessário efetuar comparações para eliminar *chunks* de dados redundantes.

2.5.1 Message-Locked Encryption (MLE)

As primitivas MLE são formalizações feitas por Bellare, Keelveedhi e Ristenpart (2013). Antes do trabalho de Bellare, Keelveedhi e Ristenpart (2013), Douceur et al. (2002) introduziram um protocolo de compartilhamento descentralizado que implementava a deduplicação segura. Douceur et al. (2002) utilizavam um artifício denominado "Encriptação Convergente"(CE). Na Encriptação Convergente, a chave é gerada deterministicamente a partir da própria mensagem (i.e., *hash* da mensagem) e utilizada para encriptar a própria mensagem. Uma mesma mensagem sempre gerará a mesma chave e, portanto, também gerará o mesmo texto cifrado. As formalizações feitas por Bellare, Keelveedhi e Ristenpart (2013) introduziram novas características que reforçam a segurança desta linha de primitivas. Desta forma, as primitivas MLE serão utilizadas neste trabalho pois fornecem as características necessárias para a deduplicação segura. Em seguida definições mais formais serão dadas.

Definições. Um algoritmo MLE é uma tupla de cinco algoritmos fundamentais: configuração ($\mathcal{P}$), geração de chave ($\mathcal{K}$), encriptação ($\mathcal{E}$), decriptação ($\mathcal{D}$) e geração de tag ($\mathcal{T}$) (BELLARE; KEELVEEDHI; RISTENPART, 2013). O algoritmo de configuração retorna o parâmetro público P. Utilizando P e a mensagem M, o algoritmo de geração de chave retorna a chave $K \leftarrow K_{\mathcal{P}}(M)$. O algoritmo de encriptação tem como valores de entrada: o parâmetro público P, a chave K e a mensagem M; e retorna o texto cifrado $C \leftarrow E_{\mathcal{P}}(K, M)$. Para recuperar a mensagem M; P, K e C são parâmetros de entrada do algoritmo D: $M \leftarrow D_{\mathcal{P}}(K, C)$. Há também o algoritmo de geração de *tag*. Uma *tag* é uma representação determinística do texto cifrado C. Ela é utilizada para a comparação da deduplicação. Uma tag é gerada através de: $T \leftarrow T_{\mathcal{P}}(C)$.

A primitiva MLE exige que a função $\mathcal{K}$ seja uma função de sentido único. A função $K : \{0,1\}^* \rightarrow \{0,1\}^*$ é de sentido único se: existe um algoritmo polinomial M_K que computa facilmente K, onde $M_K(m) = K(m)\ \forall m \in M$; e para cada algoritmo polinomial $\mathcal{A}$, a possibilidade de se *inverter* seja desprezível e muito difícil, ou seja: $Pr[Invert_{\mathcal{A},K}(n) = 1] \leq negl(n)$ (KATZ; LINDELL, 2014).

O algoritmo de encriptação simétrica com uma chave k, uma mensagem m e um texto-cifrado c, é um par de algoritmos probabilísticos (Enc, Dec), tal que $Enc : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^c$, $Dec : \{0,1\}^k \times \{0,1\}^c \rightarrow \{0,1\}^m$, e $\forall m \in M$: $\mathbb{P}[Dec(K, Enc(K, M)) = M] = 1$.

A corretude do algoritmo de decriptação do MLE requer que toda decriptação $D_{\mathcal{P}}(K, C) \rightarrow M$ seja possível para todos os valores de P, K, M e C. A função $\mathcal{T}$ também requer uma função de corretude, também chamada de taxa de falso negativo (BELLARE; KEELVEEDHI; RISTENPART, 2013). Se Alice faz *Upload* da cifra de M, ou seja, de C e Bob, posteriormente, faz o mesmo, corretude de tag significa dizer que as tags serão iguais

e o servidor armazenará apenas uma instância de C. Requisições de *Download* retornarão para ambos a cifra C e a corretude da decriptação garantirá que ambos possam decifrar C, sob suas respectivas chaves, para obterem M. Formalmente, corretude de tag significa dizer que: $[Tag(C') = Tag(C'')] \Leftrightarrow [M \leftarrow D(K, C') \ \wedge \ M \leftarrow D(K, C'')], \forall \{P, M, C, T\}^*$.

2.5.2 Encriptação Convergente (CE)

O algoritmo de Encriptação Convergente (CE) (DOUCEUR et al., 2002) é também um algoritmo do tipo MLE e portanto satisfaz uma importante necessidade deste trabalho: a deduplicação segura.

Formalmente, um algoritmo de Encriptação Convergente (CE) é composto pelos seguintes algoritmos: configuração ($\mathcal{P}$), geração de chave ($\mathcal{K}$), encriptação ($\mathcal{E}$) e decriptação ($\mathcal{D}$). Todos estes algoritmos seguem as características dos algoritmos que fazem parte das primitivas criptográficas MLE vistas anteriormente. O algoritmo de configuração inicia todas as variáveis públicas que podem ser utilizadas: *nonce*, *salt*, número de *rounds* etc. Para o algoritmo de geração da chave, recomenda-se o uso de uma função *hash* segura (e.g., SHA-3 (National Institute of Standards and Technology, 2015)), que satisfaça a condição de irreversibilidade (i.e., funções de sentido único (ver seção 2.1)). Portanto, para gerar uma chave k será necessário calcular o *hash* de m: $k \leftarrow K(m)$. Qualquer algoritmo seguro de criptografia simétrica poderá ser utilizado e a encriptação de uma mensagem m resultará em $c \leftarrow E(k, m)$. Para decifrar utiliza-se o algoritmo de decriptação, que retorna $m \leftarrow D(k, c)$. A tag é então gerada a partir do cálculo do *hash* de c: $t \leftarrow H(c)$ (DOUCEUR et al., 2002).

Tabela 2 – Encriptação Convergente (CE)

Configuração	Geração da Chave	Encriptação	Decriptação	Geração da Tag
$p \leftarrow P(\lambda)$	$k \leftarrow H(m)$	$c \leftarrow E(k, m)$	$m \leftarrow D(k, c)$	$t \leftarrow H(c)$

Fonte – Elaborado pelo autor

2.6 Encriptação Baseada em Atributos com a Política na Chave Privada (KP-ABE)

Nesta seção será introduzida a encriptação baseada em atributos (ABE). Ela permite que um usuário encripte arquivos de acordo com um conjunto de atributos específicos. Com o objetivo de que apenas usuários que possuam os atributos necessários possam decifrar um arquivo. Esta técnica pode ser utilizada, por exemplo, para limitar o acesso a determinadas informações de uma empresa de acordo com o cargo (i.e., atributo) do funcionário. Com a política na chave privada, esta técnica estabelece que a estrutura de

acesso se encontra na chave secreta dos usuários e um texto cifrado é computado de acordo com esses atributos.

2.6.1 Definição

Sahai e Waters (2005) introduziram o conceito de Encriptação Baseada em Atributos (ABE). Em um sistema ABE a chave secreta do usuário e o texto cifrado são marcadas por um conjunto de atributos descritivos, e somente a correspondência entre ambos permitirá o sucesso da decriptação. O criptosistema desenvolvido por Sahai e Waters (2005) permite a decriptação quando houver correspondência de pelo menos k atributos entre a chave secreta e o texto cifrado.

A Encriptação Baseada em Atributos com a Política na Chave Privada (KP-ABE) é uma primitiva criptográfica introduzida por Goyal et al. (2006) que permite o controle de acesso sobre dados encriptados. Isto é, usuários somente poderão descriptografar textos cifrados nos quais eles possuam os atributos necessários para isso. Como resultado, esta técnica permite um acesso controlado sobre quais usuários poderão descriptografar um certo arquivo. Além disso, contorna-se um aspecto da encriptação, no qual exige o compartilhamento de dados à "grosso modo"(i.e., dando a uma outra entidade sua chave secreta) (GOYAL et al., 2006); pois o protocolo KP-ABE permite que o acesso seja feito de forma não combinada, onde textos cifrados serão marcados com um conjunto de atributos e as chaves privadas serão associadas a esses atributos.

Cada chave do usuário é associada com uma estrutura de árvore de acesso, onde as folhas são associadas aos atributos. Um usuário é capaz de decifrar um texto cifrado se os atributos associados com o texto cifrado satisfizerem a estrutura de acesso da chave. Suponha que Alice tenha uma chave associada com a estrutura de acesso "X **AND** Y", e que Bob tenha uma chave associada a uma estrutura de acesso "Y **AND** Z", não é desejável que eles sejam capazes de decifrar um texto cifrado que precise somente do atributo Y ao combinarem para trapacear a estrutura necessária (GOYAL et al., 2006). Para evitar o conluio entre as partes envolvidas os autores fazem adaptações e generalizações do protocolo desenvolvido por Sahai e Waters (2005).

