UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Hy-SAIL: UMA NOVA ABORDAGEM PARA DISTRIBUIÇÃO E ARMAZENAMENTO DE INFORMAÇÕES EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

DINO MACEDO AMARAL

ORIENTADOR: ANDERSON CLAYTON ALVES NASCIMENTO

TESE DE DOUTORADO EM
ENGENHARIA ELÉTRICA

BRASÍLIA/DF: NOVEMBRO/2013.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Hy-SAIL: UMA NOVA ABORDAGEM PARA DISTRIBUIÇÃO E ARMAZENAMENTO DE INFORMAÇÕES EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

DINO MACEDO AMARAL

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Anderson Clayton Alves Nascimento, PhD. (ENE-UnB)
(Orientador)

Prof. Rafael Timóteo de Sousa Júnior, Dr. (ENE-UnB)
(Examinador Interno)

Prof. Edna Dias Canedo, Dra. (FGA-UnB)
(Examinadora Interna)

Prof. Robson de Oliveira Albuquerque, Dr. (ABIN)
(Examinador Externo)

Prof. Laerte Peotta de Melo, Dr. (Banco do Brasil)
(Examinador Externo)

BRASÍLIA/DF, 26 DE NOVEMBRO DE 2013.

FICHA CATALOGRÁFICA

AMARAL, DINO MACEDO

Hy-SAIL: Uma nova abordagem para armazenamento de informações em ambientes de computação em nuvem . [Distrito Federal] 2013.

xii, 82 p., 297 mm (ENE/FT/UnB, Doutor, Engenharia Elétrica, 2013).

Tese de Doutorado - Universidade de Brasília.

Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Computação em Nuvem	2. Criptografia
3. Protocolos de Prova de Recuperabilidade	4. Fountain Codes
5. Armazenamento em Nuvem	6. MAC homomórfico
7. Código corretor de erros	
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Amaral, D. M. (2013). Hy-SAIL: Uma nova abordagem para distribuição e armazenamento de informações em ambientes de computação em nuvem. Tese de Doutorado em Engenharia Elétrica, Publicação PPGEE.TD - 082 A/2013, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 82p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Dino Macedo Amaral.

TÍTULO DA TESE DE DOUTORADO: Hy-SAIL: Uma nova abordagem para distribuição e armazenamento de informações em ambientes de computação em nuvem.

GRAU / ANO: Doutor / 2013

Dino Macedo Amaral

SQS 105 - Bloco H - Apt. 503, Asa Sul - CEP: 70.344-080 Brasília - DF - Brasil.

DEDICATÓRIA

Dedico este trabalho à minha esposa Suzanne,
e aos meus filhos, Lukito e Mel,
pelo apoio nos momentos difíceis
e o incentivo de sempre seguir em frente.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus por mais uma etapa concluída na minha vida.

À Minha Esposa que me acompanhou, me apoiou e sempre me incentivou na busca de melhores resultados.

À minha mãe, Dona Cida, e ao meu irmão, Dedé, os meus agradecimentos, que mesmo à distância, me incentivaram no percurso desta jornada. Valeu mesmo pelos inúmeros momentos de descontração que passamos juntos, o que me ajudou a ter energia suficiente para concluir este trabalho.

Agradeço ao orientador, Prof. Anderson Nascimento, pela confiança depositada e pelo apoio durante todo o curso de Doutorado. Devo agradecer também a todas as pessoas do grupo de *Cripto* que colaboraram para a conclusão deste trabalho.

Ao colega Josias (Joca), que sempre demonstrou paciência e tranquilidade para alimentar discussões que muito contribuíram para o alcance desse resultado.

À Dona Marlene, meus sinceros agradecimentos pelas palavras de incentivo.

Aos professores Rafael Timóteo, Robson Albuquerque, Edna Canedo, Laerte Peotta e Flávio Elias pelas constribuições fornecidas para a finalização deste trabalho.

Aos gestores da Diretoria de Gestão da Segurança da Informação do Banco do Brasil, instituição que apoiou, financiou e acreditou nesta pesquisa.

Dino Macedo Amaral

RESUMO

Hy-SAIL: UMA NOVA ABORDAGEM PARA DISTRIBUIÇÃO E ARMAZENAMENTO DE INFORMAÇÕES EM AMBIENTES DE COMPUTAÇÃO EM NUVEM

Autor: Dino Macedo Amaral

Orientador: Anderson Clayton Alves Nascimento

Programa de Pós-graduação em Engenharia Elétrica

Brasília, Novembro de 2013

A preocupação com os dados armazenados em um ambiente de Computação em Nuvem tem sido amplamente discutido. Entre os desafios, há o problema de garantir a integridade e recuperabilidade dos dados armazenados remotamente. Com o objetivo de preencher esta lacuna, a comunidade de criptografia tem proposto alguns conceitos, entre eles: *PDP* (Prova de Possessão de Dados), *PoW* (Prova de Propriedade) e *PoR* (Prova de Recuperabilidade).

Neste documento, é proposto um novo esquema de *PoR*: *Hy-SAIL* (*Hyper-Scalability, Availability and Integrity Layer*). Nesta proposta, um *PoR* é projetado e implementado usando *Online Codes*, um caso especial de *Fountain Codes*, . Para executar as verificações de integridade, propriedades em um Corpo de Galois, $GF(2^n)$, são usadas para construir um *MAC* com propriedade XOR homomórfica. Foi demonstrado que o *Hy-SAIL* é um sistema criptográfico seguro e escalável para fornecer disponibilidade dos dados armazenados remotamente, atendendo às exigências no que tange às complexidades de comunicação, armazenamento e processamento. Para tanto, apresenta-se um novo modelo adversarial que concentra as principais funcionalidades de um adversário realístico para ambientes de Computação em Nuvem, chamado de Modelo de Corrupção Limitada. É provado analiticamente que o *Hy-SAIL* possui segurança demonstrável nesse modelo adversarial, e que a probabilidade de ataque à este esquema é assintonticamente desprezível. São mostrados também os resultados experimentais coletados da implementação do *Hy-SAIL*, confirmando a prova analítica.

ABSTRACT

Hy-SAIL: A novel approach for distributing and storing data in cloud computing enviroment

Author: Dino Macedo Amaral

Supervisor: Anderson Clayton Alves Nascimento

Programa de Pós-graduação em Engenharia Elétrica

Brasília, November of 2013

Cloud computing has gained increasing attention from the industry and research communities. Despite the crucial benefits provided by this new paradigm, some numerous challenges arise. The concern about data stored in the cloud computing enviroment has been widely discussed, in according to recents events shown in the international media. Among these challenges, there is the problem of ensuring the integrity and retrievability of users' data stored in the cloud. Many definitions has been proposed: PDP (Proof of Data Possession), PoR (Proof of Retrievability), PoW (Proof of Ownership). The difference among them reside in the guarantee to retrieve the data stored remotely.

In this paper, we propose a novel cryptographic system: *Hy-SAIL* (Hyper-Scalability, Availability and Integrity Layer). In the proposed protocol, a new PoR scheme is built using an Online Codes, a special case of Fountain Codes, as building blocks which adds a higher degree of availability of stored data. To perform integrity checks, properties in Galois Field, $GF(2^n)$, is used to build a MAC with XOR homomorphic property. It is demonstrated that *Hy-SAIL* leads to an efficient and scalable cryptographic system that meets near-optimal bounds in the communication, storage and time complexities. Finally, the Bounded Corruption Model, a new adversarial model that aggregates the main functionalities of a realistic adversary in cloud computing environments is proposed. It is proved that *Hy-SAIL* has provable security in this new approach. It is also shown the results collected of an unoptimized implementation.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE PSEUDO-CÓDIGOS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

IDC: International Data Corporation.

MAC: Message Authentication Code.

IPH-ECC: Integrity Protected Homomorphic Error Correcting Code.

PDP: Proof of Data Possession.

PoR: Proof of Retrievability.

PoW: Proof of Ownership.

PRF: Pseudo Random Function.

UHF: Universal Hash Function.

1 INTRODUÇÃO

As previsões do IDC em 2012 (GANTZ; REINSEL, 2012) projetam que o total de dados gerados e armazenados no universo digital deve alcançar 40 *zettabytes* em 2020. No ano de 2012, o total estimado foi de 2.8 *zettabytes* contra os 0.13 *zettabytes* de 2005. O ponto relevante desta tendência se refere à forma como se dá o armazenamento dessa massa de dados. Se antes os dados ficavam armazenados em CDs, disquetes e *pen-drives*, hoje esses dados ficam também armazenados em *data centers*. Toda essa movimentação de dados possui origem no modelo atual de computação e armazenamento, no qual os dados e até mesmo aplicativos ficam hospedados em *data centers*.

Para suprir essa nova tendência, as tarefas que demandam processamento e armazenamento dessa massa de dados exigem uma arquitetura diferenciada, de modo que tais tarefas possam ser divididas para otimização de tempo, fornecendo um resultado final de maneira eficiente e rápida. A distribuição destas tarefas deve utilizar a alocação de recursos computacionais locais e também de recursos geograficamente espalhados através da Internet com a heterogeneidade que o ambiente fornece. Para suportar demandas de processamento distribuído existem conceitos como *Cluster*, *Grid* e recentemente, *Cloud Computing* (Computação em Nuvem).

Um *cluster* (ANDERBERG, 1973) é um ambiente que possui dois ou mais computadores (chamados de nós), que interligados trabalham para executar tarefas de grande porte, fornecendo a impressão de um único recurso integrado. Sendo uma combinação de hardware e software, a computação em *grid* é um modelo computacional motivado pela necessidade de recursos computacionais para atender demandas no campo de pesquisas científicas. Para um melhor entendimento, *grid* (THAIN; LIVNY, 2003) é um *middleware* de computação distribuída que fornece compartilhamento de recursos de maneira coordenada para aplicações com alta demanda computacional, tais como cálculos científicos e de engenharia.

Com o objetivo de otimizar os recursos disponíveis nos *data centers*, um novo paradigma de computação compatível com este cenário foi concebido, com as seguintes características: sistemas de arquivos distribuídos (SHVACHKO et al., 2010; GHEMAWAT; GOBIOFF; LEUNG, 2003), alto nível de escalabilidade, interoperabilidade

entre sistemas heterogêneos, virtualização e, principalmente, um modelo comercial para oferecer serviços através da Internet (CHANG et al., 2010).

O conceito de computação em nuvem atende estes requisitos e pode ser definido como uma quantidade expressiva, da ordem de milhares, de computadores conectados através de uma rede, que possuem a capacidade de trabalhar simultânea e cooperativamente, sob demanda, a fim de entregar resultados de maneira rápida e eficiente (ARMBRUST et al., 2009).

Com a movimentação dos recursos disponíveis nos *desktops* e computadores portáteis, que se direcionam cada vez mais para grandes *data centers* (WEISS, 2007), é possível perceber que a disponibilização de recursos computacionais em larga escala não ficou disponível apenas às grandes empresas de tecnologia.

Com a computação em nuvem é possível disponibilizar uma infraestrutura, de processamento e armazenamento, com alto nível de abstração, capaz de realizar tarefas complexas para qualquer usuário com acesso à Internet, ofertando serviços sob demanda. Esses serviços possuem uma classificação quanto à disponibilização, seja um *software*, uma plataforma de desenvolvimento ou uma infraestrutura de processamento ou armazenamento. Em (CANEDO, 2012) é apresentada esta classificação segundo suas respectivas aplicabilidades no âmbito tecnológico.

A utilização deste paradigma, além oferecer mobilidade e otimização dos recursos tecnológicos, constitui em uma possibilidade para um aumento de competitividade nos negócios. Esta convergência de conceitos tem o propósito de permitir um melhor aproveitamento dos recursos computacionais distribuídos com a oferta de novas funcionalidades, que até então, eram inviáveis aos usuários com seus dispositivos de baixo poder computacional.

A definição do conceito de Computação em Nuvem é motivo de questionamentos, porém em (MELL; GRANCE, 2011) os autores destacam 5 características essencias que compõem este ambiente :

- Autoatendimento sob demanda: Um usuário com uma necessidade não programada pode se dispor de recursos computacionais (tais como tempo de processamento, armazenamento em disco, imagem de um sistema operacional com ins-

talação padrão) de maneira automática sem a necessidade de interação humana na configuração desses recursos.

- Amplo acesso à rede: Estes recursos computacionais são disponibilizados através da Internet e usado por várias aplicações com plataformas heterogêneas, tanto de hardware como de software. A variedade dos dispositivos que acessam os serviços disponíveis exige uma camada de abstração que permite interagir com qualquer plataforma, esta ausência de restrição contribui para um maior difusão do conceito de computação em nuvem.

- Recursos compartilhados: Um provedor de serviços em nuvem realiza uma junção dos recursos computacionais disponíveis com o objetivo de atender à múltiplos usuários, seja através de virtualização ou *multitenancy*. O resultado desta junção é transparente ao usuário, o qual possui pouco controle ou conhecimento da origem, composição ou localização desses recursos (memória, armazenamento em disco, processamento).

- Rápida elasticidade: Para os usuários, os recursos computacionais estão a disposição de forma imediata, não havendo um compromisso contratual de uma quantidade fixa. Os recursos são oferecidos de maneira flexível e escalável, permitindo a liberação destes recursos assim que terminam as tarefas para os quais foram designados. Para o usuário, os recursos provisionados aparentam ser infinitos, o que permite atender demandas pontuais, a qualquer momento, de maneira rápida e eficiente. Esta elasticidade contribui para uma economia no custo operacional dos recursos.

- Medição dos serviços utilizados: Embora os recursos computacionais estão agregados e compartilhados por vários usuários, a infraestrutura em nuvem está apta a utilizar mecanismos apropriados para medir o uso destes recursos por cada usuário de maneira individual, através de suas capacidades de medição.

Com as caracteríticas acima mencionadas, o conceito de computação em nuvem fornece novos métodos para o armazenamento e processamento de grandes quantidades de dados. Com o intuito de processar grande quantidade de dados, o *Google* introduziu o seu *framework MapReduce* (DEAN; GHEMAWAT, 2008a) com o sistema de arquivo distribuído correspondente (GHEMAWAT; GOBIOFF; LEUNG, 2003) e que foi amplamente popularizado através de uma implementação de código aberto, chamado *Hadoop* (BORTHAKUR, 2007). Basicamente, esse paradigma consiste de mecanismos de processamento e armazenamento distribuídos. Esta abordagem é utilizada

por aplicações que precisam processar quantidade expressiva de dados (da ordem de *terabytes* e *petabytes*), tais como: indexação de documentos, atualizações constantes em mecanismos de buscas, análise de grafos e processamento de dados estatísticos. A Figura 1.1 exemplifica como funciona a submissão de uma tarefa no paradigma de computação *MapReduce*.

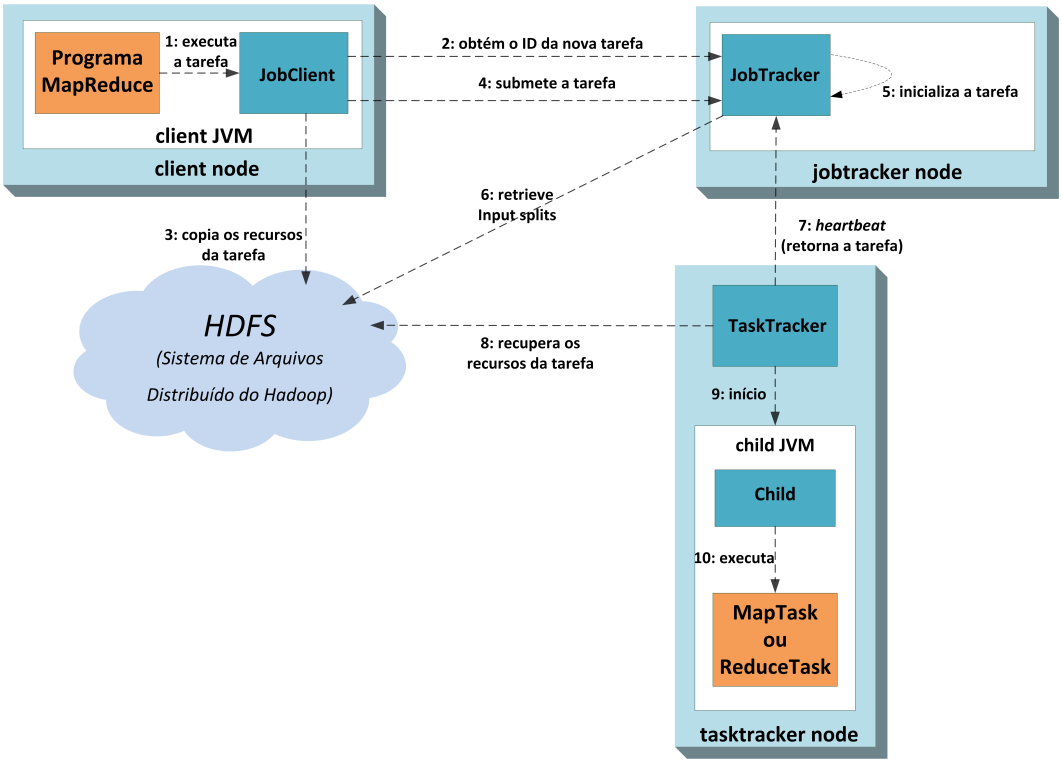

Figura 1.1: Anatomia de submissão de uma tarefa no paradigma MapReduce (WHITE, 2009)

No modelo de computação *MapReduce*, cada tarefa recebe um conjunto de entrada de "variável/valor" e produz um conjunto de saída de "variável/valor", isto é possível ao utilizar duas funções para executar esta operação: *Map* e *Reduce*. A função *Map* recebe os dados de entrada a serem processados e fornece um conjunto de valores intermediários e os encaminha para a função de redução. A função *Reduce* recebe variáveis já mapeadas com seus respectivos valores intermediários, realizando uma soma dos mesmos com o objetivo de formar o resultado final na relação "variável/valor" do arquivo processado. Embora o modelo de processamento em *batch* do *MapReduce* consiga atender as demandas atuais de processamento em larga escala, a necessidade de processamento em tempo real consiste em uma fronteira a ser transposta por este modelo.