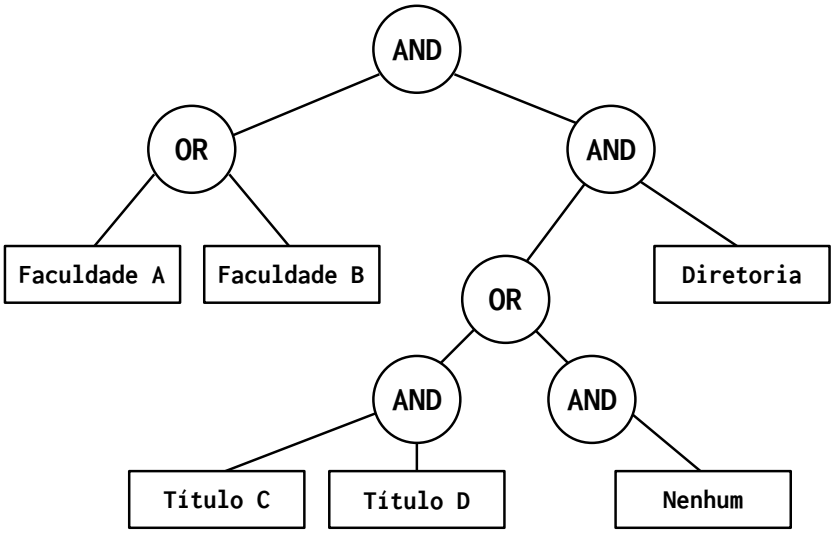
2.6.2 Estrutura de Acesso

Convencionalmente o controle de acesso consiste no uso de software para garantir que um usuário possa acessar um determinado dado se e somente se ele for autorizado a fazê-lo. Goyal et al. (2006) argumentam que tal característica não é satisfatória devido aos riscos causados por possíveis falhas e vulnerabilidades de software. Também é mencionado o caso de ataques originados de dentro ("insider attacks"), onde uma pessoa, que possui acesso ao servidor, rouba e distribui informações. Estruturas de acesso, no entanto, permitem

que sejam criadas hierarquias de acesso para dados protegidos (e.g., Figura 9). Ou seja, os dados são criptografados e poderão ser recuperados somente pelos usuários que possuem a chave correta de decifragem.

Goyal et al. (2006) implementam um esquema de acesso de controle fino, onde o acesso é feito através de uma árvore com todos os conjuntos autorizados de atributos. Cada chave do usuário é associada com uma árvore de acesso onde cada folha é associada com os atributos. Cada nó é associado à uma porta lógica (i.e., "**AND**", "**OR**").

Figura 9 – Diagrama de uma árvore de acesso de uma universidade.



Fonte – Elaborado pelo autor

Seja $\mathcal{T}$ uma árvore de acesso, cada nodo não-folha da árvore representa uma porta ("threshold gate"), descrita pelos filhos dela e com um valor, que deve ser satisfeito, para a passagem pela porta (GOYAL et al., 2006). Se num_x o número de filhos de um nó x e k_x o respectivo valor, então $0 < k_x \leq num_x$. Quando $k_x = 1$, a porta é um OR e quando é $k_x = num_x$ é uma porta AND. Cada nodo folha x da árvore é descrito por um atributo e um valor $k_x = 1$. Mais detalhes em (GOYAL et al., 2006).

2.6.3 Definições

Seja $P_1, P_2, ..., P_n$ um conjunto de partes. Uma coleção $\mathbb{A} \subseteq 2^{\{P_1,P_2,...,P_n\}}$ Uma estrutura de acesso é uma coleção $\mathbb{A}$ de subconjuntos não-vazios de $P_1, P_2, ..., P_n$. Os conjuntos em $\mathbb{A}$ são chamados de conjuntos autorizados, e os sets não em $\mathbb{A}$ são chamados de conjuntos não-autorizados. No trabalho de (GOYAL et al., 2006) o papel das partes envolvidas é responsabilidade dos atributos. Desta forma, a estrutura de acesso $\mathbb{A}$ conterá os conjuntos de atributos autorizados.

2.6.4 Algoritmos KP-ABE

Um esquema de Encriptação Baseado em Atributos com a Política na Chave Privada (KP-ABE) consiste em quatro algoritmos principais:

Configuração: Este algoritmo aleatório não recebe parâmetros de entrada, além do parâmetro de segurança implícito, e devolve os parâmetros públicos PK e uma chave mestre MK.

Encriptação: Neste algoritmo, também aleatório, é recebido como entrada a mensagem m e um conjunto de atributos γ, e os parâmetros públicos PK. Este algoritmo devolve o texto cifrado E.

Geração da Chave: Neste algoritmo aleatório é utilizado como entrada a estrutura de acesso $\mathbb{A}$, a chave mestre MK e os parâmetros públicos PK. Devolve-se a chave para decriptação D.

Decriptação: Este algoritmo recebe como entrada o texto cifrado E, que foi encriptado sob o conjunto de atributos γ, a chave D para decriptação que está associada a estrutura de controle de acesso $\mathbb{A}$ e os parâmetros públicos PK. O algoritmo devolve a mensagem m SE $\gamma \in \mathbb{A}$.

Mais detalhes sobre o funcionamento do protocolo KP-ABE pode ser encontrado em (GOYAL et al., 2006).

2.7 Caixas de Interesses

Nesta seção será introduzida a ideia de Caixas de Interesses (SILVA et al., 2014). Tal mecanismo permite a colaboração automática entre usuários desconhecidos baseando-se nos interesses desses usuários dentro de uma rede de compartilhamento.

2.7.1 Visão Geral

O esquema introduzido por Silva et al. (2014) permite que usuários de uma nuvem de armazenamento compartilhem arquivos com outros usuários, com base nos interesses similares. Cada usuário possui atributos que correspondem aos interesses dele na nuvem de armazenamento, e cada usuário pode criar um diretório especial (i.e., uma caixa de interesses) na nuvem, relacionada à esses interesses (i.e., atributos). Usuários com interesses similares possuem os mesmos atributos. A partir dos atributos das caixas de interesses, o protocolo identifica usuários com interesses similares e habilita o compartilhamento dos arquivos entre eles.

Alice é pesquisadora de uma universidade e faz parte de uma nuvem de dados, ela deseja colaborar com outros usuários acerca da pesquisa que ela está fazendo sobre o vírus

Ebola. Ela cria uma caixa de interesses nessa nuvem de armazenamento e coloca como atributos "ebola" e "pesquisa". Bob é também um usuário dessa nuvem de dados e também tem interesse em pesquisar sobre o vírus Ebola. Ele cria uma caixa de interesses sobre o tema e também coloca como atributo o termo "ebola". O sistema de armazenamento então avisa que existem outros usuários com o mesmo interesse que Bob, e pergunta se Bob deseja compartilhar as informações da caixa de interesses dele com outros pesquisadores. Se Bob aceitar, ele e Alice compartilharão do mesmo espaço e poderão trocar documentos entre eles (i.e., poderão colaborar). Dessa forma, Alice e Bob poderão desenvolver suas pesquisas de uma forma mais rica e produtiva, visto que contam agora com mais um colaborador para suas pesquisas.

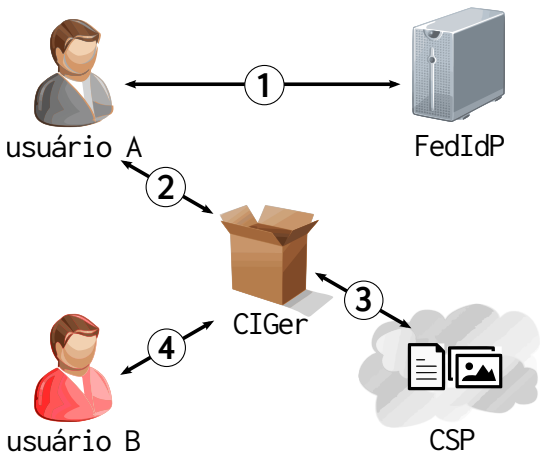
O protocolo considera três atores: um serviço de armazenamento em nuvem (*CSP*), um serviço de identidade federada (*FedIdP*), e um gerenciador de caixas de interesses(*CIGer*). O *CSP* é um serviço que armazena os arquivos e pastas dos usuários. O *CIGer* gerencia as caixas de interesses. Ele também relaciona os arquivos, armazenados no *CSP*, às caixas de interesses registradas, baseando-se nos interesses (i.e., atributos) comuns dos usuários com o auxílio de listas.

Um serviço de identidade federada (*FedIdP*) é um centro de distribuição de atributos. O conceito de federação de identidade descreve padrões e tecnologias que permitam a propagação de informações de identidades através de vários domínios seguros. Ela visa possibilitar que usuários possam utilizar serviços de outros domínios tendo como princípio de que estão autenticados (i.e., possuem identidade) em algum outro domínio parceiro, evitando a necessidade de se reautenticar (IBM DEVELOPERWORKS, 2010) e ao mesmo tempo permitir que as identidades sejam confirmadas como verdadeiras. A comunicação entre essas organizações no protocolo de Silva et al. (2014) é feita a partir do uso do *framework Security Assertion Markup Language* (SAML). O SAML é um formato de dados baseado no XML que objetiva trocar dados de autenticação e autorização entre partes (OASIS, 2008). O requisito mais importante estabelecido pelo SAML é a possibilidade de login único para todas as aplicações (*web browser single sign-on*), característica útil adotada pelo protocolo de Silva et al. (2014), visto que o mesmo é constituído de vários serviços separados conectados via federação de identidades (i.e., *CSP, FedIdP, CIGer*).

Além disso, o protocolo de Caixas de Interesses introduzido por Silva et al. (2014) ainda conta com algumas etapas de funcionamento. A primeira delas é o registro do usuário; seguido da criação da caixa de interesses dele; depois o armazenamento de um arquivo nessa caixa de interesses recém-criada; e finalmente o compartilhamento automático, a partir da identificação pelo sistema de que dois usuários possuem interesses semelhantes.

Na Figura 10, que ilustra o funcionamento do protocolo de caixas de interesses, a autoridade *FedIdP* registra os atributos do usuário A (etapa 1). O usuário A interage com o *CIGer* para criar uma caixa de interesses e armazenar os arquivos dele (etapa 2). O

Figura 10 – Visão geral do protocolo de Caixas de Interesses.



Fonte – Elaborado pelo autor

CIGer armazena os arquivos do usuário A no *CSP* (etapa 3). Finalmente, usuários A e B poderão compartilhar arquivos através do *CIGer* (etapa 4).

2.7.2 Etapas de Funcionamento

A seguir serão detalhadas as etapas de funcionamento do protocolo original de Caixas de Interesses, introduzido por Silva et al. (2014).

I. Autenticação e Obtenção dos Atributos. A primeira etapa do protocolo de caixas de interesses consiste na autenticação e obtenção de atributos. O serviço de identidade federada permite que o usuário autentique-se para em seguida obter os atributos dele na federação. A autenticação ocorre de acordo com o framework SAML. Detalhes do protocolo são demonstrados no algoritmo a seguir.

Algoritmo 1. Etapa de Autenticação e Obtenção de Atributos:

1: $u_i \rightarrow CSP$
2: $CSP \xrightarrow{u_i} FedIDP$
3: $FedIDP \xrightarrow{Att,ID} u_i$
4: $u_i \xrightarrow{Att,ID} CSP$

O usuário u_i primeiramente solicita o acesso ao serviço de armazenamento de dados em nuvem *CSP*. A nuvem então o redireciona para um servidor gerenciador de identidades para que ele se autentique e receba permissão de acesso ao serviço de armazenamento. Após o sucesso na autenticação, o usuário recebe a credencial e os atributos dele (passo 3).

Em seguida, u_i utiliza essas informações recebidas para acessar o espaço de armazenamento dele no *CSP*.

II. Criação das Caixas de Interesses. Após a primeira etapa o usuário já possui seus atributos e já está autenticado. Nesta segunda etapa ele define os metadados de um diretório especial que será a caixa de interesses. O usuário poderá, em seguida, enviar arquivos para a caixa de interesses dele. O algoritmo (2.7.2) a seguir demonstra esse passo.

Algoritmo 2. Etapa de Criação de Caixas de Interesses:

1: $u_i \xrightarrow{IB_{u_i}} CSP$
2: $CSP \xrightarrow{u_i, IB_{u_i}, att} CIGer$
3: $u_i \xrightarrow{H(cf_i), H(cf_{ij})} CSP$
4: CSP computa: $Dedup(cf_{ij}), \forall j, 0 \leq j \leq k$;
5: $CSP \xrightarrow{H(cf_i), H(cf_{ij})} CIGer$

Esta etapa começa a partir da criação da caixa de interesses IB_{u_i} na nuvem pelo usuário u_i. No passo de criação da caixa de interesses, o usuário deve definir o nome, a data de criação e uma descrição dessa caixa de interesses. Tais atributos serão os metadados gerais da caixa de interesses. Adicionalmente, os atributos específicos serão aqueles que foram recebidos pela entidade *Att* de atributos da federação de identidades (2.7.2). O *CSP* então informa para o *CIGer* que o usuário u_i criou uma caixa de interesses IB_{u_i}. O gerenciador de caixas de interesses, i.e *CIGer*, registra esses dados em sua lista de caixas de interesses criadas.

Após a finalização da configuração da caixa de interesses no passo anterior, o usuário pode seguir com o envio de arquivos para a mesma, como um mecanismo tradicional de armazenamento em nuvem. Todavia, como o protocolo conta com um mecanismo de deduplicação de dados, os dados são primeiramente verificados se possuem uma cópia no serviço de armazenamento para evitar o armazenamento de dados duplicados. A verificação de deduplicação ocorre a partir do cálculo do *hash* do arquivo e do *hash* de cada pedaço fragmentado. São respectivamente $H(cf_i)$ e $H(cf_{ij})$, os *hash* enviados para o *CSP* para serem verificados.

O *CSP* então verifica na sua lista de arquivos se os valores de *hash* recebidos já se encontram armazenados e então, caso já estejam armazenados, o usuário não precisa seguir com o envio do arquivo. O *CSP* registra os valores de *hash* recebidos na caixa de interesses do usuário, tal registro funciona como um link que significa que o arquivo, previamente armazenado, possui um outro usuário. Assim ocorre a deduplicação.

Caso a deduplicação não ocorra, o que significa que este arquivo é novo no serviço de armazenamento, o arquivo é armazenado nessa caixa de interesses e os valores de *hash*

calculados na etapa anterior são registrados pelo *CSP*. No entanto, caso a deduplicação ocorra, a próxima etapa é iniciada.

III. Identificação de Usuários. Nesta penúltima etapa o *CIGer* e o *CSP* atuam em conjunto, com o objetivo de identificar um arquivo que pertença a duas caixas de interesses, de usuários diferentes, mas com interesses semelhantes, i.e., atributos iguais. Se o *CSP* encontrar todos os valores *hash* $H(cf_{ij}), \forall j, 0 \leq j \leq k$ ele segue com a execução do seguinte algoritmo (SILVA et al., 2014).

Algoritmo 3. Etapa de Identificação de Usuários:

1: *CIGer* procura por caixas de interesses que possuam o arquivo com *hash* $H(cf_i)$ associado e atributos iguais.
2: $CIGer \xrightarrow{CIcomp} CSP$

Como detalhado acima, é possível identificar caixas de interesses com *atributos iguais* e pelo menos *um arquivo em comum* a partir das informações registradas nas etapas anteriores, como por exemplo, o registro das caixas de interesses no *CIGer* e o registro de arquivos no *CSP*. Essas duas características, i.e., um arquivo em comum e atributos similares, são compulsórias no protocolo introduzido por Silva et al. (2014).

IV. Compartilhamento de Dados. A última etapa do protocolo consiste no compartilhamento dos arquivos. Esta etapa se dá a partir da identificação de caixas de interesses com arquivos em comum. Após a notificação dada ao usuário e seguido o aceite do mesmo, o compartilhamento é então efetuado e os dados de uma caixa de interesses são compartilhados com a(s) outra(s) caixa(s) de interesses. O seguinte algoritmo ilustra esta etapa.

Algoritmo 4. Etapa de Compartilhamento de Dados:

1: $CSP \xrightarrow{CIcomp} u_i$
2: $u_i \xrightarrow{CIselec} CSP$
3: $CSP \xrightarrow{CIselec} CIGer$
4: $CIGer \xrightarrow{HComp} CSP$
5: *CSP* obtém para cada elemento de *HComp*
cf_{ij} a partir de cada $H(cf_{ij}) \forall i, 1 \leq i \leq n$
cf_i a partir de cada $cf_{ij}, \forall i, 1 \leq i \leq n$
6: $CSP \xrightarrow{Fcomp} u_i$

No primeiro passo (passo 1) desta última etapa, a lista de caixas de interesses encontrada na etapa anterior (2.7.2), onde fora encontrado um arquivo em comum conforme

na etapa (2.7.2), gerará uma notificação que será enviada para o aceite do usuário. Desta forma o usuário poderá aceitar apenas as caixas de interesses que desejar. No segundo passo é demonstrada a ação de envio para a nuvem da lista de caixas aceitas pelo usuário u_i.

No terceiro passo o *CSP* envia para o *CIGer* as caixas de interesses selecionadas pelo usuário u_i. No passo seguinte, o *CIGer* retorna os valores dos hashes dos arquivos pertencentes as caixas de interesses recebidas no passo anterior. No passo 5 o *CSP* encontra o(s) arquivo(s) cf_i que será(ão) disponibilizado(s) ao(s) usuário(s) ao encontrar os fragmentos de cf_i, a partir dos valores *hash* $H(cf_{ij})$. No final desta etapa, no passo 6, o *CSP* disponibiliza os arquivos das caixas de interesses aceitadas pelo usuário u_i. Antes deste último passo, a nuvem verifica quais arquivos o usuário ainda não possui acesso pela caixa de interesses dele, desta forma a nuvem envia apenas os arquivos que ainda não estão armazenados nessa caixa de interesses (SILVA et al., 2014).

2.7.3 Considerações sobre o Protocolo de Caixas de Interesses

O protocolo de caixa de interesses permite o fácil compartilhamento baseado nos interesses similares dos usuários. Todavia, o protocolo de Silva et al. (2014) não implementa medidas de segurança para proteger os dados de usuários que estão armazenados nessas nuvens de armazenamento, isto é, os dados armazenados em texto claro fornecem uma potencial ameaça para a confidencialidade dos usuários.