Como a abordagem de processamento está bem definida sob este modelo de *MapReduce*, é factível a observação que usuários de Internet, para usufruir deste conceito de com-

putação em nuvem, podem dispensar do controle físico de seus dados para a obtenção de custos menores e recursos com altos índices de disponibilidade. Isto, no entanto, aumenta a dependência dos provedores de serviços e coloca uma série de questões de segurança em pauta. Uma das preocupações é como garantir a integridade e disponibilidade de dados armazenados remotamente em provedores de serviços (KAUFMAN, 2009), que é o foco deste trabalho. Em (BERENSON, 2013) são mostrados fatos que reportam ações de perda de dados e de privacidade, o que tem atraído atenção especial da mídia internacional, confirmando que estas preocupações não são infundadas.

O principal atrativo para armazenamento em nuvem é a redução no custo de armazenamento, juntamente com o aumento da disponibilidade dos dados, que podem ser acessados a partir de qualquer dispositivo com acesso a Internet. Em (JADEJA; MODI, 2012), os autores reforçam esta ideia adicionando outras demandas que serão os alicerces para a estratégia da nova geração de *data centers*, trazendo elasticidade, escalabilidade, viabilidade econômica e um forte apelo ambiental, no que diz respeito a gasto de energia.

O armazenamento em nuvem usa princípios de um modelo escalável, o qual assegura uma maneira flexível de lidar com demandas elásticas de grande quantidade de dados. No contexto deste trabalho, o termo elástico se refere a capacidade do ambiente computacional de aumentar ou diminuir seus recursos de forma automática. Então, as demandas elásticas se referem à possibilidade de criação e disponibilização de unidades de armazenamento tão logo que sejam solicitadas. A terceirização da gestão de dados significa que o usuário paga um determinado valor a um provedor para armazenar seus dados, que os disponibilizarão para o próprio usuário sob demanda. A adoção por este serviço ocorre pelo crescente volume de dados que devem ser acessados de maneira rápida e ubíqua.

O serviço de armazenamento em nuvem visa a eficácia no uso de unidades de armazenamento e expõe os dados armazenados a uma terceira parte. De maneira geral, este serviço deve prover algumas características:

1. Mecanismo para rápida detecção de arquivo corrompido;
2. Garantir a integridade dos dados;
3. Possibilidade de reconstrução parcial das informações sem a necessidade de efetuar o download do arquivo;

4. Recuperabilidade dos arquivos armazenados, até mesmo, se uma quantidade limitada dos mesmos for corrompida (CAO et al., 2012).

As preocupações para adoção do modelo de computação em nuvem estão relacionadas a segurança que os provedores podem oferecer quanto ao transporte, armazenamento e processamento das informações. Os principais obstáculos a serem observados são:

- Controle de Acesso: Embora o proprietário das informações adote uma política de acesso de maneira a prevenir algum tipo de ação que cause indisponibilidade das mesmas, é comum que usuários com privilégio de administrador possua amplo acesso aos dados armazenados, pois o manuseio dos dados é exigido para caso de *backups*, atualização de softwares nos servidores e recuperação de desastres.

- Conformidade com as leis locais: Os usuários devem ter amplo acesso aos dados, para atender situações como legislação de seu país de origem, auditorias externas, investigação de incidentes e empresas de segurança responsável por certificações. Dessa forma, a localização dos dados torna-se um ponto crítico. As circunstâncias definidas na contratação do serviço devem estar de acordo com a jurisdição a qual o usuário se submete em seu país de origem.

- Integração com diversos provedores: Em ambiente dinâmico de TI, as empresas buscam locais que oferecem melhor desempenho ou menor custo para tratamento das informações. A padronização na formatação dos dados deve ser objeto do contrato entre provedor e usuário, para evitar possíveis problemas de indisponibilidade. Questões operacionais como versão da plataforma de desenvolvimento e sistema operacional não podem ser empecilhos para migração de dados entre diferentes provedores.

- Disponibilidade e Integridade das informações: Ao mover dados para uma nuvem de servidores em *data centers* remotos, fica subentendido que os usuários não possuem controle físico dos mesmos. Para usufruir do pleno conceito de computação em nuvem, estes dados devem estar disponíveis a qualquer momento, para qualquer usuário (desde que autorizado) e em qualquer dispositivo, de maneira íntegra. Mecanismo de replicação de dados, em repositórios diferentes, ajuda a garantir um maior nível de disponibilidade. A verificação de integridade deve ser uma rotina para evitar o manuseio de dados que não correspondem ao conteúdo original.

- Escalabilidade de processamento e armazenamento: Escalabilidade se refere a habilidade em aumentar ou diminuir seus recursos computacionais em resposta às respectivas mudanças de demanda. A elasticidade implica em uma detecção automática na mudança de requisitos, o que permite uma economia substancial de recursos e a possibilidade de provisionamento dos mesmos para outras aplicações. A escalabilidade é associada à resiliência, onde a perda de um componente pode ser suprido pela adição de um novo componente.

- Vazamento de informações com a virtualização: A utilização de várias máquinas virtuais simulteneamente em um mesmo servidor pode permitir a extração de informações através de ataques *side-channels*. Estes ataques se baseiam em caracteríticas físicas coletadas na execução de um criptosistema, tais como consumo de energia, tempo exigido para cálculos internos ou ondas eletromagnéticas (RISTENPART et al., 2009). O isolamento das máquinas virtuais consiste em um desafio para o provedor, pois a virtulização consiste em um dos pilares deste modelo.

1.1 OBJETIVO

O objetivo deste trabalho é propor e avaliar um esquema de armazenamento em nuvem que possibilita a verificação remota de integridade dos arquivos e garanta a disponibilidade dos mesmos.

1.2 OBJETIVOS ESPECÍFICOS

Diante do objetivo principal, foram definidos os seguintes objetivos específicos:

- Fazer um levantamento dos aspectos relevantes de segurança para a utilização de ambientes de computação em nuvem;

- Analisar os esquemas de prova de recuperabilidade encontrados na literatura e analisar os pontos relevantes na construção dos mesmos;

- Propor um novo esquema de prova de recuperabilidade com segurança demonstrável;

- Implementar protótipo de um protocolo de prova de recuperabilidade visando otimizar aspectos computacionais (armazenamento, processamento e largura de banda);

- Coletar informações quanto ao desempenho do protótipo;
- Possuir requisitos de desempenho no que tange à armazenamento, processamento e consumo de largura de banda.
- Apresentar uma prova de segurança formal mediante um modelo adversarial que reflita a realidade de uma ambiente de Computação em Nuvem

1.3 JUSTIFICATIVA

Recentes falhas em provedores de armazenamento em nuvem (BLODGET, 2011) tem colocado em questão a disponibilidade dos dados armazenados. As soluções encontradas consistem em replicar os arquivos em diversos provedores ou aplicar código corretores de erros (UNDHEIM; CHILWAN; HEEGAARD, 2011), que permita suportar uma perda considerável de parte do arquivo codificado sem prejuízo de sua recuperabilidade. Embora, ambos impliquem em um custo adicional de armazenamento, a primeira abordagem é mais simples de implementar, e a segunda apresenta maior eficácia.

No que tange à integridade, ao enviar seus dados para um provedor de armazenamento remoto, o usuário deve realizar consultas para garantir que o conteúdo dos dados não foram alterados. A verificação de integridade dos dados deve ser conduzida de maneira periódica e sem o conhecimento explícito dos dados armazenados (NEPAL et al., 2011), apenas com poucas informações já previamente obtidas, por exemplo o *hash* dos arquivos, antes do envio.

1.4 CONTRIBUIÇÕES DESTE TRABALHO

Neste documento é apresentado o *Hy-SAIL*(*Hyper Scalability Availability Integrity Layer*), um novo protocolo de Prova de Recuperabilidade(*PoR*), com alguns diferenciais em relação aos *PoR*'s existentes na literatura. São listados os pontos considerados relevantes para uma proposta de armazenamanto de dados em ambientes distribuídos. As principais melhorias que o *Hy-SAIL* oferece são as seguintes:

- Alta Escalabilidade: No *Hy-SAIL*, a capacidade de atender demandas elásticas de armazenamento, de maneira transparente para o usuário, é viabilizado pela

aplicação do código corretor de erros, *Online Codes* (MAYMOUNKOV, 2002), e a flexibilidade no gerenciamento das unidades de armazenamento. Com essa flexibilidade é possível adicionar indistintamente unidades de armazenamento à camada de abstração criada pelo *Hy-SAIL*. O *Online Codes* possui importantes propriedades, como: codificação local, tempo linear para codificação/decodificação, possibilidade de fornecer uma quantidade infinita de blocos codificados. Estas propriedades fornecem a flexibilidade para o *Hy-SAIL* e diferem das propostas apresentadas em (BOWERS; JUELS; OPREA, 2009a; JUELS; KALISKI, 2007; BOWERS; JUELS; OPREA, 2009b; SHACHAM; WATERS, 2008), que utilizam *Reed-Solomon*(REED; SOLOMON, 1960) como código corretor de erros. Em 5.1 esta abordagem é mostrada.

- Novo esquema de *PoR*: O esquema de *PoR* proposto é especialmente projetado para sistemas de computação distribuída. Para realizar verificações de integridade, o *Hy-SAIL* possui uma abordagem com quantidades ilimitadas de verificação de integridade, baixa complexidade de comunicação entre usuário e provedor de armazenamento, independência quanto a quantidade de provedores indisponíveis e verificação completa do arquivo em uma única consulta. Esta unificação apresenta um avanço nos esquemas de *PoR*'s em comparação aos anteriores, pois o *HAIL* (BOWERS; JUELS; OPREA, 2009a) se baseia na quantidade de provedores que estão indisponíveis e exige que o servidor armazene blocos de paridade para auxiliar na verificação de integridade; e o esquema *Juels-Kaliski*(SHACHAM; WATERS, 2008) possui complexidade considerável de comunicação entre usuário e provedor, e consulta uma pequena quantidade de blocos do arquivo para verificação de integridade; e o esquema *Shacham-Waters* (JUELS; KALISKI, 2007) possui um modelo com quantilidade limitadas de verificação de integridade. Nas seções 4.1 e 4.2 estes resultados são apresentados.

- Suporte para armazenamento distribuído em ambientes heterogêneos: *Hy-SAIL* é projetado para suportar condições heterogêneas por parte dos provedores, seja de sistema operacional, quantidade de armazenamento disponível, qualidade de conectividade. Como resultado, é possível ter uma nuvem de provedores com poder global de processamento e armazenamento constituído de dispositivos com baixo poder computacional.

- Novo modelo adversarial: O modelo adversarial mostra as principais funcionalidades de uma realidade de ambiente de computação em nuvem, pois tanto o provedor quanto o atacante podem prejudicar a recuperabilidade dos arquivos. Sob esta perspectiva, a entidade maliciosa possui a capacidade de modificar uma

quantidade limitada dos dados armazenados nos servidores. Por outro lado, o provedor pode tentar responder aos desafios enviados pelo usuário sobre o código autenticador de mensagem de determinados blocos, na seção 4.4, mostraremos que esta probabilidade é desprezível.

1.5 METODOLOGIA

A metodologia consiste em identificar pontos relevantes para o tema proposto neste trabalho, realizando um levantamento de métricas e fundamentação teórica para a apresentação de uma nova proposta. A metodologia de pesquisa descritiva e experimental foi dividida em fases, para facilitar o entendimento do trabalho segue as fases:

- Fase 1: Realizar pesquisa bibliográfica para a obtenção de informações sobre os pontos relevantes do tema: Computação em Nuvem. Pesquisar sobre armazenamento e processamento de informações em ambientes remotos. Para este contexto definimos o termo remoto como sendo ambientes acessados somente através da Internet e não através da rede local. Os artigos e documentos foram analisados e registrados para uma maior eficácia do trabalho;
- Fase 2: Obter informações detalhadas sobre as implementações de protocolos de prova de recuperabilidade. Como resultado desta fase, será proposto um novo protocolo de prova de recuperabilidade que apresente uma evolução em relação aos já estudados no que tange os requisitos computacionais (armazenamento, processamento e largura de banda).
- Fase 3: Desenvolver e implementar um protótipo, o qual tem a finalidade de atestar a viabilidade prática do esquema proposto através da coleta de informações do testes realizados e também verificar o funcionamento básico de um protocolo de prova de recuperabilidade.

1.6 ORGANIZAÇÃO

Esta tese está organizada da seguinte maneira: no capítulo 2 são abordados a notação utilizada, e a fundamentação téorica de código corretor de erros e álgebra polinomial. O capítulo 3 apresenta um estudo sobre protocolos de prova de recuperabilidade e suas implicações. No capítulo 4 é apresentada uma visão geral do *Hy-SAIL* com os modelos (Limitado e Ilimitado) de consulta de verificação de integridade e a prova de segurança

quanto à integridade e disponibilidade dos blocos do arquivo a serem armazenados. No capítulo 5, as 5 fases que compõem o *Hy-SAIL* são detalhadas, mostrando a construção de cada fase e os resultados da implementação de um protótipo, buscando encontrar parâmetros quanto ao desempenho de um protocolo de prova de recuperabilidade. No capítulo 6 são apresentados a conclusão deste trabalho.

2 CONCEITOS E FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos téoricos relacionados a este trabalho e que fundamentam a construção do protocolo de *PoR*, *Hy-SAIL*. Na seção 2.1, segue a notação usada neste documento, com referências aos parâmetros necessários para a codificação, decodificação e execução do desafio-resposta dos arquivos a serem armazenados remotamente. Na seção 2.2, é mostrado o conceito de código corretor de erros bem como suas funcionalidades e descrição detalhada do *Online Codes*. Na seção 2.3, as propriedades pertinentes a estrutura de Corpos Finitos que serão utilizados na composição do MAC(Message Authentication Code) proposto pelo *Hy-SAIL* e verificação da integridade dos arquivos.

Neste documento, são utilizados os termos "*Provador*" e "*Verificador*". O termo "*Provador*" designa o usuário que deseja armazenar, acessar e realizar desafios de integridade com os arquivos. O termo "*Verificador*" será utilizado para referenciar o provedor de armazenamento remoto que disponibiliza os arquivos e responde aos desafios propostos pelo usuário.

O termo *provador* indica que o usuário deve provar que é o proprietário dos arquivos, enquanto que o termo *verificador* ressalta a característica do provedor que recebe, verificar as solicitações de "desafio-resposta". Tais termos podem ser referenciados pelas suas iniciais, ou seja, o *provador* é referenciado pela letra "P" e o *verificador* pela letra "V".

2.1 NOTAÇÃO

Nesta seção, é apresentada a notação usada na elaboração deste documento:

- **F** - Representa um arquivo de tamanho arbitrário a ser processado, distribuído e armazenado pelo *Hy-SAIL*.
- n - Um dos parâmetros do código coretor de erros para a fragmentação do arquivo

F. Para ser codificado, o arquivo **F** é fragmentado em "n" partes de tamanhos iguais, possuindo a seguinte forma $F = \{m_i\}_{i=1}^{n} = \{m_1||m_2||\ldots||m_n\}$.

- m_i - Bloco do arquivo **F** de tamanho $\frac{|F|}{n}$. Estes blocos serão utilizados na composição dos blocos codificados c_j.
- $m_i(x)$ - Representação polinomial do bloco de arquivo (m_i) em um GF(2^n), $m_i(x) = \sum_{i=1}^{n} a_{i-1}x^{i-1}$, onde a = 0,1.
- c_j - Representação dos blocos codificados, cujo conteúdo é o resultado de operação XOR entre os blocos m_i.
- δ - Percentual de perda que ocorre em um meio de comunicação, ao enviar uma quantidade de blocos (m_i) de arquivo entre 2 ou mais nós distintos.
- ϵ - Percentual de redundância que o arquivo original receberá para suportar possíveis perdas no meio de comunicação entre 2 ou mais nós distintos.
- **k** - Quantidade de blocos de mensagens que são repetidos para cada bloco auxiliar.
- **R** - Capacidade do meio de comunicação entre 2(dois) ou mais nós distintos, corresponde a taxa de $1 - \delta$.
- **t** - Tamanho em *bits* do código autenticador de mensagem proposto pelo *HySAIL*.
- n' - Número de blocos após a realização da primeira fase no código corretor de erros, corresponde à $(1 + k\delta)n$ blocos.
- D - Quantidade máxima de blocos m_i que cada bloco codificado pode suportar. Esta variável é calculada de acordo com os parâmetros n, δ e ϵ. A fórmula para o cálculo desta variável está descrita na seção 2.2.
- d - Grau de cada c_j, o qual se refere a quantidade de m_i que compõem um determinado bloco codificado.
- τ_{maximo} - Limite máximo de blocos corrompidos em cada provedor, de maneira a não prejudicar a recuperabilidade do arquivo **F**.
- $\tau_{provedor}$ - Quantidade de blocos codificados (c_j) considerados corrompidos em cada provedor.
- τ_{global} - Quantidade total de blocos codificados (c_j) considerados corrompidos.
- $\mathcal{S}_i$ - Representação de cada servidor que armazena os blocos codificados do arquivo **F**.

2.2 CÓDIGO CORRETOR DE ERROS

Códigos corretores de erros são baseados no seguinte princípio: adicionar redundância à informação com o objetivo de corrigir possíveis erros que podem ocorrer no processo de transmissão. Blocos redundantes são adicionados à informação original a fim de obter uma sequência de informações toleráveis à perda de uma fração da informação transmitida. Na Figura 2.1 é mostrada a forma sistemática deste conceito, onde as informações são os primeiros "k" blocos e a redundância, os"$n-k$" blocos seguintes. Desta forma, pode-se afirmar que a perda de até "$n-k$" blocos não interfere na recuperação da informação original.

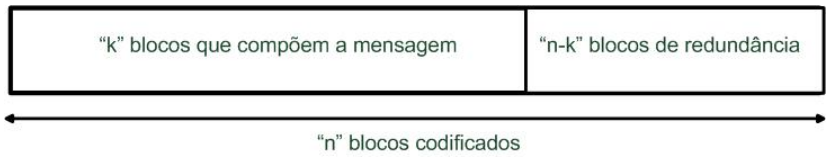

Figura 2.1: Representação simples de Código Corretor de Erros

Um modelo comum de comunicação entre duas partes é aquele onde a perda no canal de comunicação não supere uma fração δ da informação original. Então, um meio com capacidade de comunicação de $R = 1-\delta$ é estabelecido, onde $0 < \delta < 1$. Isso significa que uma mensagem de "n" blocos será codificada em n/R blocos, para que quaisquer "n" blocos recebidos possam ser usados para decodificar a informação original. O código *Reed-Solomon* (k,m) (REED; SOLOMON, 1960) é exemplo desta abordagem, o qual possui complexidade de codificação e decodificação da ordem de $\mathcal{O}(k(m-k)\log_2 m)$.

Um outro modelo de canal com perda limitada é o canal *erasure* apresentado por Elias (ELIAS, 1955), onde o autor mostra que existem códigos com taxa R (sendo $R < 1-\delta$, onde $0 < \delta < 1$) que pode ser usado para transmitir mensagens sob canais com capacidade de $1-\delta$, com tempo de codificação e decodificação da ordem de $\mathcal{O}(n \log n)$.

Com o objetivo de superar as restrições existentes nos Códigos *Reed-Solomon* e *Elias*, uma nova classe de códigos foi introduzida em (LUBY; MITZENMACHER; SHOKROLLAHI, 1998; LUBY, 1998), chamado *Tornado Codes*. Esta classe de códigos possui complexidade linear de tempo, tanto para a codificação como para a decodificação.

No que tange a robustez, é necessário uma classe de códigos com um modelo confiável de transmissão de informações em ambientes com perdas consideráveis, acima de 20%, e massivamente escalável. Para dar suporte a essas características, Códigos *Fountain* (MACKAY, 2005) é uma classe de códigos *erasure* com a propriedade de que uma sequência potencialmente ilimitada de blocos codificados podem ser gerados, os quais podem ser recuperados a partir de qualquer subconjunto de blocos codificados. Este subconjunto deve possuir um tamanho igual ou maior que o número de blocos que compõem o arquivo original. Para exemplificar, Códigos *Fountain* dividem a mensagem em "n" blocos em tamanho iguais e, que somados entre si de maneira aleatória, produzem uma quantidade ilimitada de blocos como resultado. Para recuperar os "n" blocos que compõem a mensagem, é necessário "x" blocos quaisquer, sendo $x \approx n$.

A primeira implementação de Códigos *Fountain* foi apresentada em (LUBY, 2002), chamado *LT-Codes*. Neste exemplo, a complexidade de codificação é $\mathcal{O}(\log(k/\delta))$ e de decodificação é $\mathcal{O}(k \log(k/\delta))$, onde "$k$" é o número de blocos do arquivo a ser processado e "δ" é o parâmetro de perda do meio de comunicação.

Uma proposta para melhorar o tempo de complexidade é mostrado em (SHOKROLLAHI, 2006), o *Raptor Codes* difere do *LT-Codes* adicionando uma camada de pré-codificação com os "k" blocos do arquivo, e produzindo uma quantidade "$k + \eta$" blocos. Esta quantidade intermediária é a fonte para o resultado final, que é realizada nos mesmos moldes que o *LT-Codes*. A complexidade de codificação e decodificação é $\mathcal{O}(1)$.

Para este trabalho foi utilizado o *Online Codes* (MAYMOUNKOV, 2002) para a implementação do código corretor de erros. Nas seções seguintes serão descritos em detalhes como a codificação e decodificação são realizadas.

2.2.1 ONLINE CODES

Online Codes são um tipo de *Códigos Fountain* na forma não-sistemática e inspirados no *Tornado Codes*, tendo complexidade de ordem constante, $\mathcal{O}(1)$, para a codificação e complexidade de ordem linear, $\mathcal{O}(n)$, para a decodificação. Uma análise detalhada de como estes resultados são alcançados é apresentada em (MAYMOUNKOV, 2002).

Os *Online Codes* funcionam de maneira modular com 2 fases distintas. Uma fase

intermediária de codificação, na qual o arquivo é expandido com a adição de blocos auxiliares. A fase final produz uma quantidade praticamente ilimitada de blocos codificados na forma não-sistemática.

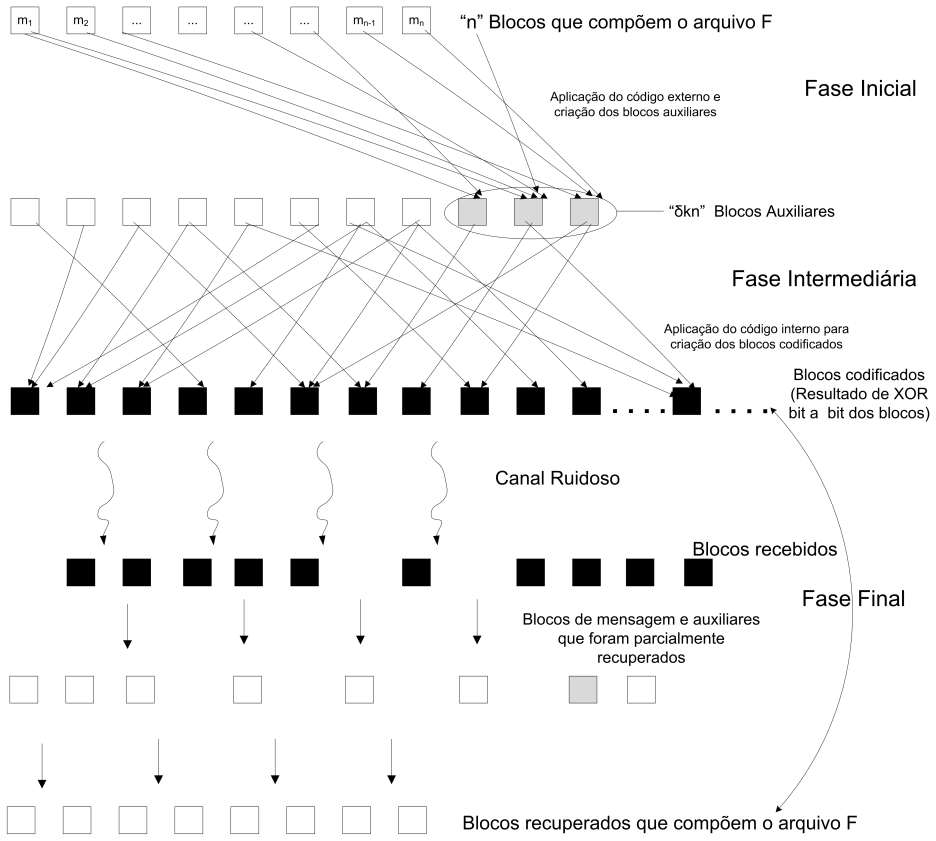

Figura 2.2: Visão Geral do *Online Codes* (MAYMOUNKOV, 2002)

As variáveis k, ϵ, δ e n, descritos na seção 2.1, são os parâmetros de entrada que determinam o comportamento das fases inicial, intermediária e final.

A fase inicial divide o arquivo F em "n" blocos de tamanhos iguais. A fase intermediária cria "δkn" blocos auxiliares, formando a mensagem composta de $(1+\delta k)n$ blocos. Para cada bloco de mensagem (m_i), k blocos auxiliares são escolhidos aleatoriamente e ao final desta fase, são realizadas operações XOR com seus respectivos conteúdos para a formação de cada bloco auxiliar.

A fase final cria uma quantidade ilimitada de blocos codificados a partir da mensagem composta. A geração dos blocos codificados é o resultado de operações XOR de uma quantidade de "d" blocos da mensagem composta. Esta variável "d" representa o grau de cada bloco codificado (c_j), que são escolhidos de acordo com a distribuição de probabilidade.

Embora apresentem semelhanças na composição dos blocos codificados, existem duas propriedades que diferenciam o *Online Codes* do *Tornado Codes*, a saber:

- *Local Encodability*: De acordo com esta propriedade, cada bloco codificado pode ser calculado independentemente de outros blocos, em um tempo constante. Com isso, é possível ter a reconstrução de um bloco corrompido sem a necessidade de efetuar o download do arquivo por completo.
- *Rateless-ness*: Em contraste com códigos de taxas fixadas, esta propriedade permite que cada arquivo tenha uma quantidade praticamente ilimitada de blocos codificados. Com isso, é possível gerar uma quantidade adicional de blocos codificados mediante algum imprevisto que possa prejudicar a disponibilidade do arquivo original.

Online codes são constituídos por três componentes distintos: um pré-codificador, um codificador e um decodificador. Esses componentes são explicados a seguir:

2.2.1.1. Pré-Codificador

O pré-codificador recebe como entrada um arquivo F composto de n blocos $\{m_i\}_{i=1}^{n}$ de tamanhos iguais e o transforma em um arquivo F' composto de $n' = (1 + k\delta)n$ blocos, para um valor k constante e um parâmetro positivo, onde $0 < \delta < 1$.

Os blocos auxiliares podem ser produzidos por duas maneiras distintas que estão descritas na seção 4.5 de (MAYMOUNKOV, 2002). Para este trabalho, a fase de pré-codificação adiciona "δkn" blocos auxiliares, formando uma mensagem composta F'. Cada bloco m_i terá um grau fixo "k" em relação aos blocos auxiliares, os quais serão escolhidos aleatoriamente. Este grau "k" se refere à quantidade de vezes que um bloco m_i será disponibilizado para a construção dos blocos auxiliares. Por conseguinte, cada bloco auxiliar é obtido pela realização de operações XOR, em média, em $1/\delta$ blocos de mensagens (m_i). Esta adição de blocos implica que a probabilidade de não recuperar a mensagem F é da ordem de δ^k, o qual pode em alguns casos atingir a ordem de $1/2^{30}$.

O objetivo deste passo inicial é possibilitar a recuperabilidade, com alta probabilidade, do arquivo original F com uma porção de $(1 - \delta)n'$ de blocos.

2.2.1.2. Codificador

Cada bloco codificado é obtido pela realização de operação XOR dos conteúdos de "d" blocos de mensagens escolhidos aleatorialmente da mensagem composta F'. O valor de "d" é conhecido como o grau de cada bloco codificado, sendo calculado de acordo com a distribuição de probabilidade $\rho_d = (\rho_1, \rho_2, \ldots, \rho_D)$, onde ρ_d é a probabilidade que um determinado bloco codificado tenha grau d, e D é uma constante que representa o grau máximo que um bloco codificado possa ter.

Para qualquer valor ϵ, δ, a distribuição de probabilidade ρ é definido pela equação 2.1:

$$\rho_1 = 1 - \frac{1 + 1/D}{1 + \epsilon} \quad \text{e} \quad \rho_d = \frac{(1 - \rho_1)}{(1 - 1/D)\, d\, (d - 1)}, \tag{2.1}$$

onde o grau máximo é definido pela equação 2.2:

$$D = \left\lceil \frac{\ln(\epsilon/2) + \ln(\delta)}{\ln(1 - \delta)} \right\rceil. \tag{2.2}$$

Por questões de mapeamento dos blocos, no *Hy-SAIL* cada bloco codificado possui a representação $\langle c, x \rangle$, onde c é o índice do bloco codificado e x é uma lista de blocos escolhidos aleatoriamente de F'.

2.2.1.3. Decodificador

O processo de decodificação consiste em recuperar os blocos m_i que compõem o arquivo F. Em um primeiro momento, são selecionados todos os blocos codificados com grau 1, cujo conteúdo são simples cópias de blocos m_i, e então são marcados como recuperados. O processo continua com a procura de blocos codificados, onde todos os blocos já estão recuperados, com a exceção de um deles. A recuperação deste bloco acontecerá com a operação XOR dos blocos já recuperados com o bloco codificado.

Supondo-se que já tenha recuperado os blocos m_{15}, m_{28}, m_{19} e o bloco codificado seja $cod_x = m_9 \oplus m_{15} \oplus m_{28} \oplus m_{19}$, ao realizar um XOR entre cod_x e os blocos já recuperados, teremos o conteúdo de m_9 como recuperado também.

Este passo será repetido continuamente até que todos os blocos de mensagens estejam recuperados, e partir deste momento é possível reconstruir o arquivo F.

Os seguintes teoremas apresentam as contribuições do *Online Codes*:

Teorema 1 *: Para qualquer mensagem F com n blocos de tamanhos iguais e quaisquer parâmetros $0 < \epsilon < 1$, $0 < \delta < 1$, existe uma distribuição de probabilidade ρ que pode recuperar uma fração $1 - \delta$ da mensagem original de $(1 + \epsilon)n$ blocos codificados em um tempo proporcional à $t \propto nln((ln\delta + ln(\epsilon/2))/ln(1 - \delta))$.*

Teorema 2 *: A mensagem pré-codificada F', composto por n blocos, pode ser recuperado de $(1 + \epsilon)n$ blocos codificados em um tempo de $t \propto nln(1/\epsilon)$, onde $\epsilon > 0$.*

2.3 CORPOS FINITOS

Para auxiliar nas operações com os arquivos armazenados pelo *Hy-SAIL*, são utilizados os conceitos de Álgebra Polinomial (MENEZES; OORSCHOT; VANSTONE, 2001). O arquivo original F é fragmentado de acordo com os parâmetros fornecidos no esquema exposto na seção 2.2.1. Cada fragmento é representado por um polinômio em um Corpo Finito para realizar operações ariméticas, com objetivo de verificar a integridade dos blocos codificados. Algumas definições e teoremas concernentes a este assunto são apresentados a seguir.

Definição 1 *: (Anel) Um anel $(\mathbb{R}, +, \times)$ é definido como um conjunto $\mathbb{R}$, com 2 (duas) operações : adição (representado por $+$) e multiplicação (representado por $\times$) , que sistemas os seguintes axiomas:*

- *$(\mathbb{R}, +)$ é um grupo abeliano com elemento identidade representado por 0.*
- *A multiplicação é associativa : $a \times (b \times c) = (a \times b) \times c$.*
- *Existe o elemento da identidade para a multiplicação, representado por 1, tal que $1 \times x = x \times 1 = x$, $\forall x \in \mathbb{R}$.*
- *Propriedade Distributiva: $a \times (b+c) = (a \times b) + (a \times c)$ e $(b+c) \times a = (b \times a) + (c \times a)$, $\forall a, b, c \in \mathbb{R}$.*

Um polinômio sobre qualquer corpo $\mathbb{F}$ é uma expressão matemática representada na equação 2.3:

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \ldots + f_1x + f_0 \tag{2.3}$$

onde x é indeterminado e os coeficientes $f_{n-1}, \ldots, f_0$ são elementos de $\mathbb{F}$ e os expoentes de x são números inteiros. O interesse deste trabalho é pelos polinômios sobre o Corpo Finito *GF(q)*.

O conjunto de todos os polinômios sobre *GF(q)* formam um anel se as operações de adição e multiplicação são definidas neste conjunto de polinômios. Este anel é definido por *GF(q)*[x]. A soma de 2 polinômios $f(x)$ e $g(x)$ em *GF(q)*[x], é outro polinômio em *GF(q)*[x] definido pela equação 2.4:

$$f(x) + g(x) = \sum_{i=1}^{\infty} (f_i + g_i)x^i \tag{2.4}$$

Como exemplo, sobre GF(2), tem-se a equação 2.5:

$$(x^5 + x^4 + x^2 + 1) + (x^4 + x + 1) = x^5 + (1+1)x^4 + x^2 + x + (1+1) = x^5 + x^2 + x \tag{2.5}$$

Definição 2 *(Polinômio Irredutível) Seja $p(x) \in \mathbb{F}[x]$ um polinômio tal que $deg[p(x)] \geq 1$. O polinômio $p(x)$ é um polinômio irredutível sobre F se não puder ser representado como o produto de 2 polinômios, de grau positivo, em $\mathbb{F}[x]$.*

Teorema 3 *Se $f(x)$ é irredutível sobre $\mathbb{F}[x]$, então $F[x]/\langle f(x)\rangle$ é um corpo finito.*

Para fins deste trabalho, o cálculo do resto da divisão polinomial torna-se mais conveniente para as questões de cálculo de integridade dos arquivos. Com isso, o Teorema 4 auxilia nesta construção.

Teorema 4 *Seja $d(x)$ um múltiplo de $g(x)$. Então para qualquer $a(x)$, tem-se a seguinte expressão: $a(x)$ mod $g(x) = [a(x)$ mod $d(x)]$ mod $g(x)$, com os seguintes desdobramentos:*

1. $[a(x) + b(x)] \mod d(x) = [a(x)] \mod d(x) + [b(x)] \mod d(x)$
2. $[a(x) \times b(x)] \mod d(x) = \{[a(x) \mod d(x)] \times [b(x) \mod d(x)]\} \mod d(x)$

Definição 3 *(Divisibilidade Polinomial) Sejam $p(x)$ e $f(x)$ polinômios pertencentes a $\mathbb{F}[x]$. Podemos afirmar que $p(x)$ divide $f(x)$, representado por $p(x)/f(x)$, se $f(x) \mod p(x) = 0$.*

A questão da congruência entre 2(dois) polinômios diferentes pode ser alcançada em um corpo $\mathbb{F}(x)$. Esta propriedade é interessante, pois podemos ter arquivos com conteúdos diferentes e ambos pertecerem ao mesmo elemento em $\mathbb{F}(x)$. A congruência entre polinômios distintos pode ser definida:

Definição 4 *(Congruência) Sejam $p(x)$ e $f(x)$ polinômios pertencentes a $\mathbb{F}[x]$. Podemos afirmar que $p(x)$ é congruente a "$f(x) \mod g(x)$", se $g(x)$ divide $p(x) - f(x)$, i.e., $p(x) \equiv f(x) \pmod{g(x)}$.*

Definição 5 *(Propriedades de Congruência) Sejam $g(x), h(x), g_1(x), h_1(x), s(x) \in \mathbb{F}[x]$. As seguintes propriedades são válidas:*

- *$g(x) \equiv h(x) \pmod{f(x)}$ se e somente se $g(x)$ e $h(x)$ possuem o mesmo na divisão por $f(x)$.*
- *(Reflexiva) $g(x) \equiv g(x) \pmod{f(x)}$.*
- *(Simétrica) Se $g(x) \equiv h(x) \pmod{f(x)}$, então $h(x) \equiv g(x) \pmod{f(x)}$.*
- *(Transitiva) Se $g(x) \equiv h(x) \pmod{f(x)}$ e $h(x) \equiv s(x) \pmod{f(x)}$, então $g(x) \equiv s(x) \pmod{f(x)}$*
- *Se $g(x) \equiv g_1(x) \pmod{f(x)}$ e $h(x) \equiv h_1(x) \pmod{f(x)}$, então $g(x) + h(x) \equiv g_1(x) + h_1(x) \pmod{f(x)}$ e $g(x)h(x) \equiv g_1(x)h_1(x) \pmod{f(x)}$.*

Definição 6 *Seja $\mathbb{F}[x]/\langle f(x)\rangle$ a representação de um conjunto de polinômios em $\mathbb{F}[x]$ de grau menor $f(x)$. As operações de adição e multiplicação nos polinômios são realizadas sobre $\pmod{f(x)}$.*

Teorema 5 *Seja $f(x) \equiv \mathbb{Z}_p[x]$ um polinômio irredutível de grau a. Então $\mathbb{Z}_p[x]/\langle f(x)\rangle$ é um corpo finito de ordem p^a. As operações de adição e multiplicação nos polinômios são realizadas sobre* $(\mathrm{mod}\ f(x))$.