Dado que o *CSP* originalmente armazena os dados em *texto-claro*, um adversário com acesso ao espaço de armazenamento (e.g., um intruso, um atacante interno etc), pode facilmente modificar ou recuperar qualquer arquivo armazenado. Atacantes internos podem analisar os dados armazenados. Um Gerenciador de Caixas de Interesses (CIGer) malicioso pode ler os dados armazenados, visto que os dados são armazenados em texto claro. Se Alice armazena um arquivo em uma caixa de interesses, Eve (um adversário) tentará ler o arquivo armazenado por Alice. Dado que o arquivo foi armazenado em texto claro, Eve obterá sucesso ao acessar os arquivos armazenados por Alice.

Também, o esquema não protege os atributos dos usuários. O *CIGer* ou um atacante interno podem facilmente lê-los e portanto descobrir os interesses dos usuários. Um atacante pode traçar as tabelas de interesses e descobrir os interesses dos usuários. Tal ação pode ser indesejável se o usuário preferir manter a confidencialidade dos interesses dele.

Apesar de que a encriptação de arquivos e dos atributos poderia prevenir que um adversário lesse os arquivos e os atributos, ela não é possível de ser implementada sem modificar alguns aspectos do protocolo original. Primeiramente, o *CIGer* precisa conhecer os atributos para poder identificar os interesses similares entre usuários. E como observado na seção 2.2, a encriptação simétrica possui a propriedade de uma função pseudo-aleatória

e portanto possui como saída um texto que nada se parece com a mensagem inicial. Se os textos cifrados são diferentes para uma mesma mensagem encriptada sob chaves diferentes, então a deduplicação não pode ser efetuada, pois a deduplicação exige que os blocos de dados sejam idênticos. Desta forma, um usuário não poderia ler um arquivo encriptado por outro usuário sem possuir a chave secreta correspondente.

2.8 Conclusão

Apesar dos avanços conquistados pela proposta de Silva et al. (2014), é necessário ainda implementar adaptações que garantam que os dados armazenados nas caixas de interesses estejam seguros e que, ainda assim, os usuários possam compartilhar arquivos entre eles de forma automática. A seção de considerações sobre o protocolo de caixas de interesses (seção 2.7.3) demonstra a necessidade de adequar o protocolo para a implementação em um ambiente real e potencialmente inseguro. As ferramentas introduzidas neste capítulo servirão como fundamentação teórica para se alcançar os objetivos deste trabalho. Essencialmente, a criptografia simétrica (seção 2.2) substituirá o armazenamento de arquivos em texto-claro, e a encriptação baseada em atributos (seção 2.6) permitirá a combinação do uso de atributos (i.e., interesses) com a encriptação dos dados. No próximo capítulo será finalmente introduzido o novo protocolo seguro proposto por este trabalho, e que utiliza extensamente as técnicas criptográficas estudadas neste capítulo.

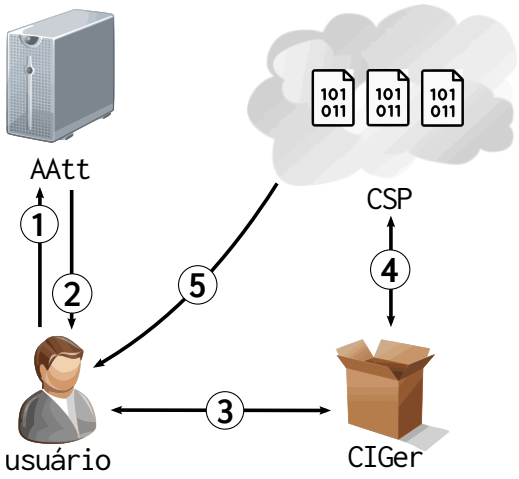
3 Protocolo Colaborativo e Seguro para Nuvens de Armazenamento

O protocolo original de caixas de interesses proposto anteriormente (ver Seção 2.7) permite que usuários compartilhem automaticamente dados a partir de seus interesses. Todavia, como descrito, esse protocolo não possui mecanismos que garantam a segurança dos dados armazenados. Em seguida será introduzido um novo protocolo, que visa superar as falhas de segurança observadas previamente.

3.1 Visão Geral

O protocolo que será apresentado, visa segurar os dados armazenados em nuvens de armazenamento, enquanto mantém as características-chave do protocolo original de caixas de interesses, isto é: o uso de atributos (i.e., interesses) para encontrar potenciais colaboradores, e a deduplicação de dados.

Figura 11 – Visão geral do novo protocolo.



Fonte – Elaborado pelo autor

Na Figura 11, a autoridade *AAtt* emite uma chave ABE secreta para o usuário (etapas 1 e 2). Um usuário interage com o *CIGer* para criar uma caixa de interesses e armazenar os arquivos encriptados dele (etapa 3). O *CIGer* armazena os arquivos encriptados no *CSP* (etapa 4). Finalmente, o usuário acessa o *CSP* para recuperar os arquivos dele (etapa 5).

O novo protocolo utiliza um serviço de armazenamento em nuvem (*CSP*). Usuários armazenam e compartilham seus arquivos de modo convencional. Todavia, eles também podem compartilhar seus arquivos utilizando diretórios especiais, i.e., caixas de interesses, para compartilhamento na nuvem. Para isso, o novo protocolo estende a nuvem tradicional adicionando dois atores: uma autoridade de atributos (*AAtt*) e um gerenciador de caixas de interesses (*CIGer*). A autoridade de atributos é responsável pela emissão de atributos e chaves para os usuários. O gerenciador de caixas de interesses controla as caixas de interesses. Ele registra todas as relações de posse entre caixas de interesses, arquivos e usuários. Para isso, ele mantém uma lista de caixas de interesses e uma lista de arquivos.

O esquema começa com uma fase de *configuração*. Nesta fase a autoridade de atributos estabelece um conjuntos de atributos e executa um algoritmo KP-ABE (ver capítulo 2.6) para obter os parâmetros públicos do ABE. Ele também gera um par de chaves de um criptosistema de chave pública. Mais tarde, a *AAtt* gera uma chave secreta ABE para cada usuário marcada com os atributos dele e os parâmetros públicos correspondentes do algoritmo ABE. Usuários com interesses similares possuem chaves secretas com os mesmos atributos. Adicionalmente, a *AAtt* calcula o *hash* dos atributos do usuário em conjunto com um número aleatório secreto para segurá-lo, assina o *hash* resultante, e entrega esta assinatura para o usuário. Usuários com os mesmos atributos terão o mesmo *hash* resultante.

O usuário agora é capaz de criar sua caixa de interesses. Para isso, ele se conecta ao *CIGer*. Depois, ele cria sua caixa de interesses e a configura preenchendo o nome dela (e.g., saúde) e uma descrição (e.g., "pesquisa sobre saúde"). Então, ele envia essas informações, assim como a assinatura recebida da *AAtt*, do *CIGer*. Depois de receber essas informações, o *CIGer* registra em sua lista de caixas de interesses e cria um diretório correspondente na nuvem.

Depois de criar a caixa de interesses, o usuário poderá compartilhar dados enviando arquivos para a caixa de interesses dele, e após isso outras etapas do protocolo serão acionadas para habilitarem automaticamente a instância colaborativa deste protocolo. Para armazenar um arquivo, ele primeiramente encripta o aquivo utilizando um algoritmo de encriptação convergente e então encripta a chave do arquivo, gerada na encriptação convergente, com o algoritmo ABE a partir dos parâmetros públicos recebidos nas primeiras etapas do protocolo. Caso este arquivo não esteja armazenado na nuvem, o usuário envia para o *CIGer* as cifras geradas. O *CIGer* armazena esta informação na sua lista de arquivos e guarda o arquivo na nuvem. Mais detalhes serão fornecidos na seção 3.3 de apresentação de detalhes do novo protocolo.

Todos os arquivos armazenados em uma caixa de interesses estão encriptados. Para recuperar um arquivo de uma caixa de interesses, o usuário primeiramente se conecta ao *CIGer* e então informa qual arquivo encriptado ele deseja obter. Então, o *CIGer* retorna

para ele a cifra correspondente a chave de encriptação convergente do arquivo. Depois de receber esta cifra, o usuário decifra utilizando a chave ABE secreta e obtém a chave convergente do arquivo. O usuário agora recebe a cifra do arquivo da caixa de interesses dele e então ele a decifra utilizando a chave do arquivo.

O *CIGer* identifica usuários que possuam os mesmos interesses comparando os *hashes* assinados na sua lista de caixas de interesses. A correspondência entre os *hashes* de dois usuários os classificam como possuidores dos mesmos interesses. Quando identificados os interesses similares, o *CIGer* compartilha (i.e., cria um link) os arquivos desses usuários através das duas caixas de interesses.

3.2 Requisitos, Premissas do Protocolo, Modelo de Adversário, Participantes e Notação

Nesta seção serão apresentados os requerimentos, premissas, modelo de adversário e a notação, assim como os participantes do novo protocolo.