Teorema 6 *Para cada $a \geq 1$, existe um polinômio irredutível de grau a sobre $\mathbb{Z}_p[x]$. Consequentemente, cada corpo finito possui uma representação polinomial básica.*

Definição 7 *Um polinômio irredutível $f(x) \in \mathbb{Z}_p[x]$ de grau a é um polinômio primitivo, se x é o gerador de $\mathbb{F}^*_{p^a}$, o grupo multiplicativo de todos os elementos, diferentes de zero, em $\mathbb{F}_{p^a} = \mathbb{Z}_p[x]/\langle f(x)\rangle$.*

Neste capítulo foi apresentado a fundamentação teórica de 2 conceitos importantes na construção do *Hy-SAIL*: código corretor de erros e álgebra polinomial. O *Online Codes* fornece uma flexibilidade na codificação e decodificação das mensagens. Os conceitos de álgebra polinomial ajudam na construção de código autenticadores de mensagens com características homomórficas. No capítulo seguinte, o conceito de prova de recuperabilidade é mostrado com seus principais requisitos, bem como exemplos de implementações já realizadas.

3 PROTOCOLOS DE PROVA DE RECUPERABILIDADE

Este capítulo apresenta o conceito de protocolos de Prova de Recuperabilidade, suas aplicabilidades e exemplos de implementações que nortearam este trabalho. Protocolos de Prova de Recuperabilidade (*PoR*) introduzidos em (JUELS; KALISKI, 2007), permitem que um usuário armazene um arquivo F em um provedor remoto e tenha garantias da recuperabilidade do conteúdo original, mesmo que haja a perda de uma fração do mesmo. Nestes protocolos, o usuário deve manter uma pequena quantidade ($\cong$ 3% do tamanho do arquivo original) de informações do arquivo armazenado remotamente e realizar consultas periódicas ao provedor, que deverá responder ao usuário com provas irrefutáveis da guarda do mesmo. A segurança destes protocolos reside na alta probabilidade de um mecanismo de recuperabilidade do arquivo F, a partir de provedores que tenham sido submetidos, e respondido de maneira satsifatória, a uma auditoria no que se refere à integridade do mesmo.

Um modelo simples de *PoR*, seria uma assinatura de um arquivo com o armazenamento da chave localmente. Para a obtenção de êxito no "desafio-resposta", o provedor receberia a chave e enviaria ao usuário a assinatura deste arquivo com chave recebida. Para esta abordagem, fica óbvio que bastaria uma única consulta para que o atacante armazenasse a assinatura do arquivo e em consultas posteriores, o atacante enviaria a assinatura sem a necessidade de consultar o conteúdo do arquivo. Para um caso prático e ideal, necessitamos de um modelo no qual o usuário poderá realizar uma quantidade ilimitada de consultas ao provedor com o mínino de custo computacional.

Um protocolo de *PoR* em sua forma elementar possui três fases: Codificação, Desafio e Resposta:

- **Codificação** : O usuário transforma o arquivo F em um arquivo F', após aplicação de um código corretor de erros. Isto garante que o arquivo F pode ser recuperado mesmo que uma fração "α" seja corrompida, onde $0 < \alpha < 1$. Geralmente, "α" é uma valor constante e o $|F'| = |F/(1-\alpha)|$.
- **Desafio**: Envia ao provedor as t posições (p_1,p_3,...,p_t) do arquivo F' que devem ser verificadas e uma chave ϱ para cálculo de uma função de *hash* para otimizar

o consumo de largura de banda ao enviar a resposta. O provedor captura as posições F' = $F'[p_1]||F'[p_3]||\ldots||F'[p_t]$ e calcula a resposta, Resp = $hash_{\varrho}$(F').

- **Resposta**: O provedor envia a resposta *Resp* ao usuário, que compara com o resultado que possui armazenado.

Neste contexto, é possível perceber que os esquemas de *PoR* tem como prioridade a minimização de armazenamento de informações dos arquivos tanto por parte do usuário como do provedor, uma complexidade mínima na comunicação para os protocolos de auditoria e o acesso à uma quantidade razoável de blocos dos arquivos, por parte do *Verificador*, para a execução de auditoria na integridade dos mesmos.

A Figura 3.1 representa um modelo de prova de recuperabilidade. Um algoritmo de codificação transforma o arquivo F em um arquivo codificado F_{cod} para ser armazenando junto ao Servidor. Este algoritmo de codificação utiliza uma chave que será armazenada pelo Usuário. Esta chave não possui nenhuma relação com o protocolo de prova de recuperabilidade. Para se certificar da integridade e recuperabilidade do arquivo F, o Usuário realiza um "desafio-resposta" com o Servidor.

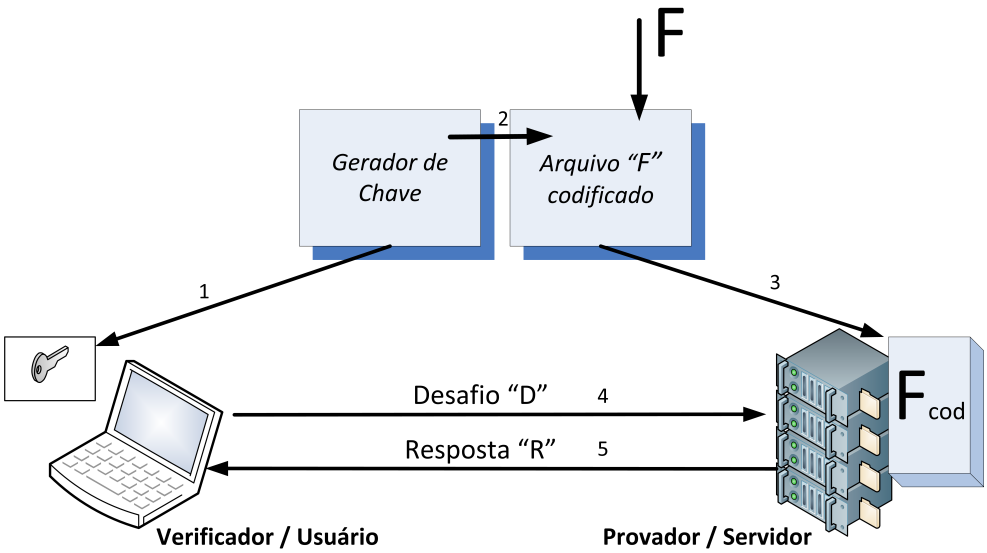

Figura 3.1: Esquema do protocolo de prova de recuperabilidade

Os protocolos de prova de recuperabilidade podem ser implementados em 2 modelo diferentes: limitado e ilimitado. Em um modelo ilimitado, o usuário não é obrigado a prever a quantidade de vezes que a integridade dos arquivos é verificada, porém as soluções apresentadas em (SHACHAM; WATERS, 2008) exigem uma quantidade

considerável de armazenamento extra de informações do arquivo. No modelo limitado, semelhante ao que é apresentado no *HAIL* (BOWERS; JUELS; OPREA, 2009a), o usuário armazena uma quantidade de códigos autenticador de mensagem dos blocos do arquivo e mediante algumas propriedades de homomorfismo, poderá realizar uma quantidade limitada de consulta de integridade aos arquivos.

O *MAC* proposto pelo *Hy-SAIL* (MAC_{hysail}) permite uma flexibilidade no esquema a ser utilizado, apresentando tanto o modelo limitado quanto o ilimitado de consulta de integridade. A complexidade de comunicação e armazenamento de informações dos arquivos torna este protocolo atrativo para a distribuição de dados em ambientes de computação em nuvem.

3.1 TRABALHOS RELACIONADOS

A medida que o tema computação em nuvem ganha espaço, é natural que haja uma preocupação com os desafios de lidar com dados em ambientes distribuídos (DILLON; WU; CHANG, 2010), principalmente no que se refere a confidencialidade, integridade e disponibilidade (KAMARA; LAUTER, 2010).

Neste cenário, a principal utilização deste modelo de computação distribuída é a possibilidade de lidar com grande quantidade de dados, processando a baixo custo em tempo considerado razoável (SHVACHKO et al., 2010; DEAN; GHEMAWAT, 2008b), e a oportunidade de armazenar informações em locais diversos, seja através de redes *P2P* ou unidades de armazenamento remotas, conferindo níveis de disponibilidade (UNDHEIM; CHILWAN; HEEGAARD, 2011) semelhantes aos que são exigidos de negócios com missões críticas, como comércio eletrônico, sites governamentais e serviços bancários.

A utilização de redes *P2P* para armazenamanto de *backups* com computadores conectados a Internet e trabalhando cooperativamente é apresentado em (LILLIBRIDGE et al., 2003). Cada computador é um nó, o qual possui um conjunto de parceiros que coletivamente armazenam parte de seus dados. O código corretor de erros *Reed-Solomon* (REED; SOLOMON, 1960) é usado para garantir a redundância dos dados, através da expansão de um arquivo de "k" blocos em "$k + m$" blocos, permitindo que uma falha em qualquer "m" dos "$k + m$" blocos não prejudique a recuperabilidade do arquivo. Sendo que os pares não são considerados maliciosos.

Para verificar a integridade de dados armazenandos remotamente, a comunidade de criptografia tem apresentado novos conceitos para atender esta demanda, os mais conhecidos são Prova de Recuperabilidade (PoR) (JUELS; KALISKI, 2007), (SHACHAM; WATERS, 2008), (BOWERS; JUELS; OPREA, 2009b), (BOWERS; JUELS; OPREA, 2009a), (DODIS; VADHAN; WICHS, 2009), Prova de Possessão de Dados (PDP) (ATENIESE et al., 2007), Prova de Propriedade (PoW) (HALEVI et al., 2011). Embora tenham a mesma finalidade, eles diferem na capacidade de assegurar a recuperação dos arquivos.

O conceito de *PoR* é definido em (JUELS; KALISKI, 2007), onde os autores apresentam um modelo limitado de consultas de integridade dos arquivos com o uso de "sentinelas" ao longo dos mesmos, que otimizam o uso de largura de banda no *PoR* proposto. Neste modelo, os "sentinelas" são inseridos de maneira aleatória, após a aplicação do código corretor de erros e no momento da verificação, são questionados se as posições persistem. Caso não haja mudança significativa nas posições "sentinelas", o arquivo é considerado recuperável. Como a verificação de integridade ocorre em uma pequena porção do arquivo, a recuperabilidade pode ser afetada se ocorrer alguma alteração no arquivo que não coincida com a posição de um dos "sentinelas".

Em (SHACHAM; WATERS, 2008), os autores propõem dois esquemas de *PoR* com definições formais de segurança, o primeiro possui a segurança demonstrada com o conceito de emparelhamento bilinear (BONEH; LYNN; SHACHAM, 2001) no modelo de oráculo aleatório (BELLARE; ROGAWAY, 1993), e o segundo usa uma função pseudo-aleatória (*PRF*) e é seguro no modelo padrão, na qual o adversário é limitado pelo tempo e poder computacional disponível para realizar seus ataques ao critposistema (CRAMER; SHOUP, 1998).

Seguindo a mesma abordagem, Bowers *et al.* (BOWERS; JUELS; OPREA, 2009b) propõem um *framework* teórico que suporta um modelo adversarial Bizantino e aprimoram os modelos de Juels-Kaliski (JUELS; KALISKI, 2007) e Shacham-Waters (SHACHAM; WATERS, 2008) limitando a taxa de erro do adversário.

Uma abordagem prática de um modelo de *PoR* é mostrado em (BOWERS; JUELS; OPREA, 2009a), o *HAIL* possui um modelo adversarial mais detalhado, onde os atacantes podem corromper servidores e alterar blocos dos arquivos durante uma rodada de tempo pré-determinado. O *HAIL* exige um custo computacional e de largura de banda menor que os PoR's já propostos. Embora tenha uma proposta de proteger

elementos estáticos, com pequenas modificações, o *HAIL* pode atender a demanda em assegurar a integridade de arquivos que possuem alterações dinâmicas. Além de fazer distinção entre servidores que armazenam os arquivos, o *HAIL* não possui um modelo de consulta ilimitada de verificação de integridade.

Uma outra visão teórica de *PoR* é apresentada em (DODIS; VADHAN; WICHS, 2009), onde os autores usam noções de dificuldade de amplificação, da área de teoria da complexidade computacional, para propor esquemas de *PoR*'s visando atender problemas abertos em abordagens de armazenamento em nuvem. Um primeiro esquema se baseia na construção do esquema de Juels-Kaliski (JUELS; KALISKI, 2007), fazendo uma variante do modelo limitado e provando formalmente a segurança sem nenhuma suposição sobre o comportamento do adversário. O modelo ilimitado, onde a complexidade de comunicação é linear e sua segurança não se baseia em oráculos aleatórios, abordando um problema em aberto exposto em (SHACHAM; WATERS, 2008).

Outro conceito a respeito de armazenamento em sites remotos é definido em (ATENIESE et al., 2007) , onde os autores abordam um modelo de Prova de Posse de Dados (*PDP*), o qual se baseia em uma demonstração probabilística de que servidores não confiáveis de fato armazenam o arquivo original. Em uma abordagem semelhante a (JUELS; KALISKI, 2007), o servidor acessa uma pequena porção do arquivo para fornecer provas da originalidade do mesmo. O avanço desta abordagem é a quantidade constante de dados armazenados pelo usuário e o tráfego da ordem de *Kilobits* para o protocolo de "desafio-resposta". Em (ZHU et al., 2010), o conceito de *PDP* é aplicado para uma abordagem híbrida, com nuvens privadas e públicas armazenando dados.

Em (HALEVI et al., 2011), os autores apresentam o conceito de Prova de Propriedade (*PoW*), onde existe uma inversão de papéis no momento da verificação de posse dos arquivos. A preocupação desta abordagem consiste em inibir a ação do atacante em efetuar o download do arquivo do servidor, possuindo apenas o conhecimento de uma pequena informação do mesmo.

Nos esquemas de *PoR*, a tarefa de auditoria dos dados armazenandos pode ser delegada à uma terceira parte confiável a fim de evitar possíveis sobrecargas no processamento e armazenamento das estações dos usuários (WANG et al., 2010). Em (WANG et al., 2009) é apresentado um esquema com este artifício, que além de localizar quais servidores possuem dados corrompidos, suporta alterações dinâmicas nos blocos dos arquivos, como atualizar, adicionar e apagar os dados. Em (WANG et al., 2009), os autores

propõem a combinação assinatura de emparelhamento bilinear com a manipulação de árvores de *Merkle* para suportar uma auditabilidade pública e o dinamismo dos dados.

As abordagens atuais de verificação de integridade de arquivo (PoR, PDP, PoW) não conseguem unificar uma solução de consulta ilimitada com baixo custo de processamento, armazenamento e largura de banda. Do ponto de vista prático, para cada abordagem utilizada é necessário uma avaliação de qual item será sub-otimizado em prol de um modelo considerado aceitável. A necessidade de efetuar o *download* dos blocos para a recompô-los é uma limitação em todos os modelos utilizados. A falta de elasticidade em adicionar unidades de armazenamento às camadas de abstração, criadas pelos modelos de PoR's, é outra restrição que contraria ao paradigma de computação em nuvem.

3.2 MODELO DE JUELS-KALISKI

Em (JUELS; KALISKI, 2007), Juels-Kaliski apresentam o conceito de *Prova de Recuperabilidade* baseado na inserção de "sentinelas" ao longo do arquivo a ser armazenado remotamente. Neste modelo, o usuário armazena uma única chave criptográfica, como também uma pequena quantidade de informações do arquivo armazenado. De maneira contundente, a unidade de armazenamento acessa uma pequena porção do arquivo $\tilde{F}$ (arquivo F cifrado com os "sentinelas" distribuídos) para provar a integridade do mesmo. A porção acessada pela unidade verificadora independe do tamanho do arquivo, e de modo geral, engloba uma quantidade de blocos do arquivo que não possuem nenhuma relação com o arquivo original.

De modo geral, esta abordagem obedece aos modelos gerais de PoR's e possui a seguinte estrutura: Codificação, Decodificação, Desafio, Resposta e Verificação. Para efeito de desenvolvimento, é de se considerar que o arquivo F é constituído de b blocos, possuindo a seguinte representação: $F[1], \ldots, F[b]$.

A função de *Codificação* possui quatro fases distintas para a obtenção de um arquivo "F" cifrado com "sentinelas" posicionados aleatoriamente, com isto a unidade de armazenamento não consegue fazer distinção entre o conteúdo do arquivo original e os sentinelas. Desta maneira, se houver alguma modificação em uma fração-γ do arquivo "F", é possível detectar a modificação de uma fração-γ dos "sentinelas". Vale lembrar

que a probabilidade de detecção está diretamente ligada à quantidade de "sentinelas" inseridas ao longo do arquivo.

A seguir, a descrição de cada fase da função de Codificação:

- Código Corretor de Erros: O arquivo é dividido e cada fragmento possui k-blocos. Para cada fragmento são aplicados o código corretor de erros (n, k, d). Esta operação expande cada fragmento a um tamanho de n blocos, gerando o arquivo $F' = F'[1], ..., F'[b']$, onde $b' = bn/k$ blocos.

- Cifragem dos blocos: O arquivo F' é cifrado usando uma chave simétrica E, gerando o arquivo F''. Um dos requisitos do protocolo consiste na habilidade de decifrar os blocos de modo independente, permitindo a recuperação do arquivo F caso algum bloco esteja corrompido. A razão de cifrar o arquivo F' é a indistiguibilidade criada entre os "sentinelas" e o conteúdo do arquivo original F.

- Criação dos Sentinelas: Seja $f(x) : \{0,1\}^j \times \{0,1\}^* \longrightarrow \{0,1\}^l$ uma função de uma única via. Um conjunto "s" sentinelas são calculados $\{a_w\}_{w=1}^{s}$, sendo $a_w = f(\kappa, w)$. Esses "sentinelas" são inseridos ao longo do arquivo F'', gerando o arquivo F'''. Portanto, é possível realizar até $\lfloor s/q \rfloor$ desafios, cada um com q consultas.

- Permutação: Seja $g(x) : \{0,1\}^j \times \{1, \ldots, b'+s\} \longrightarrow \{1, \ldots, b'+s\}$ uma função de permutação pseudo-aleatória, que é aplicada para permutar os blocos do arquivo F''', gerando o arquivo $\tilde{F}$. Esta função é necessária para criar uma aleatoriedade na distribuição dos "sentinelas". Então ao término desta fase, é gerado a seguinte representação: $\tilde{F}[i] = F'''[g(\kappa, i)]$

Ao ser codificado, o arquivo F está apto a ser submetido a verificações sobre a integridade do mesmo e ser decodificado, caso haja necessidade. Segue as descrições das funções de decodificação, verificação, desafio e resposta:

- *Decodificação:* Esta função, realizada pelo usuário, requer uma quantidade mínina de blocos de $\tilde{F}$, de acordo com o código corretor de erros. Executa a operação inversa de *Codificação*. Realiza a função inversa de permutação de acordo com a função g^{-1}, retira os sentinelas, decifra com a chave simétrica E e aplica o código corretor de erros para a recomposição do arquivo F.

- *Desafio:* Esta função possui um único dado de entrada, a variável σ, um contador inicialmente definido com 1(um). Esta variável fornece a posição do σ^{esimo} "sentinela" através da função g, $p = g(b' + \sigma)$. Este processo é repetido q vezes, gerando posições para diferentes q "sentinelas".

- *Resposta:* A unidade de verificação recebe como desafio um conjunto de q posições, determina os valores correspondentes dos q blocos, e os envia para o usuário.

- *Verificação:* Esta função compara o valor recebido do desafio enviado, validando ou não as posições dos "sentinelas".

Este modelo utiliza apenas uma camada de código corretor de erros, o que pode afetar na recuperabilidade do arquivo original. Na fase de Desafio, o usuário efetua o *download* dos valores dos "sentinelas" do servidor e verifica a corretude destes valores. Neste contexto, o modelo adversarial é baseado no comportamento do adversário de não ultrapassar os limites impostos pelo fragmentos criados na aplicação do código corretor de erros. Este modelo é resiliente contra um adversário que corrompe uma pequena fração γ dos blocos, o que não comprova a eficiência quanto a recuperabilidade do arquivo original.

Vantagens e Desvantagens do Modelo de Juels-Kaliski:

A) Vantagens:

- Armazena uma única chave simétrica para a execução do *PoR*.

- Pouco tráfego de informações na execução do *PoR*, apenas o conteúdo dos "sentinelas".

B) Desvantagens:

- Quantidade de consultas a ser realizadas está diretamente ligada à quantidade de "sentinelas", o que pode impactar no tamanho final do arquivo a ser armazenado remotamente.

- Não permite a reconstrução *online* dos blocos corrompidos do arquivo. Caso houver algum problema, o usuário deverá efetuar o *download* do arquivo, decodificá-lo e repetir a função de Codificação.

- Acessa pouca informação do arquivo para a execução do PoR, apenas os valores retornados pelos "sentinelas".

3.3 MODELO DE SHACHAM-WATERS

Em (SHACHAM; WATERS, 2008), são apresentados esquemas de PoR com provas formais de segurança contra adversários arbitrários no modelo Juels-Kaliski. O primeiro esquema, construído de assinaturas com emparelhamento bilinear (BONEH; LYNN; SHACHAM, 2001) e segurança formal baseada no modelo de oráculo aleatório (BELLARE; ROGAWAY, 1993) possui um tamanho menor para a consulta e a resposta que qualquer PoR com verificação pública. O segundo esquema, construído com funções pseudo-aleatórias (PRF) e segurança formal baseada no modelo padrão possui a menor resposta de qualquer PoR com verificação privada. A semelhança entre os dois esquemas apresentados residem em propriedades homomórficas para agregar os valores de autenticação de cada bloco em um único valor.

Neste modelo, o usuário fragmenta o arquivo em n blocos $m_1, \ldots, m_n \in \mathbb{Z}_p$, para um número primo grande p. Em seguida, utiliza o código corretor de erros para garantir a recuperabilidade do arquivo, caso haja algum tipo de perda.

3.3.1 VERIFICAÇÃO PRIVADA

Para uma abordagem com verificação pública de integridade dos blocos gerados, o usuário autentica cada bloco (m_i) da seguinte maneira: é escolhido um valor aleatório $\alpha \in \mathbb{Z}_p$ e uma função pseudo-aleatória f com uma chave k. Estes valores são as chaves privadas do usuário, e servirão na etapa de "desafio-resposta" para verificação da integridade de um conjunto de blocos m_i. O cálculo do valor dos códigos autenticadores de cada bloco é realizado com base na equação 3.1:

$$\sigma_i = f_k(i) + \alpha.m_i \in \mathbb{Z}_p \tag{3.1}$$

Cada bloco m_i e seus autenticadores σ_i são armazenados no provedor. Para a realização do protocolo de prova de recuperabilidade, o verificador escolhe i índices juntamente i coeficientes em $\mathbb{Z}_p$ como desafios aleatórios. Estes índices se referem a quais blocos serão utiizados na verificação de integridade. Então o verificador envia um conjunto de

i indices e os coeficientes (ν_i) correspondentes aos blocos escolhidos, $\{(i, \nu_i)\}$. Como resposta o provador calcula duas variáveis, como mostrados nas equações 3.2 e 3.3.

$$\sigma \leftarrow \sum_{(i,\nu_i)} \nu_i.\sigma_i \qquad (3.2) \qquad\qquad \mu \leftarrow \sum_{(i,\nu_i)} \nu_i.m_i \qquad (3.3)$$

O verificador averigua se a resposta enviada pelo provador está correta, realizando o seguinte cálculo mostrado na equação 3.4.

$$\sigma \stackrel{?}{=} \alpha.\mu + \sum_{(i,\nu_i)} \nu_i.f_k(i) \qquad (3.4)$$

Esta abordagem exige pouco consumo na largura de banda na resposta, porém o tamanho do desafio enviado ao provador consiste em um conjunto de i coeficientes pertencentes a $\mathbb{Z}_p$. Outro ponto se refere a recuperabilidade dos arquivos armazenados remotamente. A segurança dessa abordagem está fundamentada no Teorema 7 com a prova formal baseado em Teoria dos Jogos, como explicado na subseção 4.1 em (SHACHAM; WATERS, 2008).

Teorema 7 *Se o esquema do código autenticador de mensagem não for falsificável, o esquema de cifragem simétrica é semanticamente seguro e a função pseudo-aleatória é segura. Então, exceto com probabilidade desprezível, nenhum adversário poderá burlar o esquema de verificação privada e consequentemente o protocolo de prova de recuperabilidade, exceto que possua valores corretos de μ_j e σ, o que será possível utilizando os valores das chaves privadas α e k.*

3.3.2 VERIFICAÇÃO PÚBLICA

Este abordagem é semelhante à apresentada na seção 3.3.1, a diferença reside na construção dos códigos autenticadores que são baseados em assinaturas com emparelhamento bilinear (BONEH; LYNN; SHACHAM, 2001) que podem ser verificados publicamente. A estrutura dessas assinaturas permitem que as mesmas possam ser agregadas em uma combinação linear com a mesma abordagem utilizada na verificação privada.

Seja $e : GXG \rightarrow G_T$ uma mapeamento bilinear computável com os elementos de G pertencentes a $\mathbb{Z}_p$. A chave privada do usuário é $x \in \mathbb{Z}_p$ e a chave pública é $v = g^x$ junto com outro gerador $u \in G$. A assinatura σ_i de cada bloco m_i é igual a $[H(i)u^{m_i}]^x$. O verificador utiliza a mesma metodologia da abordagem de verificação privada, onde são escolhidos i índices com i coeficientes em $\mathbb{Z}_p$ como desafios aleatórios. Ao receber este conjunto Q $= \{(i, \nu_i)\}$ como desafio, o Provador calcula duas variáveis e envia de volta ao Verificador. Segue as duas variáveis em 3.5 e 3.6:

$$\sigma \leftarrow \prod_{(i,\nu_i)\in Q} \sigma_i^{\nu_i} \qquad (3.5) \qquad\qquad \mu \leftarrow \sum_{(i,\nu_i)\in Q} \nu_i.m_i \qquad (3.6)$$

O verificador, ao receber as 2 variáveis resultantes das equações 3.5 e 3.6, averigua a igualdade conforme demonstrado na equação 3.7:

$$\sigma \stackrel{?}{=} e(\prod_{(i,\nu_i)\in Q} H(i)^{\nu_i}.u^{\mu}, v) \qquad (3.7)$$

Esta abordagem com verificabilidade pública funciona da seguinte maneira: a chave privada x é exigida para gerar os códigos autenticadores σ_i, mas a chave pública é suficiente para o verificador no protocolo de prova de recuperabilidade. A segurança desta abordagem está fundamentada no Teorema 8 com a prova formal baseado em Teoria dos Jogos, como explicado na subseção 4.2 em (SHACHAM; WATERS, 2008).

Teorema 8 *Se o esquema de assinatura usados pelos arquivos não for falsificável e o problema computacional de Diffie-Hellman for difícil de ser solucionado em grupos bilineares, então, no modelo de oráculo aleatório, exceto com probabilidade desprezível, nenhum adversário poderá burlar o esquema de verificação pública e consequentemente o protocolo de prova de recuperabilidade, exceto que possua valores corretos de μ_j e σ.*

Vantagens e Desvantagens do Modelo de Shacham-Waters:

A) Vantagens:

- Possui um esquema de verificabilidade pública, o qual o usuário pode delegar a uma terceira parte a tarefa de realizar consultas quanto a integridade do arquivo armazenado.

- No esquema de verificação pública, o tamanho do desafio e da resposta são da ordem de 20 e 40 bytes, respectivamente.

B) Desvantagens:

- O tamanho do código autenticador de mensagem corresponde ao tamanho da mensagem, o que inicialmente dobra o tamanho do espaço de armazenamento exigido pelo modelo.
- Enquanto que no esquema de verificação privada, o desafio enviado ao provador é igual ao tamanho do autenticador multiplicado pela quantidade de blocos que serão verificados.

3.4 MODELO DE BOWERS-JUELS-OPREA

Em (BOWERS; JUELS; OPREA, 2009a), os autores procuram unificar dois pontos de extrema importância no que tange ao armazenamento de arquivos remotamente: integridade e disponibilidade. O *HAIL* (*High-Availability and Integrity Layer*) gerencia essas necessidades com provedores de armazenamento explorando redundâncias não apenas dentro do provedor, mas de maneira interligada como mostrado na Figura 3.2.

Inicialmente, o arquivo F é particionado em l fragmentos $F^1, \ldots, F^l$ e distribuído nos servidores primários $S_1, \ldots, S_l$ respectivamente. Neste momento, temos os arquivos sem nenhum tipo de codificação armazenados nos servidores primários. Então, cada fragmento F^l é codificado através de um código corretor de erros para garantir robustez contra possíveis problemas que possa ocorrer em qualquer servidor S_l. Deste modo, o adversário poderá corromper uma fração substancial de F^l, e mesmo assim poderá ser recuperado. Em seguida são aplicados *códigos de dispersão* que criam blocos de paridade dos arquivos F^l, a natureza sistemática destes códigos permitem que estes blocos criados adicionalmente sejam distribuídos nos servidores secundários $S_{l+1}, \ldots, S_n$.

No *HAIL*, estas duas camadas de codificação utilizam códigos Reed-Solomon (REED; SOLOMON, 1960) como código corretor de erros. A distinção básica entre servidores primários e secundários reside nos dados que armazenam, enquanto os servidores primários armazenam arquivos em texto claro, os servidores secundários armazenam arquivos codificados.

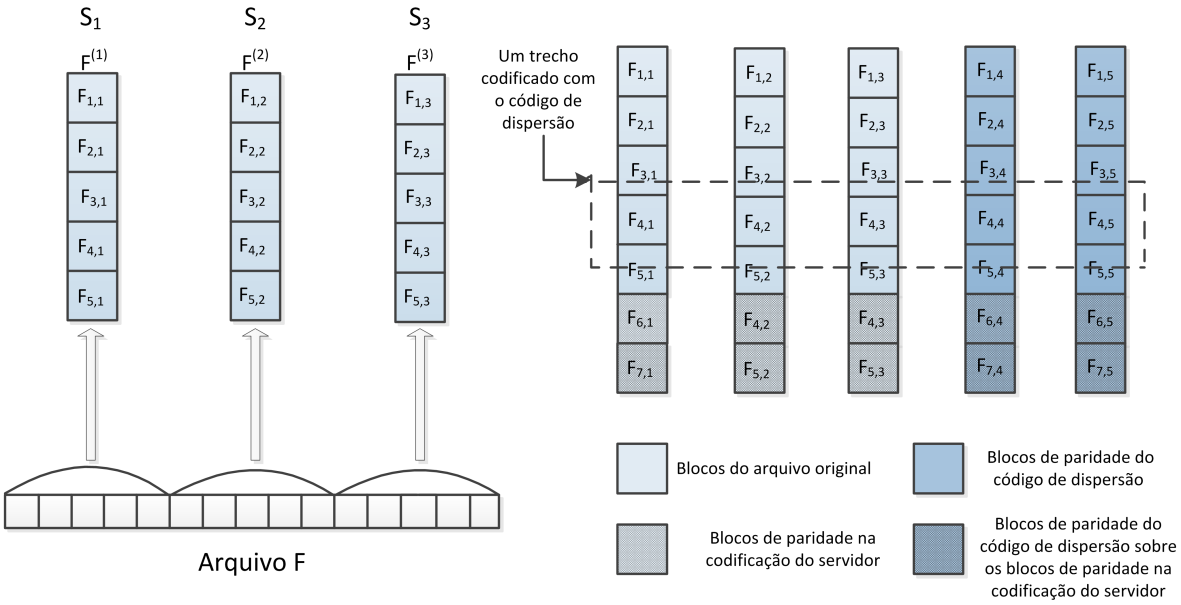

Figura 3.2: Codificação do arquivo F: A esquerda, o arquivo original é representado com uma matriz; a direita, o arquivo codificado com blocos de paridade adicionados tanto na codificação dos servidores como na codificação de dispersão (BOWERS; JUELS; OPREA, 2009a)

Com a distribuição do arquivo em diferentes servidores, esta abordagem busca obter resiliência contra adversários móveis. Este tipo de adversário tem a capacidade de corromper todos os servidores em um determinado período de tempo, e torna o arquivo F irrecuperável. Uma das restrições impostas pelo *HAIL* contra adversários móveis é o que o mesmo pode controlar apenas b de n servidores (primários e secundários) dentro de um determinado período de tempo.

Outra contribuição importante que esta abordagem fornece é a garantia que os fragmentos F^l podem ser recuperados com altíssima probabilidade. Para tal, as respostas são compostas homomorficamente no protocolo de "desafio-resposta" e calculadas a medida que os blocos e respectivos desafios são submetidos ao provador.

Tipicamente, o *MAC* é adicionado ao final na mensagem. Nesta abordagem, os autores propuseram uma primitiva criptográfica (*IPH-ECC*) que funciona tanto como MAC como código corretor de erros. Para um embasamento teórico, segue a definição do *IPH-ECC*:

Definição 8 *Um IPH-ECC com parâmetros (n, k, d) é definido como um código corretor de erros que codifica mensagem de k símbolos em palavras codificadas de n símbolos*

usando uma chave secreta k e possui distância mínima igual a d. Possui uma propriedade adicional que as palavras codificadas $(c_1, \ldots, c_n)$ *é um MAC homomórfico na mensagem* $(m_1, \ldots, m_k)$*.*

O *IPH-ECC* é construído com base em um código corretor de erros *Reed-Solomon*(REED; SOLOMON, 1960) com parâmetros (n, k, d) seguido de uma função *PRF*. Intuitivamente, o código *Reed-Solomon* é aplicada na mensagem $M = (m_1, m_2, \ldots, m_k)$, seguido de uma função PRF g. Em particular, a representação de $\vec{\kappa} = (\kappa_1, \kappa_2, \ldots, \kappa_n)$ e do mesmo modo para $\vec{\kappa'}$.

Sendo H a função que executa ambas as ações (*Reed-Solomon* e *PRF*), as equações 3.8 e 3.9 definem a codificação proposta pelo *IPH-ECC* da mensagem M e o código identificador, sendo $i = [1, n]$:

$$H_{\vec{\kappa},\vec{\kappa'}}(M, \tau) = (c_1, \ldots, c_n) \tag{3.8}$$

$$c_i = RS - UHF_{\kappa_i}(M) + g_{\kappa'_i}(\tau) \tag{3.9}$$

Em uma versão sistemática da função H, a qual é utilizado no *HAIL*, a função *PRF* é aplicada somente aos blocos de paridade. Então, ocorre uma divisão na distribuição dos blocos gerados onde os servidores primários armazenam os blocos $c_i = m_i$ para $i = [1, k]$ e os servidores secundários armazenam $c_i = RS - UHF_{\kappa_i}(M) + g_{\kappa'_i}(\tau)$ para $i = [k + 1, n]$.

De modo geral, o protocolo desafio-resposta implementado pelo *HAIL* verifica a integridade de um subconjunto aleatório de filas $D = i_1, \ldots, i_v$ na matriz que é formada de acordo com a Figura 3.2. O desafio do cliente consiste em uma semente κ_c, que cada servidor usará para derivar um conjunto D como também um valor único u. Seguem os passos da execução do desafio-resposta submetido ao provedor:

1. O cliente envia um desafio κ_c para todos os servidores.
2. Ao receber o desafio, o servidor S_j obtém um conjunto $D = i_1, \ldots, i_v$ como também um valor $u \in I$. A resposta do servidor S_j é $R_j = RS{-}UHF_v(F^d_{i_1 j}, \ldots, F^d_{i_v j})$.

3. O cliente executa o algoritmo de composição linear do código de dispersão em $(R_1, \ldots, R_n)$. Se o algoritmo fornecer $(\vec{m}, 0)$ como resposta, então a verificação falha e $(\kappa, j, \{\kappa_c, R_i\}_{i=1}^{n})$ retorna 0 para todo j.