3.2.1 Requisitos

O novo protocolo aqui proposto utiliza duas técnicas-chave criptográficas. A primeira é o algoritmo de encriptação convergente (CE) (ver capítulo 2.5.2). Esta primitiva garante a deduplicação segura de arquivos. A segunda é a encriptação baseada em atributos com a política na chave privada (KP-ABE) (capítulo 2.6). Ela é usada para prover chaves para usuários a partir de um conjunto de atributos, e executa a cifragem e decifragem de acordo com esses atributos. Adicionalmente, o protocolo aqui apresentado também requer um algoritmo de *hash* seguro, tal como o SHA-3 (National Institute of Standards and Technology, 2015) e também um algoritmo seguro de chave pública, como o RSA OAEP (BELLARE; ROGAWAY, 1994).

3.2.2 Premissas do Protocolo

Todos os canais de comunicação são seguros e autenticados entre as partes envolvidas. Uma primitiva criptográfica segura KP-ABE é utilizada. Usuários a utilizam para obter as suas chaves correspondentes aos atributos deles. Estes atributos são relacionados aos interesses desses usuários e tais atributos são utilizados nas caixas de interesses. A *AAtt* é uma autoridade confiável. Ela age com precisão recebendo e respondendo requisições dos usuários, e registra as informações em suas listas. A *AAtt* e o *CIGer* não colidem. O *CIGer* não pode obter um par de chaves KP-ABE e também não confabula contra usuários.

3.2.3 Modelo de Adversário

O objetivo do protocolo é evitar ataques de agentes maliciosos, como por exemplo do *CIGer*. O agente malicioso seria capaz de quebrar a privacidade dos usuários das caixas de interesses ao ler os arquivos desses usuários. Em outras palavras, esse adversário pode ler o conteúdo dos arquivos armazenados em caixas de interesses.

3.2.4 Participantes e Notação

O protocolo leva em consideração quatro atores: um usuário u_i, um gerenciador de caixas de interesses (*CIGer*), uma autoridade de atributos (*AAtt*) e um serviço de armazenamento em nuvem (*CSP*). Seja $u_i \in U$ um usuário parte de um conjunto de usuários U e att_{u_i} sejam os atributos do usuário u_i. Seja CI_{u_i} a caixa de interesses do usuário u_i e *gmdata* os atributos gerais de CI_{u_i}. O usuário cria a caixa de interesses CI_{u_i} e compartilha arquivos através dela. O gerenciador de caixas de interesses (*CIGer*) recebe as requisições relacionadas a sua caixa de interesses e as responde. Isto é, ele cria esses diretórios e armazena os arquivos dos usuários na nuvem; ele também registra todas as relações de posse entre caixas de interesses e usuários. Para isso, ele mantém uma lista L_{CI} contendo as informações de caixas de interesses e seus donos. O *CIGer* também mantém uma lista L_{Files} com informações sobre os arquivos que pertencem a caixas de interesses e seus usuários.

A autoridade de atributos (*AAtt*) age como uma autoridade de atributos ABE e emite pares de chave para seus usuários. Adicionalmente, ela emite atributos para os usuários. Sejam PK_{ABE} os parâmetros públicos de um algoritmo ABE e D_{ABE} a chave para decifragem do usuário u_i que refere-se ao conjunto de atributos deste usuário. A *AAtt* também possui um par de chaves de um criptosistema seguro de chave pública. Sejam PK_{AAtt} e SK_{AAtt} a chave pública e a chave secreta, respectivamente. O serviço de armazenamento em nuvem armazena arquivos de forma convencional. Ele também executa requisições recebidas do *CIGer*.

Seja $H(.)$ uma função *hash* segura e seja $H(m)$ o *hash* resultante de uma mensagem m. Seja $H(m, r)$ o *hash* resultante de uma mensagem m e um número aleatório secreto r. Seja $E_k(m)$ a cifra da mensagem m com uma chave k e $D(k, E_k(m))$ a decifragem de $E_k(m)$ usando uma chave k. $T(E_{k(m)})$ é a *tag* da cifra $E_k(m)$.

3.3 O Protocolo

Nesta seção o novo protocolo será detalhado. Foi notado que o *CIGer* pode fazer parte do serviço de armazenamento em nuvem ou não. Na descrição que segue, ele será colocado separadamente para facilitar o entendimento do protocolo. Também, será focado

no funcionamento do gerenciador de caixas de interesses, e será abstraído como ele interage com a nuvem.

3.3.1 Fase de Configuração

O esquema começa com uma fase de configuração. Nesta fase a *AAtt* define o conjunto de atributos públicos e executa o algoritmo de configuração do protocolo KP-ABE. Como resultado, ele retorna uma chave pública PK_{ABE} correspondente aos atributos e uma chave secreta de decifragem D_{ABE} que será gerada para os usuários mais tarde. A *AAtt* também gera um par de chaves de um criptosistema de chave pública, isto é, uma chave pública PK_{AAtt} e uma chave secreta SK_{AAtt}. Finalmente, a *AAtt* gera um número aleatório grande r.

Depois da fase de configuração, a *AAtt* executa o passo de geração de chave do algoritmo ABE e gera uma chave de decifragem $D_{ABE_{u_i}}$. Todos os textos cifrados gerados mais tarde, usando a chave PK_{ABE}, serão marcados com atributos e podem ser decifrados somente com as chaves correspondentes a chave pública PK_{ABE}. Adicionalmente, a *AAtt* provê ao usuário um *hash* assinado correspondente aos atributos dele. Isto é, a *AAtt* calcula o *hash* dos atributos att_{u_i} juntamente com o número aleatório r e obtém o *hash* resultante: $h = H(att_{u_i}, r)$. Depois disso, ela encripta o *hash* h usando a chave secreta SK_{AAtt} e obtém a cifra $E_{SK_{AAtt}}(H(att_{u_i}, r))$. Esta última cifra é considerada como uma assinatura especial do *hash* da assinatura calculado com um fator aleatório r. A *AAtt* envia para o usuário a cifra $E_{SK_{AAtt}}(H(att_{u_i}, r))$, a chave secreta SK_{u_i}, e a chave PK_{AAtt} da *AAtt*.

Depois de obter sua chave secreta D_{ABE}, e a assinatura correspondente aos atributos dele $E_{SK_{AAtt}}(H(att_{u_i}, r))$, o usuário pode então configurar uma nova caixa de interesses. A seguir, serão apresentados os passos necessários para a criação de uma caixa de interesses e para o armazenamento de dados nela. Serão diferenciados dois casos. No primeiro caso, o usuário envia para a sua caixa de interesses um arquivo novo na nuvem de armazenamento; no segundo caso, o arquivo já existe armazenado por outro usuário. Esses passos são apresentados em seguida.

3.3.2 Criação de uma Caixa de Interesses

Para criar uma caixa de interesses CI_{u_i} o usuário u_i primeiramente se autentica com o gerenciador de caixas de interesses (*CIGer*). Depois disso, ele especifica os metadados gerais *gmdata* ao colocar o nome da caixa de interesses (e.g., "saúde") e uma descrição (e.g., "pesquisa sobre saúde"). O usuário então envia os dados para o *CIGer* uma tupla contendo: $\langle gmdata, E_{SK_{AAtt}}(H(att_{u_i}, r), PK_{AAtt})\rangle$. O último elemento da tupla é a assinatura dos atributos. Depois de receber esta tupla, o *CIGer* usa a chave pública PK_{AAtt} para verificar a assinatura $E_{SK_{AAtt}}(H(att_{u_i}, r)$ e obtém $H(att_{u_i}, r)$. Então, ele adiciona uma nova entrada

na lista L_{CI} contendo $\langle u_i, CI_{u_i}, gmdata,$
$E_{SK_{AAtt}}(H(att_{u_i}, r), (H(att_{u_i}, r)\rangle$ e cria uma caixa de interesses CI_{u_i} no serviço de armazenamento em nuvem (*CSP*). O *hash* ($H(att_{u_i}, r)$ são os metadados específicos de CI_{u_i}. O algoritmo 1 sumariza esses passos.

Algoritmo 1. Criação de uma caixa de interesse:

1: $User \rightarrow CIGer : \langle gmdata, E_{SK_{AAtt}}(H(att_{u_i}, r), PK_{AAtt})\rangle$
2: $CIGer : H(att_{u_i}, r) \leftarrow D(PK_{AAtt}, E_{SK_{AAtt}}(H(att_{u_i}, r)))$
3: $L_{CI} \leftarrow \langle u_i, CI_{u_i}, gmdata, E_{SK_{AAtt}}(H(att_{u_i}, r)), H(att_{u_i}, r)\rangle$

3.3.3 Armazenamento de um arquivo

Antes de armazenar um arquivo m na sua caixa de interesses CI_{u_i}, o usuário u_i computa a chave de encriptação convergente K_m de m. Isto é, ele calcula o *hash* de m para obter a chave K_m. Depois disso, o usuário u_i, cifra o seu arquivo m usando a chave K_m. Resultando em um texto cifrado $E_{K_m}(m)$. Ele então computa a *tag* $T(E_{K_m}(m))$ ao calcular o *hash* de $E_{K_m}(m)$. Após, o usuário encripta a chave K_m com os parâmetros públicos do protocolo ABE, ou seja PK_{ABE} e obtém a cifra $E_{PK_{ABE}}(K_m)$ (Algoritmo 2).