4. Por outro lado, seja $(\vec{m}, 1)$ o valor fornecido pelo algoritmo de composição linear. O algoritmo de verificação $(\kappa, j, \{\kappa_c, R_i\}_{i=1}^{n})$ fornece 1 se:

   - $m_j = R_j$, para todo $j \in [1, l]$; ou
   - R_j é um MAC composto válido em $\vec{m}$ com chaves (κ_j, κ'_j) e coeficientes $\{\alpha\}_{i=1}^{v}$ para todo $j \in [l+1, n]$

Vantagens e Desvantagens do Modelo de Bowers-Juels-Oprea:

A) Vantagens:

- Protege contra qualquer mudança que houver nos blocos do arquivo distribuídos nos servidores, até mesmo a mudança em um único *bit* é detectado.
- Em comparação com os *PoRs* anteriores, *HAIL* elimina a necessidade de *check values* e reduz a expansão do arquivo dentro dos servidores.
- Protege contra um adversário que é ativo, o qual pode corromper servidores e alterar arquivo, e móvel, o qual é capaz de corromper progressivamente os provedores de armazenamento em diferentes momentos.

B) Desvantagens:

- Possui um critério para armazenamento de dados, onde os dados em texto claro são armazenados nos servidores primários e os blocos codificados, nos servidores secundários.
- Caso seja detectado que algum servidor apresente blocos corrompidos, o mecanismo de redistribuição exige que o usuário efetue o download do arquivo, decodifique-o, recupere-o e o redistribua entre o servidores.
- Não possui um modelo para consultas ilimitadas, o que pode inviabilizar uma implementação para sistemas de *backups* onde os arquivos são armazenados por longo período de tempo.

No capítulo 4, é apresentada uma visão geral da construção do *Hy-SAIL*, como os modelos de consulta Limitada e Ilimitada de verificação de integridade são abordados.

4 VISÃO GERAL DO Hy-SAIL

Em um protocolo de *PoR*, o arquivo é codificado antes de ser transmitido para o provedor de armazenamento. Esta codificação permite que uma parte do arquivo possa ser corrompida e a recuperabilidade do mesmo não seja afetada. O *PoR* em sua essência procura economizar largura de banda, processamento e armazenamento nas tarefas de desafio-resposta, garantindo que o arquivo esteja disponível e íntegro nos provedores de armazenamento remotos. Com a popularização do conceito de *Software-as-a-Service* (GAO et al., 2011), são ofertados uma ampla variedades de aplicativos a serem executados através da *Internet* com o armazenamento de seus dados em provedores remotos.

Neste contexto, qualquer usuário que armazene dados em um provedor remoto, deseja obter o mesmo nível de confidencialidade e integridade que possui ao armazenar seus dados em discos rígidos locais, e um nível maior de disponibilidade e recuperabilidade, supondo que não haja nenhum problema de conectividade. Para atender estas demandas, o provedor de armazenamento deve ter uma metodologia de controle de acesso aos seus arquivos, para evitar que os dados armazenados não fiquem sujeitos a qualquer problema de vazamento de informações. E por final, é necessário ter um protocolo de verificação de integridade para se assegurar que o arquivo não sofreu nenhuma alteração, de modo a prejudicar a recuperabilidade do mesmo.

O *Hy-SAIL*(*Hyper Scalability, Avalaibility, Integrity Layer*) é apresentado focando em duas importantes preocupações para armazenamento de dados em provedores remotos: disponibilidade e integridade. Quanto à disponibilidade, foi implementado um nível de redundância com duas camadas complementares: uma interna, na aplicação de código corretor de erros, e outra, externa, na distribuição do resultado da etapa anterior entre diferentes provedores, que podem ser adicionados a qualquer momento indistintamente. Para se assegurar da integridade do arquivo, as tarefas de "desafio-resposta" são suportadas pela exploração de propriedades na representação polinomial de um $GF(2^n)$, que fornece a teoria necessária para a criação de um código autenticador de mensagem com propriedadades homomórficas.

O principal objetivo do *Hy-SAIL* é fornecer um modelo simples de distribuição de dados condizente com o tema computação em nuvem e que forneça um nível maior

de disponibilidade juntamente com a preservação da integridade. Essencialmente, o *Hy-SAIL* possui duas entidades envolvidas: usuário e provedor de armazenamento. A integridade é verificada através do mecanismo de *PoR*, que é iniciado pelo usuário. Para responder corretamente aos desafios submetidos pelo usuário, o *Hy-SAIL* exige como requisito básico que o provedor possua em suas unidades de armazenamento o conteúdo original do arquivo.

Na Figura 4.1 é mostrado uma arquitetura interna que contém a proposta do *Hy-SAIL*, com as entidades envolvidas no esquema geral de *PoR*, e suas respectivas atribuições.

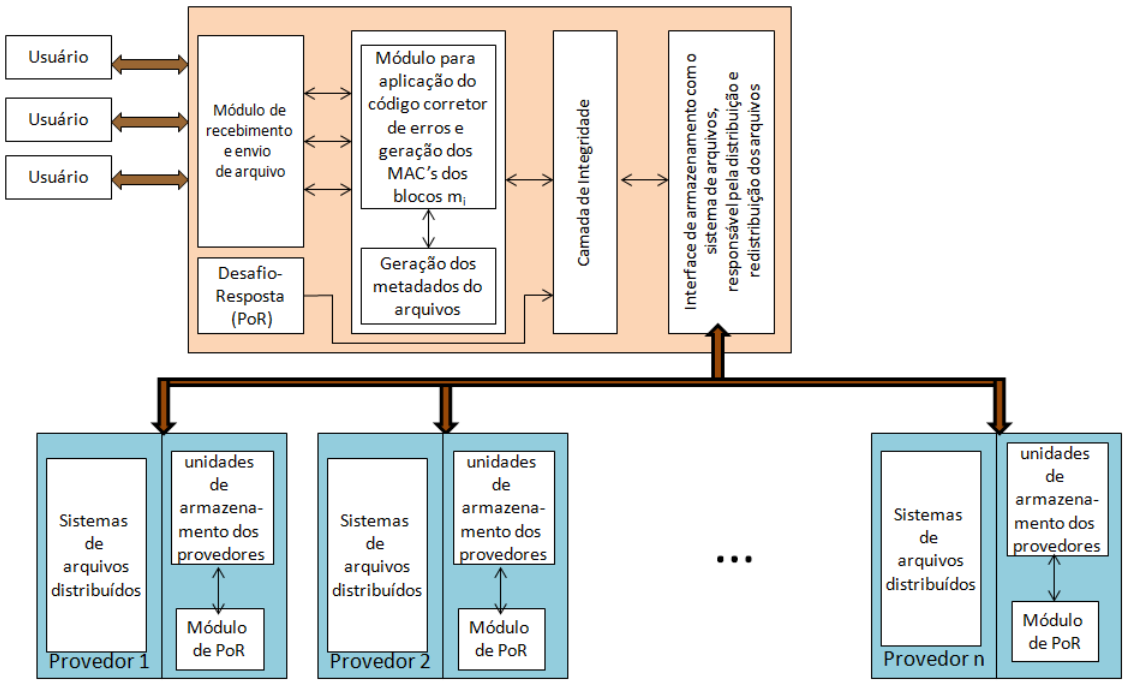

Figura 4.1: Arquitetura proposta pelo *Hy-SAIL*

- Usuário: Neste contexto, o usuário deve estar autenticado e deseja armazenar qualquer tipo de arquivo em um ambiente de armazenamento em nuvem. De acordo com o *SLA* (Nível de Acordo de Serviço), o dispositivo do usuário pode ou não fazer parte deste ambiente, e a qualquer momento ser questionado se permite compartilhar uma parte de seu disco rígido para cooperar com montante total de armazenamento. O usuário é responsável pela submissão do desafio ao provedor, para assegurar que os arquivos armazenados estão disponíveis e possuem uma quantidade mínima de perdas ou falhas, de modo a não prejudicar a recuperabilidade dos mesmos.

- Provedor de Armazenamento: Esta entidade fornece serviço de armazenamento e processamento de grande quantidade de dados em um ambiente de armazenamento distribuído. Tipicamente, as unidades de processamento e armazenamento realizam estas tarefas em conjunto e são atuantes em sistemas de arquivos distribuídos como o HDFS (*Hadoop Distributed File System*) (SHVACHKO et al., 2010). Estes recursos computacionais são solicitados a diferentes provedores, com o objetivo de assegurar um nível maior de disponibilidade, recuperabilidade e escabilidade.

Para atender aos objetivos de desempenho de largura de banda, armazenamento e processamento, o *Hy-SAIL* possui dois modelos para executar as tarefas de "desafio-resposta" na verificação da integridade dos arquivos: um modelo limitado de consultas, no qual o usuário armazena uma quantidade limitada de código autenticador de mensagem (MAC) de cada bloco de mensagem e realiza desafios aleatórios ao provedor; e outro modelo ilimitado de consultas, no qual o usuário armazena um único código autenticador de mensagem (MAC) de cada bloco de mensagem, o que o permite realizar uma quantidade ilimitada de verificações de integridade junto ao provedor.

Em uma abordagem de armazenamento de dados em nuvem, um determinado provedor pode trabalhar colaborativamente com outros provedores, os quais possuem servidores redundantes espalhados na Internet sem restrições quanto a localização geográfica, sistema operacional ou sistemas de arquivos. Funcionando como uma via de acesso para outros provedores, a interface de armazenamanto criada pelo *Hy-SAIL* (Figura 4.1) permite a interação entre diferentes sistemas de arquivos distribuídos. Esta característica adicional fornece uma maneira eficiente de realizar *downloads* de várias fontes simultaneamente e maximizar o uso dos provedores (MAYMOUNKOV; MAZIÈRES, 2003).

Além da possibilidade de redundância por parte dos provedores, o uso de código corretor de erros possibilita uma perspectiva a mais para garantir disponibilidade do conteúdo armazenado. Com os códigos *erasure*, é possível converter um arquivo de "n" blocos em uma mensagem codificada com "$(1 + \epsilon)n$" blocos, onde $\epsilon > 0$, de maneira que seja possível recuperar a mensagem original com uma quantidade de "n" blocos quaisquer. Ao receber os parâmetros descritos na seção 2.2.1, a fase de *Codificação* (seção 5.1) produz uma quantidade de blocos codificados $c_1, c_2, \ldots, c_j$, os quais são os resultados de operações XOR do conteúdo de uma quantidade aleatória de blocos m_i's, e então, são distribuídos nos provedores de armazenamento de maneira aleatória. Esta abordagem

permite obter um nível maior de disponibilidade ao *Hy-SAIL*, pois não há nenhuma distinção entre os provedores de armazenamento.

A integridade dos blocos codificados é verificada através de operações polinomiais sobre Corpos Finitos de característica igual a 2. A variedade de formas como cada elemento pode ser representado em um determinado Corpo Finito é explicado em (GUAJARDO et al., 2006). Em uma forma padrão, um elemento de um GF(2^n) pode ser representado como um polinômio $a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$ de grau $n-1$ com os coeficientes pertecentes a um GF(2). Esta representação possibilitou ao *Hy-SAIL* a criação de um *MAC* de cada bloco "m_i" e utilizá-los posteriormente na verificação de integridade dos blocos.

Para assegurar a integridade dos blocos, um código autenticador de mensagem com propriedade *XOR* homomórfica é necessária, pois o provedor armazena um bloco codificado (ex.: $c_i = m_1 \oplus m_2$) e o usuário possui os *MAC*'s dos blocos de mensagem com seus respectivos desafios. Isto significa, que para qualquer conjunto de mensagens (m_1, m_2), uma função *MAC* com esta propriedade de homomorfismo deve atender a Função 4.1:

$$f(m_1) \oplus f(m_2) = f(m_1 \oplus m_2). \tag{4.1}$$

Embora os métodos para assegurar a disponibilidade mostrem a sua eficácia, o paradigma de computação em nuvem possui um ambiente dinâmico, então, é de se considerar a possibilidade de qualquer dispositivo ser violado, desligado ou desconectado, o que pode interferir na disponibilidade de alguns blocos codificados. Para mitigar esta situação, o *Hy-SAIL* possui um mecanismo de *Redistribuição* (seção 5.5). Para otimizar esta tarefa, o *Online Codes* (MAYMOUNKOV, 2002), código corretor de erros utilizado pelo *Hy-SAIL*, possui a propriedade de processamento local, a qual permite a criação de qualquer bloco independentemente de outros blocos que compõe o arquivo (MAYMOUNKOV; MAZIÈRES, 2003). Na seção 5.5, este mecanismo é explicado em detalhes.

Nas seções a seguir, serão descritos os diferentes modelos de *PoR*'s implementados pelo *Hy-SAIL* e como a função *MAC* XOR homomórfica utilizada no *PoR* é construída pelo *Hy-SAIL*, bem como sua prova formal de segurança.

4.1 MODELO LIMITADO DE VERIFICAÇÃO DE INTEGRIDADE

Neste modelo, a fragmentação do arquivo F em "n" blocos, o *Hy-SAIL* armazena uma quantidade de informações sobre os blocos de mensagem para realizar verificações de integridade no arquivo F. O desafio deste modelo consiste em encontrar um equilíbrio no armazenamento de informações dos blocos de mensagem e na quantidade de verificações que serão necessárias durante o período que o arquivo estiver armazenado nos provedores remotos.

Primeiramente, cada bloco do arquivo é representado na forma polinomial sobre um Corpo Finito de característica 2, então serão criados blocos m_1, m_2, ..., m_n, com representações $m_1(x)$, $m_2(x)$, ..., $m_n(x)$, na forma de $\sum_{i=1}^{t-1} a_i x^i$, onde $a_i \in \{0, 1\}$.

Neste momento, o usuário seleciona uma chave $\mathcal{K}$, que em conjunto com uma função G (PRF) geram os desafios $P_j = G_{\mathcal{K}}(j)$, que são representados da mesma forma que os blocos de mensagem foram anteriormente.

Quanto ao MAC, o *Hy-SAIL* calcula o resto da divisão polinomial de $m_i(x)$ com $P_j(x)$ em um GF(2). Para fins de execução do PoR, o usuário armazena o código autenticador de mensagem de cada bloco $(\sigma_{i,j})$ e o desafio correspondente a este MAC, $G_{\mathcal{K}}(j)$.

Como os blocos armazenados nos provedores são o resultante de operações XOR dos conteúdos dos blocos, i.e. $c_e = m_a \oplus m_b$. E o usuário possui o resto de uma divisão polinomial entre o conteúdo dos m_i's blocos e um desafio P_j, torna-se necessário uma função MAC com propriedade XOR homomórfica para a verificação de integridade dos blocos armazenados. A condição de integridade será atendida se a equação 4.2 for verdadeira:

$$MAC(m_a) \oplus MAC(m_b) = MAC(m_a \oplus m_b) \tag{4.2}$$

As operações aritméticas no GF(2^n), explicadas na seção 2.3, fornecem a base teórica para a verificação de integridade nos modelos de PoR usados neste trabalho. A representação utilizada nos permite calcular o resto da divisão polinomial entre o arquivo m_i, representado por $m(x)$ e o desafio P_j, representado por $P_j(x)$, e garantir a propri-

edade desejada de homomorfismo entre os códigos autenticadores dos elementos de um mesmo Corpo Finito.

Para o modelo limitado, as operações aritméticas utilizadas serão de adição e divisão. As operações de adição são simples XOR entre os elementos, já a de divisão, exige implementações otimizadas para suportar valores cada vez maiores, em (LUO et al., 2012) tem-se um amplo estudo deste assunto.

Assumindo que o usuário possui os códigos autenticadores dos blocos m_a e m_b com um dos desafios já gerados anteriormente (P_j). Resultando em $\sigma_{a,j}$ e $\sigma_{b,j}$, através das equações 4.3 e 4.4 :

$$m_a(x)\ mod\ P_j(x) = \sigma_{a,j}(x) \tag{4.3}$$

$$m_b(x)\ mod\ P_j(x) = \sigma_{b,j}(x) \tag{4.4}$$

O provedor armazena em seus sistemas de arquivos o seguinte bloco codificado: $c_e = m_a \oplus m_b$. O usuário envia ao provedor um dos desafios de um conjunto pertencentes a P_j, e solicita o resto da divisão da equação 4.5:

$$[m_a(x) \oplus m_b(x)]\ \ mod\ P_j(x) = \sigma_{ab,j}(x) \tag{4.5}$$

No item 1 do Teorema 4 (na seção 2.3 deste documento), temos a seguinte afirmação $[a(x)+b(x)]\ \mod d(x) = [a(x)\ \mod d(x)]+[b(x)\ \mod d(x)]$ sobre elementos de Corpos Finitos na forma polinomial. Como no GF(2), a soma equivale a operação XOR *bit a bit*, podemos afirmar que a integridade será atendida se a equação 4.6 for verdadeira:

$$[m_a(x)\ mod\ P_j(x)] \oplus [m_b(x)\ mod\ P_j(x)] = [m_a(x) \oplus m_b(x)]\ \ mod\ P_j(x) \tag{4.6}$$

Ou seja, através da combinação das equações 4.3, 4.4, 4.5, 4.6, temos a equação 4.7:

$$\sigma_{a,j}(x) \oplus \sigma_{b,j}(x) = \sigma_{ab,j}(x) \tag{4.7}$$

Após esta rodada de verificação, o *Hy-SAIL* utiliza outro desafio para prosseguir no protocolo de *PoR*, e assim o faz até esgotar a quantidade de desafios $P_{(j)}$'s que possui. Visando apresentar uma proposta que viabilize um modelo mais flexível de verificação de integridade, na seção 4.2 segue uma proposta de modelo ilimitado de consultas à integridade de arquivos.

4.2 MODELO ILIMITADO DE VERIFICAÇÃO DE INTEGRIDADE

O modelo ilimitado segue a mesma representação polinomial de blocos de arquivos m_i's e desafio P_j utilizado no Modelo Limitado 4.1. A idéia deste modelo consiste em realizar uma quantidade ilimitada de verificações de integridade dos blocos codificados armazenados nos provedores. A vantagem deste modelo é o armazenamento de um único MAC de cada bloco m_i, em comparação com o modelo anterior, porém, exige que realize operações aritméticas de adição e divisão em um GF(2^n) de representação polinomial.

Neste modelo, o *Hy-SAIL* gera um único desafio aleatório P, o qual será a chave secreta, sendo $P(x)$ a sua representação de um polinômio primitivo com coeficientes em GF(2). Este desafio é armazenado com o usuário. Em seguida, é calculado o MAC de cada bloco do arquivo m_i com o desafio gerado. O desafio enviado aos provedores pelo usuário consiste em um conjunto de blocos codificados que poderá ser solicitado aleatoriamente a qualquer provedor de armazenamento, e a resposta esperada pelo usuário é conteúdo da operação XOR destes blocos. Explorando a propriedade XOR homomórfica, o usuário calcula o MAC deste resultado com o desafio $P(x)$ que possui como chave secreta.

Suponha que o usuário possui o MAC de todos os "n" blocos de mensagens com o desafio $P(x)$ e deseja verificar a integridade dos blocos codificados c_8, c_{25}, c_{38} composto dos seguintes blocos de mensagens: $c_8 = m_7 \oplus m_{34} \oplus m_{68} \oplus m_{79}$, $c_{25} = m_{34} \oplus m_{42} \oplus m_{45} \oplus m_{79}$ e $c_{38} = m_7 \oplus m_{42}$.

Então, é realizada uma operação XOR do conteúdos dos blocos codificados 8, 25 e 38, e este resultado é enviado de volta ao usuário.

Esta operação é representada na equação 4.8:

$$c_8 \oplus c_{25} \oplus c_{38} = [m_7 \oplus m_{34} \oplus m_{68} \oplus m_{79}] \oplus [m_{34} \oplus m_{42} \oplus m_{45}] \oplus [m_{79} \oplus m_7 \oplus m_{42}] \quad (4.8)$$

Realizando as operações de eliminação, o resultado é apresentado na equação 4.9:

$$c_8 \oplus c_{25} \oplus c_{38} = m_{68} \oplus m_{45} \quad (4.9)$$

Ao receber o valor resultante da equação 4.9, para dar continuidade a verificação de integridade dos blocos, o *Hy-SAIL* realiza a seguinte comparação:

$$[m_{68}(x) \mod P(x)] \oplus [m_{45}(x) \mod P(x)] = [m_{68}(x) \oplus m_{45}] \mod P(x) \quad (4.10)$$

Caso seja verdadeira a equação 4.10, inicia-se uma nova rodada de verificação com a escolha aleatória dos blocos codificados. No capítulo 5, é descrito cada fase que compõe o *Hy-SAIL*.

4.3 MODELO ADVERSARIAL DO Hy-SAIL

As ameaças de segurança enfrentadas por esta abordagem de armazenamento de arquivos em nuvem podem surgir de duas fontes distintas:

- O próprio provedor de armazenamento que tenha interesse em armazenar uma quantidade maior que a sua capacidade possui, seja apagando arquivos que são acessados raramente ou escondendo possíveis falhas de gerenciamento dos dados armazenados, o que permitiria obter ganhos econômicos maiores que a sua infraestrutura possa suportar;
- O adversário que possui a capacidade de comprometer uma quantidade de blocos codificados em um determinado período de tempo (Δt), e assim prejudiquar a recuperabilidade do arquivo F.

Neste trabalho, uma rodada equivale a um intervalo de tempo entre duas verificações consecutivas de integridade em um determinado provedor.

Será considerado um modelo adversarial geral, aquele em que o adversário é uma entidade maliciosa que realiza ataques em um modelo de quantidade limitada de blocos corrompidos. O *Hy-SAIL* considera um adversário $\mathcal{A}$ que corrompe no máximo uma fração "α" dos blocos codificados a cada rodada de verificação. Pela proposta do *Hy-SAIL*, como não há distinção entre os provedores de armazenamento, também não há restrição sobre a localização dos blocos corrompidos. O ponto relevante no modelo adversarial não foca nos provedores de armazenamento e sim na quantidade de blocos corrompidos.

Tanto o provedor quanto um adversário qualquer possuem a chance de causar algum dano aos dados armazenados. Neste contexto, a cada rodada de verificação existe a possibilidade de detectar possíveis falhas nos dados armazenandos. Uma rodada de verificação de integridade no *Hy-SAIL* é composta de 3(três) fases:

1. O adversário $\mathcal{A}$ consegue obter acesso a um dos provedores e modifica o conteúdo de uma quantidade de blocos que seja menor ou igual à $\delta(1 + k\delta) * n$ blocos codificados do arquivo F'.

2. O usuário submete o provedor à uma auditoria de verificação de integridade dos blocos codificados.

3. O usuário detecta a quantidade de blocos corrompidos e, se necessário, aciona um mecanismo de *Redistribuição* descrito na seção 5.5.

Para parametrizar o valor de "α", de acordo com os códigos corretores de erros aplicados pelo *Hy-SAIL* (seção 2.2), o arquivo F pode ser reconstruído após ter recebido uma quantidade de $(1 - \delta)(1 + k\delta)$n blocos quaisquer. A definição 9 apresenta o modelo adversarial:

Definição 9 *(Quantidade Limitada de Blocos Corrompidos) Seja* $\mathcal{C} = \{c_1, c_2, \ldots, c_\varphi\}$ *a representação dos blocos codificados gerados pelo* Hy-SAIL *e* $|\varphi|$*, o total de blocos codificados e armazenados nos diversos provedores. Iremos assumir que o adversário* $\mathcal{A}$ *pode corromper até* $\alpha|\varphi|$ *a cada rodada, onde* $0 \leq \alpha < 1$.

Então, a vantagem do *Hy-SAIL* em relação ao adversário $\mathcal{A}$ é apresentado na equação 4.11:

$$Vantagem_{\mathcal{A}}^{Hy\text{-}SAIL} = Pr\left[(1-\alpha)|\varphi| \geq (1-\delta)(1+k\delta)n\right] \tag{4.11}$$

Para complementar a equação 4.11, em (MAYMOUNKOV, 2002) é mostrado que esta probabilidade converge para $1-\delta^k$. Então a vantagem que o *Hy-SAIL* possui em relação ao adversário $\mathcal{A}$ é a mesma em termos de recuperabilidade e pode ser mostrada na equação 4.12:

$$Pr\left[(1-\alpha)|\varphi| \geq (1-\delta)(1+k\delta)n\right] = 1-\delta^k \tag{4.12}$$

Caso haja alguma falha ou perda no conteúdo dos blocos armazenados, o provedor pode acionar outros provedores e assim recuperar os blocos corrompidos. Mas caso tenha a intenção de enganar o usuário a cerca do conteúdo, o provedor e/ou adversário deverá responder aos desafios enviados pelo usuário. Como mostrados nas seções 4.1 e 4.2, o usuário enviará um desafio aleatório ao provedor, que deverá responder com o MAC_{hysail} dos blocos selecionados ou um conteúdo que seja possível verificar a integridade dos blocos.

Para um esclarecimento de como o *MAC* é relevante na construção do protocolo de *PoR*, uma construção genérica de um *MAC* é apresentado na definição 10:

Definição 10 *Seja F uma função PRF. Define-se um MAC de tamanho fixo para mensagens de tamanho "n" :*

- *Gen: recebe* $\{0,1\}^a$ *como entrada, escolhe* $k \leftarrow \{0,1\}^a$ *uniformemente aleatório.*
- *Mac: recebe uma chave* $k \in \{0,1\}^a$ *e uma mensagem* $m \in \{0,1\}^*$*, e produz o resultado da seguinte expressão:* $F_k(m)$*.*
- *Verf: recebe uma chave* $k \in \{0,1\}^a$ *e uma mensagem* $m \in \{0,1\}^*$*, e* $t \in \{0,1\}^a$*. Se* $t = F_k(m)$*, então a verificação do* $MAC_k(m)$ *está correta.*

Um adversário ($\mathcal{A}$), que pretende responder ao desafio submetido pelo usuário, possuirá uma função $\widetilde{\Pi} = (\tilde{Gen},\tilde{Mac},\tilde{Verf})$, a qual possui os mesmos elementos da Definição 10, exceto pela função pseudo-aleatória "f" que será usado ao invés de F. Então a probabilidade que um adversário $\mathcal{A}$ consiga forjar um MAC verdadeiro, de tamanho de "n" bits, de uma mensagem "m", segundo as propriedades descritas em (CARTER; WEGMAN, 1977), é descrito na equação 4.13 :

$$Pr\left[MAC^{falso}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] \leq 1/2^n \tag{4.13}$$

Para o caso do *Hy-SAIL* no modelo Limitado de consulta de integridade descrito na seção 4.1, o provedor pode tentar enganar o usuário ao ser consultado sobre o conteúdo de um bloco do arquivo F. Seja $E = (BlCod, Prov, \omega)$ um conjunto de informações enviado pelo usuário, onde $BlCod \in \left\{1, c_{|\varphi|}\right\}^{Prov}$, que representa um conjunto de blocos codificados em um determinado provedor $Prov$ e $\omega \in \{0,1\}^n$, o desafio enviado ao provedor para calcular o código autenticador de mensagem. Então este provedor responde ao usuário com um MAC de tamanho n com funções idênticas às usadas em $\widetilde{\Pi}$. Então para o provedor, a probabilidade de êxito na resposta do MAC é descrita na definição 11:

Definição 11 *(Probabilidade do Provedor em forjar um MAC válido) A probabilidade que um provedor tem em forjar um MAC que atenda, de maneira positiva, à consulta de verificação de integridade efetuada pelo esquema de PoR, de um determinado conjunto de bloco codificados, é definido pelo tamanho do MAC. Neste caso,* $n = |f_\omega(BlCod)^{Prov}|$

Segue a equação 4.14 que representa a definição 11:

$$Pr\left[MAC^{BlCod,\omega}_{Provedor} = MAC^{BlCod,\omega}_{Usuario}\right] \leq 1/2^n \tag{4.14}$$

Na seção 4.4 será apresentada a prova de segurança da função MAC construída pelo *Hy-SAIL*.

4.4 PROVA DE SEGURANÇA DA CONSTRUÇÃO DO CÓDIGO AUTENTICADOR DE MENSAGEM DO Hy-SAIL

Com base em definições já conhecidas de Álgebra Polinomial contidas em (MENEZES; OORSCHOT; VANSTONE, 2001), o objetivo desta seção é demonstrar a segurança da função MAC_{hysail} segundo o conceito apresentado em (CARTER; WEGMAN, 1977). Para atender as necessidades de homomorfismo exigidas na construção do MAC_{hysail}, foi proposto neste trabalho uma função de *MAC*, apresentada na equação 4.15:

$$MAC_{hysail}(m_i(x)) = m_i(x) \mod P(x) \tag{4.15}$$

onde $m_i(x)$ e $P(x)$ são representações polinomiais de elementos em um GF(2^n) e GF(2^t) respectivamente, onde $n > t$.

A função *MAC* construída no protocolo *Hy-SAIL* deve ser uma função de hash 2-universal, nos mesmos moldes que foi introduzida por *Carter and Wegman* em (CARTER; WEGMAN, 1977). Em uma visão geral, uma família consiste de um conjunto de funções de *hash* que mapeiam mensagens de tamanho arbitrário $\in \mathcal{U} = \{0,1\}^n$ em uma imagem $\mathcal{T} = \{0,1\}^t = \{1,\ldots,M\}$. Dado que uma função h de *hash* pertence à uma família de funções de *hash* 2-universal com distribuição uniforme, a probabilidade de colisão entre os *hashes* de duas mensagens diferentes é definida a seguir:

Definição 12 *Definição de uma Função de Hash 2-Universal (CARTER; WEGMAN, 1977): A família $\mathcal{H}$ de funções de hash $h : \mathcal{U} \to \mathcal{T}$ é 2-universal se, para $\forall\ x \neq y \in \mathcal{U}$, $Pr_{h\in\mathcal{H}}[h(x) = h(y)] \leq 1/2^t$, onde h é escolhida uniforme e aleatoriamente na família $\mathcal{H}$.*

Como ponto de partida é necessário saber qual o tamanho da família de funções $\mathcal{H}$, o que corresponde à quantidade de polinônios primitivos $P(x)$ de grau máximo "t" existem no $GF(2^t)[x]$. A definição de Corpo Finito constitui parte essencial para a obtenção desta resposta.

Definição 13 *Definição de Corpo Finito: O corpo finito é um corpo F o qual contém um número finito de elementos. A ordem de F é o número de elementos em F.(MENEZES; OORSCHOT; VANSTONE, 2001)*

A partir da definição 13, é possível afirmar que:

- Se F é um corpo finito, então F contém p^m elementos, para algum número primo "p" e um inteiro $m \geq 1$.
- Para toda potência de número primo p, existe um único corpo finito de ordem p^m. Este corpo é denotado por $\mathbb{F}_{p^m}$, ou por GF(p^m).

Seja F_p um corpo de ordem p. E $f(x) \in \mathbb{F}_p[x]$ um polinômio irredutível de grau "m". Então $\mathbb{F}_p[x]/f(x)$ é um corpo finito de ordem p^m. As operações polinomiais de adição e multiplicação são realizadas módulo $f(x)$.

Lema 1 *Um polinômio irredutível $f(x) \in \mathbb{F}_p[x]$ de grau "m" é chamado de polinônio primitivo, se "x" é um gerador de $\mathbb{F}^*_{p^m}$, o grupo multiplicativo de todos os elementos diferentes de zero em $\mathbb{F}^*_{p^m} = \mathbb{F}_p[x]/(f(x))$.*

O polinômio $f(x) \in \mathbb{F}_p$ de grau "m" é primitivo, se e somente se $f(x)$ divide $x^k - 1$ para $k = p^m - 1$, sendo "k" um número inteiro grande. Com isso, para cada $m \geq 1$, existem $\phi(p^m - 1)/m$ polinômios primitivos de grau "m" em $\mathbb{F}_p$.

Como os elementos ($P(x)$) são representações polinomiais de elementos de GF(2^t), então p = 2. No que tange à questão 1, podemos afirmar que existem $\phi(2^t - 1)/t$ polinômios com grau máximo "t". É sabido que $2^t - 1$ é conhecido como número de *Mersenne* (ALSHIBAMI; BOUSSAKTA; AZIZ,). Para um número de *Mersenne* (N_M) ser primo, "t" deve ser primo. Para efeitos de exemplo prático, temos t = 127 (=$2^7 - 1$), então $2^{127} - 1$ é primo.

No caso do *Hy-SAIL*, o grau máximo dos polinômios do conjunto $\mathcal{T}$ é igual à 127. Com $p = 2$ e $t = 127$, pode-se afirmar que $\phi(2^t - 1)/t$ ($(2^{127} - 2)/(2^7 - 1) \cong 2^{120}$), ou seja, existem aproximadamente 2^{120} polinômios disponíveis como possíveis resultados para o $MAC_{Hy-SAIL}$. Este número corresponde ao tamanho da família $\mathcal{H}$ de funções de hash h.

Com base no tamanho da família $\mathcal{H}$, é possível encontrar a probabilidade de colisão entre $hashes(MAC_{hysail})$ de duas mensagens diferentes.

PROVA: Para que $h(m_i)$ seja 2-universal, ela deve atender a condição estabelecida na Definição 12.

O *MAC* calculado pelo *Hy-SAIL* mapeia valores para um conjunto $\mathcal{T}$, através da família $\mathcal{H}$ de funções de hash h. Como já mostrado anteriormente, esta família possui uma quantidade de funções da ordem de 2^{120}.

Segundo a Definição 12 sobre *2-UHF*, sendo $\mathcal{H}$ uma família de funções de *hash*, $\forall$ $m_1 \neq m_2 \in \mathcal{U}$, $h \in \mathcal{H}$, temos $h : \mathcal{U} \rightarrow \mathcal{T}$, a propabilidade de que os *hashes* de m_1 e m_2 colidam é $\frac{2}{|\mathcal{T}|}$.

A função de *MAC* para cada"m_i" é definida na equação 4.16:

$$h((m_i(x)) = m_i(x)(\ mod\ P(x)) \tag{4.16}$$

Sendo $P(x)$ a representação polinomial dos elementos do conjunto $\mathcal{T}$. Pela definição 1, se $P(x)$ é um polinômio primitivo binário qualquer de grau t, então ele é irredutível, como é descrito no lema 2:

Lema 2 *Um polinômio irredutível $f(x) \in \mathbb{F}_p[x]$ de grau "m" é chamado de polinônio primitivo, se "x" é um gerador de $\mathbb{F}^*_{p^m}$, o grupo multiplicativo de todos os elementos diferentes de zero em $\mathbb{F}^*_{p^m} = \mathbb{F}_p[x]/(f(x))$.*

Pelo lema 1, $\mathbb{Z}_2[x]/P(x)$ sempre define um $GF(2^t)$:

Lema 3 *Seja $f(x) \in \mathbb{Z}_p[x]$ um polinômio irredutível de grau "m". Então $\mathbb{F}_p[x]/f(x)$ é um corpo finito de ordem p^m. As operações polinomiais de adição e multiplicação são realizadas no módulo de $f(x)$ (MENEZES; OORSCHOT; VANSTONE, 2001).*

No caso do *Hy-SAIL*, o resultado do resto da divisão polinomial possui imagem no conjunto de elementos em $\mathcal{T}$, sendo estes elementos com representações polinomiais em GF(2^t). Pode-se afirmar que, para $t = 127$, temos $|\mathcal{T}| = \mathrm{M}$ ($= 2^{127}$).

A seguir, são mostrados como o $MAC_{Hy-SAIL}$ pode alcançar o mesmo resultado apresentado na Definição 12.

Cada polinômio $m_i(x) \in \mathbb{Z}_2[x]$ é congruente a apenas um elemento de $GF(2^k)$, conforme as Propriedades de Congruências polinomiais mostradas na Definição 5 do capítulo 2 deste trabalho. Sendo $m_1 \neq m_2$, temos que definir a propabilidade de colisão dos *hashes* de m_1 e m_2, como mostrado na equação 4.17:

$$Pr_{h \in \mathcal{H}}\left[h(m_1) = h(m_2)\right] = ? \tag{4.17}$$

Supondo que para $\forall\, m_1 \neq m_2$ em $\mathcal{U}$ e pertencentes à um GF(2^n), seus valores de MAC's distribuídos de maneira uniforme e aleatória no conjunto $\mathcal{T}$. Então a probabilidade de colisão é apresentada na equação 4.18.

$$Pr_{h \in \mathcal{H}}\left[m_1(x) \mod P(x) = m_2(x) \mod P(x)\right] = \frac{\mid \{h \in H \mid h(m_1) = h(m_2)\} \mid}{\mid \mathcal{T} \mid}. \tag{4.18}$$

Ou seja, como o resto da divisão entre dois polinômios sobre um corpo é única, qualquer polinômio é congruente a um e apenas um elemento desse corpo. Logo, a congruência entre dois polinômios distintos quaisquer ocorrerá com probabilidade $|\mathbb{Z}_2[x]/P(x)|^{-1} = |GF(2^t)|^{-1} = 2^{-t}$. Logo, pode-se afirmar que a probabilidade de que dois elementos distintos possuam o mesmo valor de *hash* é o inverso da cardinalidade da imagem do resultado, ou seja, 1/M.