Algoritmo 2. Armazenando um arquivo na caixa de interesses CI_{u_i}:

1: *User:*
2: $K_m \leftarrow H(m)$
3: $E_{K_m}(m) \leftarrow K_m, m$
4: $T(E_{K_m}(m)) \leftarrow H(E_{K_m}(m))$
5: $E_{PK_{ABE}}(K_m) \leftarrow E_{ABE}(PK_{ABE}, K_m)$

O usuário então envia a *tag* $T(E_{K_m}(m))$ para o *CIGer*. Depois disso, o *CIGer* verifica se já existe uma entrada desta *tag* na lista L_{Files}.

Caso esta *tag* não seja encontrada, significa então que o arquivo ainda não foi armazenado na nuvem. O *CIGer* então informa ao usuário que a *tag* $T(E_{K_m}(m))$ não foi encontrada enviando uma mensagem contendo *false*. Depois de receber essa mensagem, o usuário envia para o *CIGer* a tupla $\langle T(E_{E_{K_m}}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m)\rangle$. Depois de receber esta tupla, o *CIGer* adiciona na lista L_{Files} uma entrada contendo: $CI_{u_i}, T(E_{K_m}(m))$, $E_{K_m}(m), E_{PK_{ABE}}(K_m), u_i$. O *CIGer* então armazena o arquivo $E_{K_m}(m)$ na nuvem.

Se a *tag* $T(E_{K_m}(m))$ já existir na lista L_{Files}, o *CIGer* envia para o usuário uma mensagem contendo *true*. O usuário, portanto, não precisa enviar a cifra $E_{K_m}(m)$ para a caixa de interesses CI_{u_i}. Ao invés, o *CIGer* apenas atualiza na lista L_{Files} um link deste arquivo para a caixa de interesses criada por esse novo usuário u_i. Portanto, o arquivo é acessível por todas as caixas de interesses nas quais ele foi enviado. Assim, o *CIGer* adiciona adiciona a seguinte entrada: $CI_{u_2}, T(E_{K_m}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m), u_2$. Especificando que este arquivo tem um outro novo usuário u_2 e é também armazenado em outra caixa de interesses, CI_{u_2}. O Algoritmo 3 sumariza esses passos.

Algoritmo 3. Checando uma Tag:

User: $T(E_{K_m}(m)) \rightarrow$ *CIGer*
if "*false*" **then**
  $User \rightarrow CIGer : \langle T(E_{K_m}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m)\rangle$
  $CIGer{:}L_{Files} \leftarrow (CI_{u_i}, T(E_{K_m}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m), u_i)$
  $CIGer \leftarrow E_{K_m}(m)$
else if "*true*" **then**
  $CIGer{:}L_{Files} \leftarrow CI_{u_2}, T(E_{K_m}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m), u_2$
end if

3.3.4 Recuperação de um arquivo

O usuário u_i compartilhou arquivos através da caixa de interesses CI_{u_i}. Todos os arquivos nessa caixa de interesses estão cifrados. Para recuperar um arquivo m, o usuário primeiramente acessa a caixa de interesses CI_{u_i} e do *CSP* ele obtém o arquivo $E_{k_m})(m)$. Após isso, ele se conecta ao *CIGer* e informa o arquivo que ele quer decifrar. A nuvem então envia para ele o texto cifrado $E_{ABE}(K_m)$ que contém a chave para decifrar $E_{k_m}(m)$. Depois de receber a cifra da chave, o usuário utiliza a chave secreta ABE $D_{ABE_{u_i}}$ dele para decifrá-la. Depois de decifrar a chave K_m do arquivo, o usuário utiliza a chave K_m para decifrar o texto cifrado $E_{K_m}(m)$ e obter o arquivo m. Esta parte do protocolo é demonstrada no Algoritmo 4.

Algoritmo 4. Recuperando um arquivo de uma caixa de interesse:

CSP: $u_i \leftarrow E_{K_m}(m)$
CIGer: $u_i \leftarrow E_{ABE}(K_m)$
User: $K_m \leftarrow D_{ABE}(D_{ABE_{u_i}}, E_{ABE}(K_m))$
  $m \leftarrow D(K_m, E_{K_m}(m))$

3.3.5 Compartilhamento Automático

O *CIGer* mantém a lista L_{CI}. Esta lista contém o *hash* dos atributos de todos os usuários, i.e., $H(att_{u_i}, r)$. Ao comparar esses *hashes*, o *CIGer* é capaz de compartilhar arquivos entre usuários com interesses similares. Isto é, suponha que um usuário u_1 tenha uma entrada na lista L_{Files} contendo $\langle CI_{u_1}, T(E_{K_m}(m)), E_{K_m}, E_{PK_{ABE}}(K_m)\rangle$, ao comparar o *hash* do usuário u_1 com o *hash* de outro usuário u_2, o *CIGer* identifica que ambos possuem interesses semelhantes e então atualiza a lista L_{Files} para $\langle [CI_{u_1}, CI_{u_2}], T(E_{K_m}(m)), E_{K_m}(m), E_{PK_{ABE}}(K_m)\rangle$. Este procedimento é executado sem a necessidade da ação do usuário, portanto é automático.

3.4 Conclusão

Neste capítulo foi introduzido o novo protocolo colaborativo e seguro para o compartilhamento de dados, visando a aplicação em nuvens de armazenamento. Essa nova construção supera o grau de segurança do protocolo original de Silva et al. (2014), pois visa a aplicação do novo protocolo em ambientes reais e potencialmente inseguros, diferentemente do anterior, dado que suas etapas de funcionamento não utilizavam métodos que garantissem a segurança dos dados armazenados ou o sigilo das informações dos usuários. Desta forma, o novo protocolo supera o anterior ao adotar técnicas criptográficas seguras, como *hash* criptográfico, criptografias simétrica e assimétrica etc, como visto nas etapas do novo protocolo, e anteriormente introduzidas no capítulo de embasamento teórico (Capítulo 2). No próximo capítulo será efetuada uma análise sobre o novo protocolo.

4 Análise do novo protocolo

Neste capítulo será discutido o novo protocolo proposto. Primeiramente serão abordados os requisitos e as limitações do novo protocolo. Após, considerações sobre a segurança semântica, e logo depois o comportamento do novo protocolo sob alguns principais ataques.

4.1 Considerações sobre o novo protocolo

O protocolo apresentado considera que o *CIGer* não possa quebrar a privacidade dos usuários ao ler seus arquivos. Suponha que o *CIGer* tenha a seguinte entrada em sua lista: $IB_{u_i}, T(E_{K_m}), E_{K_m}(m), E_{PK_{ABE}}(K_m)$. Para decifrar $E_{K_m}(m)$, o *CIGer* primeiramente precisa decifrar $E_{PK_{ABE}}(K_m)$ para obter a chave secreta K_m correspondente a encriptação convergente. Enquanto o *CIGer* não possuir SK_{ABE} marcado com os atributos para decifrar $E_{PK_{ABE}}(K_m)$, ele não pode obter a chave secreta K_m. No entanto, porque a chave secreta K_m é gerada pelo cálculo do *hash* do arquivo m, o *CIGer* pode calcular o *hash* de arquivos até ele encontrar uma chave que decifre $E_{K_m}(m)$ (i.e., força bruta). Porém, seria difícil encontrar uma chave correspondente a um arquivo de conteúdo desconhecido. Esta vulnerabilidade é referente ao protocolo de encriptação convergente.

No esquema apresentado, foi empregado o uso do protocolo CE para permitir a deduplicação de arquivos. Isto é, pela checagem se a tag $T(E_{K_m}(m))$ já existe na lista L_{Files}, o *CIGer* sabe se esse arquivo já foi armazenado ou não. Essa característica salva tanto espaço em disco como banda de rede, visto que o usuário não precisará enviar um arquivo que já fora armazenado. Adicionalmente, para um não-usuário de uma caixa de interesse, o protocolo CE manteria a deduplicação segura global no serviço de armazenamento em nuvem. Isto é, ao invés de usa o algoritmo KP-ABE para cifrar as chaves CE dos arquivos, um não usuário de caixa de interesse é capaz de cifrar o arquivo dele usando a chave K_m. Ele então salva K_m (e.g., no computador dele) e armazena o arquivo cifrado na nuvem como usualmente. Enquanto verifica as tags, caso do *CIGer* encontre uma tag de um arquivo já armazenado em nuvem, o *CIGer* não precisa armazenar este arquivo novamente. Uma desvantagem desta abordagem refere-se ao fato de que usuários de caixas de interesse terão arquivos em comum com não usuários de caixas de interesse. Contudo, a busca pela comparação de tags permite uma busca rápida: o servidor pode usar indexação ou executar uma busca binária em tempo logarítmico.

Em um cenário onde a deduplicação não fosse prioridade, o protocolo CE não se faria necessário e novo protocolo apresentado pode ser facilmente adaptado a este cenário. Isto é, ao invés de usar o algoritmo CE para gerar a chave secreta K_m, o usuário pode

gerar uma chave secreta aleatória e cifrá-la utilizando PK_{ABE}. Como resultado, tags não são mais utilizadas e o *CIGer* não mais executa a deduplicação nos arquivos. Também, um *CIGer* malicioso não obtém nenhuma informação sobre os arquivos que possa utilizá-la para decifrá-los.

Como no protocolo original, o novo esquema aqui apresentado também depende de atributos para a identificação de interesses comuns. No entanto, no novo protocolo, os atributos são emitidos por uma autoridade de atributos. O protocolo proposto neste trabalho abandona a ideia de federações, como contida no trabalho de Silva et al. (2014) e utiliza apenas um servidor do tipo ABE (SAHAI; WATERS, 2005) que cuidará das identidades e das atribuições dos usuários. Essa autoridade emite chaves secretas para os usuários marcadas com os atributos deles. Também, essa autoridade emite assinaturas especiais que protegem os atributos dos usuários. Essas assinaturas procuram comparar os atributos dos usuários e assim identificar os interesses em comuns. Em uma abordagem mais flexível, usuários poderiam definir os próprios atributos enquanto criam uma caixa de interesse. Todavia, esta abordagem não é segura visto que os usuários poderiam definir qualquer conjunto de atributos e assim poderiam obter, indevidamente, arquivos de outros usuários.

4.2 Segurança Semântica

Nenhum algoritmo MLE alcança segurança semântica por definição. Segurança semântica formaliza a noção de que nenhuma informação parcial pode ser obtida. "Segurança Semântica"implica na "Indistinguibilidade", e vice-versa (KATZ; LINDELL, 2014). Que significa que a capacidade de se obter alguma informação da mensagem M a partir do texto-cifrado é desprezível (TREVISAN, 2009).

Considere dois experimentos: EXP(0) e EXP(1). Um "Desafiante"($\mathcal{D}$) cria um desafio para uma outra parte denominada "Adversário"($\mathcal{A}$). $\mathcal{D}$ gera secretamente uma chave $k \leftarrow K$. O adversário $\mathcal{A}$ envia para $\mathcal{D}$ duas mensagens: $m_0, m_1 \in M$. $\mathcal{D}$ encripta as duas mensagens, separadamente, utilizando a chave k. O experimento consiste em desafiar o adversário $\mathcal{A}$ à dizer qual texto-cifrado pertence a m_0 (experimento EXP(0)) e qual texto-cifrado pertence a m_1 (experimento EXP(1)). Diz-se que um algoritmo de encriptação $\mathbb{E}$ é *semanticamente seguro* se, para todos os possíveis adversários, a probabilidade de diferenciar uma cifra da outra no experimento anterior for desprezível: $Adv_{SS}[A, E] \leq \epsilon$, onde ϵ representa um valor desprezível (e.g., 2^{-60}). Portanto, dependendo da resposta do adversário, se ele conseguiu *distinguir* os textos cifrados de m_0 e m_1, classifica-se o algoritmo de encriptação como semanticamente seguro ou não (KATZ; LINDELL, 2014).

Após formalizado anteriormente o conceito de segurança semântica, assume-se por consequência que nenhum algoritmo MLE pode alcançar segurança semântica. Isto porque,

se uma mensagem-alvo M, derivada de um espaço S de tamanho s, um adversário que possui a cifra C de M pode recuperar M em $\mathcal{O}(s)$ tentativas: $\forall M' \in S, testar\ até\ \mathcal{D}(\mathcal{K}(\mathcal{M}', \mathcal{C}) = \mathcal{M}'$ (BELLARE; KEELVEEDHI; RISTENPART, 2013). E de acordo com os experimentos anteriores (EXP(0) e EXP(1)), se o adversário conhece as mensagens, ele também conhecerá as chaves de encriptação e obteria sucesso (i.e., distinguibilidade).

4.3 Ataques

Nesta seção estuda-se o comportamento do protocolo proposto sob alguns tipos de ataques. Em todos os ataques à seguir explora-se a fragilidade trazida pela deduplicação de dados. Um sistema que implementa a deduplicação de dados, pode desejar também a economia de banda. Esta foi uma das justificativas para a implementação de tags, feita na seção sobre deduplicação segura (Seção 2.5). Ao monitorar a transferência de dados pela rede, é possível saber, pela *timestamp* e pela quantidade dados transferidos, se foi utilizada a deduplicação.

4.3.1 Ataque da Confirmação de Arquivo

Digamos que Alice objetiva saber se Bob armazenou um determinado arquivo em uma nuvem de armazenamento de dados. A deduplicação pode ser usada para este fim. Se o arquivo em cheque é natureza restrita (i.e., dificilmente estará em possessão de outro usuário além de Bob), Alice faz *Upload* do arquivo em questão e verifica se o mesmo foi deduplicado. Caso positivo, pode-se entender que Bob já armazenou este arquivo anteriormente.

Este ataque pode ser utilizado por autoridades policiais que investiguem se um determinado arquivo ilegal está armazenado na nuvem de dados. Com uma ação judicial, essas autoridades podem pedir a identificação dos usuários que possuem esses arquivos (HARNIK; PINKAS; SHULMAN-PELEG, 2010).

Este ataque pode ser efetuado no protocolo proposto visto que não foi tratada a privacidade no registro dos dados usuários, mas apenas dos arquivos deles. Uma autoridade ainda pode, mesmo que verificando apenas os textos-cifrados, saber se Bob possui tal arquivo. Para contornar este aspecto é necessário que implementem-se mecanismos que protejam a identidade dos usuários registrados (e.g., permitir o registro anônimo). Este aspecto não foi tratado pois estava fora do escopo de estudo deste protocolo, onde estudou-se apenas a segurança dos dados *per se*.

4.3.2 Ataque do Aprendizado do texto restante

Diferente do ataque anterior que buscava apenas confirmar se um arquivo já fora armazenado, no ataque do aprendizado do texto restante busca-se saber, por tentativa e erro, o restante do conteúdo de um arquivo que conhece-se em parte. Por exemplo: digamos que Alice e Bob trabalhem em uma mesma empresa, e os dois receberam os comprovantes de pagamento dos salários respectivos. Alice deseja saber quanto Bob recebeu, e para isso utiliza o *template* de comprovantes de pagamentos e preenche com todas as informações que ela conhece sobre Bob. A partir disso, Alice, por meio de um ataque de força bruta, vai testando cada possível valor de salário e envia para a nuvem de armazenamento da empresa, e verifica se o arquivo foi deduplicado ou não. Caso tenha sido deduplicado, Alice conseguiu recuperar a informação restante que estava procurando. Caso não tenha sido deduplicado, Alice preenche o comprovante de pagamento com outros valores e continua até obter sucesso.

Este ataque pode ser aplicado quando a quantidade de possíveis versões do alvo for praticável. Essa fragilidade é inerente aos sistemas que utilizam a deduplicação, seja ela segura ou não. Uma forma de contornar esta fragilidade seria remover a economia de banda ao se abandonar a verificação por tag; ou implementar algum tipo de *delay*, fixo para toda transferência de dados, com o objetivo de reduzir a praticabilidade do ataque.

4.4 Conclusão

Neste capítulo foi discutido o novo protocolo a partir de observações sobre as características do mesmo, sobre suas limitações e seu comportamento diante de alguns ataques bem conhecidos na literatura. Apesar das observações sobre as fragilidades inerentes da segurança semântica, os ataques possuem pouca praticabilidade pois alguns cenários específicos devem ser estabelecidos. Um exemplo é a necessidade de um número pequeno de textos candidatos para poder obter o restante das informações desejadas (Ataque do Aprendizado do texto restante). Ainda assim, caso a deduplicação de dados não seja necessária, é possível removê-la sem prejuízo para o restante do protocolo, ou podem-se também implementar artifícios que reduzam a praticabilidade dos ataques, como por exemplo: implementação de um *delay* fixo para evitar o ataque de Confirmação de Arquivo. Desta forma, pode-se dizer que apesar das fragilidades identificadas, o novo protocolo proposto alcança um grau de segurança maior que o protocolo original de Silva et al. (2014).

5 Conclusão e Trabalhos Futuros

Neste trabalho foi apresentado um novo protocolo colaborativo e mais seguro quando comparado ao protocolo de Silva et al. (2014), base deste trabalho. O protocolo original permite a colaboração, através do compartilhamento de arquivos, de acordo com os interesses comuns dos usuários de uma nuvem de armazenamento de dados. Todavia, o protocolo original não implementa nenhum tipo de procedimento seguro para o armazenamento de dados. Isto é, os arquivos eram armazenados em texto-claro e qualquer invasor malicioso da nuvem de dados poderia recuperá-los.

Para assegurar os arquivos armazenados em nuvem, o novo protocolo proposto baseia-se em primitivas criptográficas seguras, como a encriptação baseada em atributos e a encriptação convergente. Ao mesmo tempo que o novo protocolo permite o armazenamento seguro dos dados, também é possível desempenhar a deduplicação de dados. Portanto, chunks redundantes poderão ser identificados e removidos, uma realidade incomum dadas as circunstâncias aleatórias dos algoritmos de encriptação (Seção 2.5). Contudo, como observado no capítulo de análise do protocolo (Seção 4) algumas fragilidades ainda podem ser notadas. Apesar disso, de modo geral alcança-se um protocolo prático, e mais seguro que o originalmente proposto.

No novo protocolo emprega-se uma única autoridade de atributos e um único gerenciador de caixas de interesses. Adicionalmente, considera-se que a autoridade de atributos é *confiável* (i.e., não conspira contra os usuários) e que o gerenciador de caixas de interesses é parcialmente confiável (i.e., pode ou não confabular contra os usuários). Para reduzir as premissas anteriores é necessário que o conjunto de partes do protocolo atue em uma configuração distribuída. Também, neste trabalho não foi tratada a revocação de atributos. Tais melhorias servirão para trabalhos futuros, assim como a implementação do protocolo em um cenário real.

Referências

AMAZON WEB SERVICES (AWS). 2016. Disponível em: <https://aws.amazon.com/>. Acesso em: 22 abr. 2016. Citado na página 21.

BELLARE, M.; KEELVEEDHI, S. Interactive message-locked encryption and secure deduplication. In: KATZ, J. (Ed.). *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*. Springer, 2015. (Lecture Notes in Computer Science, v. 9020), p. 516–538. Disponível em: <http://dx.doi.org/10.1007/978-3-662-46447-2_23>. Citado na página 21.

BELLARE, M.; KEELVEEDHI, S.; RISTENPART, T. Message-locked encryption and secure deduplication. In: JOHANSSON, T.; NGUYEN, P. Q. (Ed.). *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Springer, 2013. (Lecture Notes in Computer Science, v. 7881), p. 296–312. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38348-9_18>. Citado 4 vezes nas páginas 21, 25, 26 e 47.

BELLARE, M.; ROGAWAY, P. Optimal asymmetric encryption. In: SANTIS, A. D. (Ed.). *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*. Springer, 1994. (Lecture Notes in Computer Science, v. 950), p. 92–111. Disponível em: <http://dx.doi.org/10.1007/BFb0053428>. Citado na página 39.

CHARD, K. et al. Social cloud computing: A vision for socially motivated resource sharing. *IEEE Trans. Services Computing*, v. 5, n. 4, p. 551–563, 2012. Disponível em: <http://dx.doi.org/10.1109/TSC.2011.39>. Citado na página 13.

CHOW, R. et al. Controlling data in the cloud. *Proceedings of the 2009 ACM workshop on Cloud computing security - CCSW '09*, p. 85, 2009. ISSN 15437221. Disponível em: <http://portal.acm.org/citation.cfm?doid=1655008.1655020>. Citado na página 12.

DIFFIE, W. The First Ten Years of Public-Key Cryptography. *Proceedings of the IEEE*, v. 76, n. 5, p. 560–577, 1988. ISSN 15582256. Citado na página 18.

DIFFIE, W.; DIFFIE, W.; HELLMAN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, v. 22, n. 6, p. 644–654, 1976. ISSN 15579654. Citado na página 18.

DOUCEUR, J. R. et al. Reclaiming space from duplicate files in a serverless distributed file system. In: *ICDCS*. [s.n.], 2002. p. 617–624. Disponível em: <http://dx.doi.org/10.1109/ICDCS.2002.1022312>. Citado 3 vezes nas páginas 25, 26 e 27.

DRIVE, G. 2016. <http://www.drive.google.com>. Accessed: 2016-2-20. Citado 3 vezes nas páginas 11, 20 e 21.

DROPBOX. 2016. <http://www.dropbox.com>. Accessed: 2015-12-10. Citado na página 11.

EMC CORPORATION. EMC VNX2 Deduplication and Compression. n. March 2016. Disponível em: <https://www.emc.com/collateral/white-papers/h12209-vnx-deduplication-compression-wp.pdf>. Nenhuma citação no texto.

FEISTEL, H. Cryptography and computer privacy. v. 228, n. 5, p. 15–23, maio 1973. ISSN 0036-8733 (print), 1946-7087 (electronic). Disponível em: <http://www.nature.com/scientificamerican/journal/v228/n5/pdf/scientificamerican0573-15.pdf>. Citado 2 vezes nas páginas 16 e 23.

GOOGLE CLOUD PLATFORM. Disponível em: <https://cloud.google.com/>. Acesso em: 22 abr. 2016. Nenhuma citação no texto.

GOYAL, V. et al. Attribute-based encryption for fine-grained access control of encrypted data. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*. [s.n.], 2006. p. 89–98. Disponível em: <http://doi.acm.org/10.1145/1180405.1180418>. Citado 4 vezes nas páginas 27, 28, 29 e 30.

HACKERNEWS. 2010. <https://news.ycombinator.com/item?id=2478595>. Accessed: 2016-12-12. Citado na página 20.

HARNIK, D.; PINKAS, B.; SHULMAN-PELEG, A. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, v. 8, n. 6, p. 40–47, 2010. Disponível em: <http://dx.doi.org/10.1109/MSP.2010.187>. Citado na página 47.

IBM DEVELOPERWORKS. *Federação de identidade utilizando a SAML e o software WebSphere*. 2010. Disponível em: <https://www.ibm.com/developerworks/br/websphere/library/ws-SAMLWAS/index.html>. Acesso em: 21 fev. 2017. Citado na página 31.

KARP, R. M.; RABIN, M. O. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, v. 31, n. 2, p. 249–260, 1987. ISSN 0018-8646. Citado na página 24.

KATZ, J.; LINDELL, Y. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. (Chapman & Hall/CRC Cryptography and Network Security Series). ISBN 9781466570276. Disponível em: <https://books.google.com.br/books?id=-iDcBQAAQBAJ>. Citado 4 vezes nas páginas 15, 18, 26 e 46.

KOULOUZIS, S. et al. Cloud federation for sharing scientific data. In: *8th International Conference on eScience 2012*. [S.l.: s.n.], 2012. Citado na página 13.

LAURENT, J. S. *Data Efficiency: Deduplication*. Disponível em: <https://www.simplivity.com/blog/2015/03/data-efficiency-deduplication/>. Acesso em: 15 maio 2015. Nenhuma citação no texto.

LI, J. et al. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans. Parallel Distrib. Syst.*, v. 25, n. 6, p. 1615–1625, 2014. Disponível em: <http://dx.doi.org/10.1109/TPDS.2013.284>. Citado 2 vezes nas páginas 13 e 14.

MARINESCU, D. C. Chapter 3 - cloud infrastructure. In: MARINESCU, D. C. (Ed.). *Cloud Computing*. Boston: Morgan Kaufmann, 2013. p. 67 – 98. ISBN 978-0-12-404627-6. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780124046276000038>. Citado na página 22.

MEYER, D. T.; BOLOSKY, W. J. A study of practical deduplication. *TOS*, v. 7, n. 4, p. 14, 2012. Disponível em: <http://doi.acm.org/10.1145/2078861.2078864>. Citado na página 23.

MICROSOFT AZURE. Disponível em: <https://azure.microsoft.com/en-us/>. Acesso em: 22 abr. 2016. Nenhuma citação no texto.

National Institute of Standards and Technology. *Final Version of NIST Cloud Computing Definition Published*. http://www.nist.gov/: NIST, 2011. NIST Special Publication 800-145. Disponível em: <http://dx.doi.org/10.6028/NIST.FIPS.202>. Nenhuma citação no texto.

National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. http://www.nist.gov/: NIST-FIPS, 2015. Federal Inf. Process. Stds. (NIST FIPS) - 202. Disponível em: <http://dx.doi.org/10.6028/NIST.FIPS.202>. Citado 3 vezes nas páginas 16, 27 e 39.

OASIS. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. 2008. Disponível em: <https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf>. Acesso em: 21 fev. 2017. Citado na página 31.

ONEDRIVE. 2016. Disponível em: <https://onedrive.live.com/about/en/>. Acesso em: 10 dez. 2015. Citado 3 vezes nas páginas 11, 20 e 21.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, ACM, New York, NY, USA, v. 21, n. 2, p. 120–126, fev. 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359340.359342>. Citado na página 20.

SAHAI, A.; WATERS, B. Fuzzy Identity-Based Encryption. *EUROCRYPT 2005, Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, p. 457–473, 2005. ISSN 03029743. Citado 3 vezes nas páginas 27, 28 e 46.

SILVA, F. d. et al. Caixas de interesses: um novo mecanismo para a colaboração através de nuvem de armazenamento de dados. In: SBSC. *SBSC 2014 Proceedings - Ongoing Research*. [S.l.], 2014. p. 102–110. Citado 16 vezes nas páginas 4, 5, 12, 14, 15, 25, 30, 31, 32, 33, 34, 35, 44, 46, 48 e 49.

STALLINGS, W. *Cryptography and Network Security - Principles and Practice (5. ed.)*. [S.l.]: Prentice Hall, 2011. ISBN 978-0-13-609704-4. Citado 5 vezes nas páginas 15, 16, 18, 19 e 20.

TREVISAN, L. *Notes for Lecture 2, U.C. Berkeley — CS276: Cryptography*. 2009. Disponível em: <https://people.eecs.berkeley.edu/~luca/cs276/lecture02.pdf>. Acesso em: 10 dez. 2015. Citado 3 vezes nas páginas 15, 16 e 46.