A família de funções de *hash* estabelecida pelo $MAC_{Hy-SAIL}$ distingue-se pela escolha uniformemente distribuida de $P(x) \in \mathcal{H}$, o conjunto de polinômios binários primitivos de ordem t. Esse conjunto possui $|\mathcal{H}| = \phi(2^t - 1)/k$ polinômios desse tipo. Portanto, a família $\mathcal{H}$ possui aproximadamente 2^{120} funções de *hash* distintas, quando a ordem de $P(x)$ é $t = 127$. Então, pode-se afirmar que a probabilidade dos *hashes* de elementos distintos colidirem é representado na equação 4.19:

$$Pr_{h \in \mathcal{H}}\left[m_1(x) \mod P(x) = m_2(x) \mod P(x)\right] \leq 1/M(1/2^{127}) = 2^{-127}. \tag{4.19}$$

Então, o MAC utilizado pelo *Hy-SAIL* de fato pertence à uma família de funções *hash* 2-universal. ■.

4.5 SEGURANÇA DO *Hy-SAIL*

De acordo com as definições 9 e 11, é possível afirmar o seguinte teorema sobre o protocolo *Hy-SAIL*:

Teorema 9 Hy-SAIL *é seguro em relação à integridade e disponibilidade dos dados armazenados remotamente.*

Como mostrado na equação 4.19, a probabilidade de forjar o MAC_{hysail} é igual à $1/2^t$. E também, como mostrado na equação 4.12, a probabilidade de falha na recuperabilidade é aproximadamente igual à δ^k. Assumindo que os modos de falhas são independentes, a probabilidade total de falha pode ser expressa pela seguinte expressão: $Pr[Falha_{hysail}]$ = Pr[$Falha_{integridade}$] + Pr[$Falha_{disponiblidade}$]. Então, a probabilidade total de sucesso é dada pela equação 4.20:

$$1 - Pr\left[Fail\right] = 1 - (1/2^t + \delta^k) \tag{4.20}$$

o que assintoticamente é muito próximo de 1, visto que $1/2^t$ e δ^k possuem valores desprezíveis. ■

Neste capítulo foi apresentado uma visão geral do *Hy-SAIL* com sua arquitetura interna, seus modelos de consulta de integridade, um modelo adversarial condizente com o ambiente de computação em nuvem e uma prova formal da segurança tanto da função MAC implementada para a consulta de integridade como do modelo de implementação do *Hy-SAIL*. No próximo capítulo, será mostrado os detalhes de cada fase que compõe o *Hy-SAIL* e os resultados alcançados na implementação.

5 Hy-SAIL EM DETALHES

Neste capítulo são apresentados os detalhes do protocolo *Hy-SAIL*. Para garantir a disponibilidade dos arquivos armazenados remotamente, os arquivos são expandidos e codificados através do código corretor de erros detalhado na seção 2.2. Os blocos codificados são enviados para um conjunto de provedores de armazenamento em nuvem. Para verificar a integridade dos blocos, os provedores são consultados sobre o código autenticador de mensagem de cada bloco com desafios periódicos. E caso haja alguma distorção nos blocos verificados, o *Hy-SAIL* é projetado para realocar o conteúdo destes blocos sem a necessidade de efetuar o download do arquivo F.

Todo este processo é detalhado nas cinco fases que compõem o *Hy-SAIL*: Codificação, Decodificação, Distribuição, Auditoria e Redistribuição.

A Figura 5.1 apresenta a estrutura do Hy-SAIL para executar as tarefas:

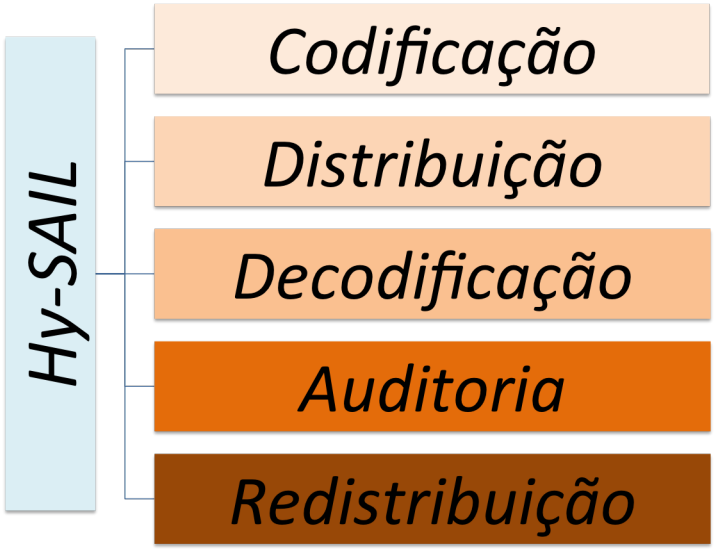

Figura 5.1: Divisão das fases que compõe o *Hy-SAIL*

Com o intuito de aferir esta teoria o *Hy-SAIL* foi implementado em *Python*, versão 2.7.3, sem que tenham havido requisitos de otimização de desempenho no desenvolvimento. Os experimentos foram realizados em uma estação *MacBook Pro* com as seguintes configurações: Sistema Operacional MAC OSX 10.6.8; processador Intel Core i7 quad

core de 2,3GHz com 6MB de cache L3; 8GB de memória DDR3; Disco rígido de 500GB com 5400 RPM.

5.1 CODIFICAÇÃO

A fase de codificação do *Hy-SAIL* possui duas etapas distintos: o primeiro é responsável pela geração dos blocos codificados e o segundo, pelo cálculo dos códigos autenticadores de mensagem dos "n" blocos de mensagem, os quais são representados por MAC_{hysail}.

Com os parâmetros do *Online Codes* 2.2.1 (k, ϵ, δ, n), o código corretor de erros é acionado e o arquivo F é dividido em n blocos de tamanhos iguais. Após isto, são criados os $(k\delta)n$ blocos auxiliares, os quais são adicionados aos n blocos. Um bloco auxiliar é o resultado da operação XOR do conteúdo dos blocos de mensagens que possuem um grau "k". Ou seja, nesta fase de pré-codificação cada bloco de mensagem escolhido aleatoriamente são replicados "k" vezes em relação a cada bloco auxiliar. Neste momento, o arquivo F é transformado em uma mensagem composta(F'), a qual possui $(1 + k\delta)n$ blocos.

A criação dos blocos codificados é realizada através de operações XOR *bit a bit* entre "d" blocos da mensagem composta ($(1 + k\delta)n$) de acordo com a distribuição de probabilidade ρ_d como descrito na equação 2.1, tal que $\sum_{d=1}^{D} \rho_d = 1$. O grau "$d$" corresponde à quantidade total de blocos (m_i) que compõem os blocos codificados, onde "d' pode variar de 1 a D conforme equação 2.2.

Para comprovar estas afirmações acerca da distribuição de probabilidade, foram realizados experimentos com diferentes parâmetros para a geração dos blocos codificados e realização do *PoR*. Os blocos codificados foram gerados de acordo com a distribuição de probabilidade mostrada em (MAYMOUNKOV, 2002). A Figura 5.2 mostra a probabilidade que cada grau possui no conjunto dos blocos codificados gerados, foram feitos experimentos com 3 funções:

- Função1 : $\epsilon = 0,01$ e $\delta = 0,005$;
- Função2 : $\epsilon = 0,02$ e $\delta = 0,01$;
- Função3 : $\epsilon = 0,1$ e $\delta = 0,2$.

Embora sejam funções tenham parâmetros diferentes, os experimentos mostram distribuições de probabilidade semelhantes na construção dos blocos codificados.

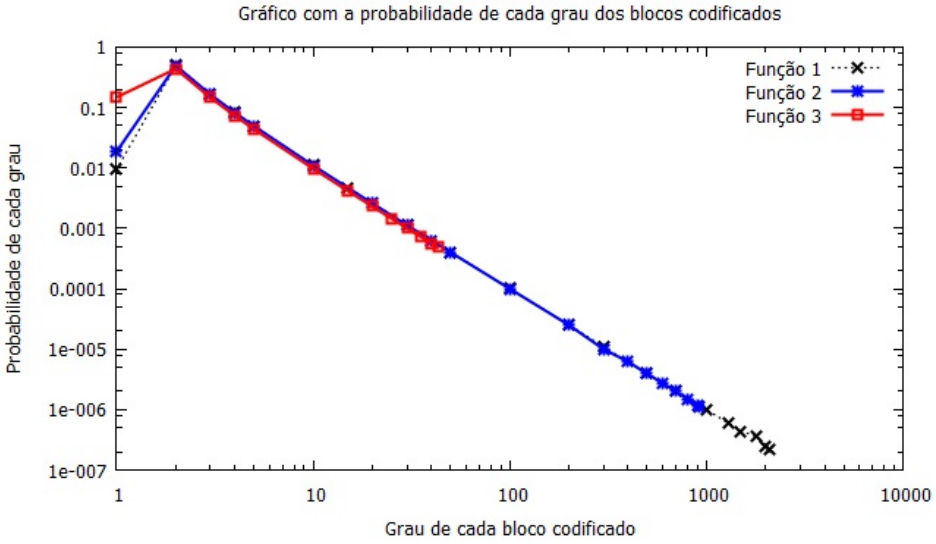

Figura 5.2: Probabilidade X Grau do bloco codificado

Após gerar todos os blocos codificados, o usuário armazena uma estrutura de dados com os índices dos blocos que compõem cada bloco codificado. Um dos itens nesta estrutura possui a seguinte forma:

$$\langle j, m_i \rangle$$

onde:

$$\left\{ \begin{array}{l} j: \text{ índice de cada bloco codificado} \\ m_i: \text{ identificação dos blocos de mensagens que geram o bloco codificado} \end{array} \right\}$$

Como exemplo da fase de Codificação, se o 8° bloco codificado é formado por $c_8 = m_1 \oplus m_5 \oplus m_{11} \oplus m_{128}$, a estrutura de dados apresenta a seguinte forma: $\langle 8, \{1, 5, 11, 128\} \rangle$. Na figura 5.3, a estrutura de dados é mostrada com os blocos codificados e seus respectivos blocos de mensagens que o compõe.

Para um melhor entendimento, a Figura 2.2 sintetiza esta etapa, apresentando desde a criação dos blocos de mensagens, blocos auxiliares e blocos codificados até a recuperação do arquivo original, mesmo havendo perdas no canal de transmissão dos blocos.

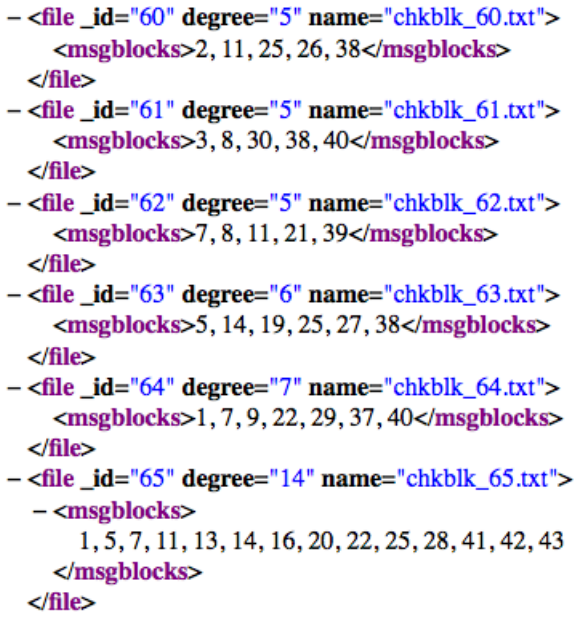

Figura 5.3: Estrutura de dados dos blocos codificados

É importante frisar que o valor de "ϵ" é escolhido levando em consideração o quão redundante F deve ser. Note que o valor de "ϵ" é um parâmetro subjetivo a ser definido pelo usuário e esta variável é influenciada por outros parâmetros como:

- Frequência na qual os blocos são corrompidos;
- Espaço disponível em disco rígido;
- O tempo exigido para todo o processo de codificação;
- Custos para manutenção de armazenamento.

Após a criação dos blocos codificados, cada bloco de mensagem que compõe o arquivo F, $\{m_1||m_2||\ldots||m_n\}$ é representado por um polinômio de grau igual ou menor que $p-1$ sobre um GF(2^p), $m_n(x) = \sum_{i=1}^{p-1} a_i x^i$, onde os coeficientes a_i são elementos de um GF(2), a saber $\{0, 1\}$. Dependendo do modelo a ser usado no *PoR*, se limitado ou ilimitado, existe uma pequena diferença na próxima etapa.

Se o usuário optar pelo modelo limitado de consultas de integridade ao provedor de armazenamento, um quantidade aleatória de l números primos são gerados e representados de modo polinomial da mesma maneira que os blocos m_i's foram representados

anteriormente, porém, com grau menor que p. Para finalizar esta etapa, são calculados os *MAC's* para cada bloco m_i, usando a seguinte equação:

$$MAC_{hysail} = m_i(x) \bmod P_j(x) \tag{5.1}$$

onde i pode variar de 1 a n, e j de 1 a l. Se for usado o método ilimitado, o valor de l é igual a 1. Tanto o pseudo-código A.1 localizado no apêndice deste trabalho como a Figura 5.4 sintetizam este detalhamento de criação do MAC_{hysail}.

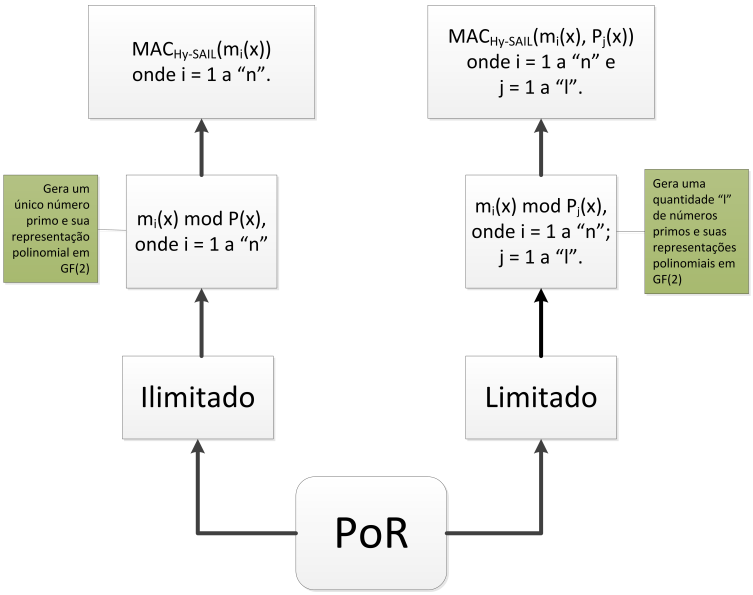

Figura 5.4: Criação do MAC para o protocolo Hy-SAIL mediante o modelo a ser utilizado no PoR

A escolha no valor de n reflete no tamanho do bloco de mensagem, consenquentemente na geração do código autenticador dos blocos de mensagens m_i. Para o cálculo do MAC_{hysail} de cada bloco de mensagem, o *Hy-SAIL* realiza uma operação de divisão polinomial com elementos de GF(2^p), o qual utiliza uma biblioteca *"ffield.py"* desenvolvida em Python disponível em (MARTINIAN, 2002). Esta operação é realizada em tempo $\mathcal{O}(p^2)$ como mostrado na Figura 5.5, onde p é o tamanho em *bits* de m_i.

A criação da estrutura de dados é finalizada junto com esta fase, a qual será usada nas fases de *Decodificação* e *Auditoria* com todos os mapeamentos dos blocos que compõem cada bloco codificado e o resultado do MAC_{hysail} de cada bloco. Como exemplo da implementação realizada para este trabalho, o arquivo "*metadata.xml*" é criado com todos os metadados gerados na fase de Codificação.

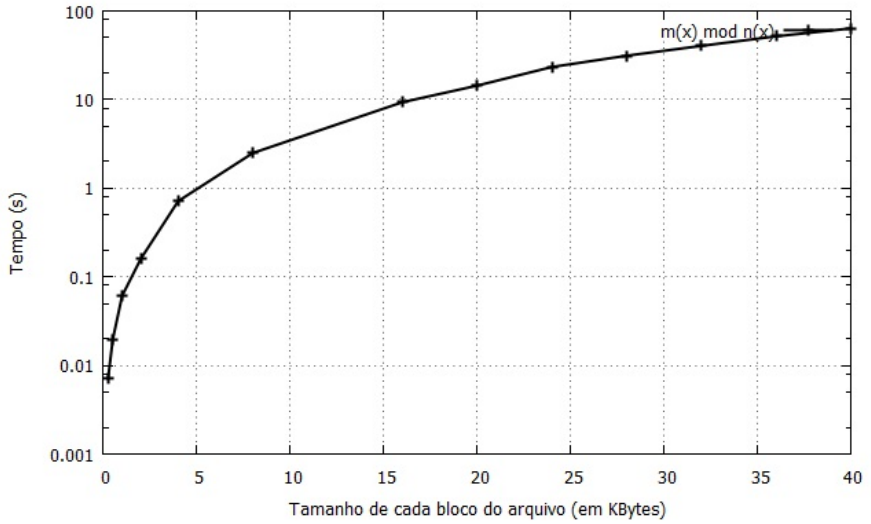

Figura 5.5: Relação entre o tamanho do bloco m_i e o tempo gasto para realização da operação polinomial

Na Figura 5.6, são mostrados os parâmetros (k, ϵ, δ, n) utilizados pelo código corretor de erros, tamanho do arquivo a ser codificado e os MAC_{hysail} dos blocos de mensagens com o desafio gerado aleatoriamente.

```
– <encode>
    <options E="0.2" K="3" S="0.1" compMsg="55"/>
  – <message>
      <file name="filein.txt" size="52428800" type="input"/>
    </message>
  – <messageblocks padding="1219256" segment="500" size="1219274" total="44" type="output">
      <key>FDE04807074BA1C9</key>
    – <file _id="0" name="fileout_0.txt">
        <mod>5BF65A92D335EBC1</mod>
      </file>
    – <file _id="1" name="fileout_1.txt">
        <mod>1BB886EE53D5DF9C</mod>
      </file>
    – <file _id="2" name="fileout_2.txt">
        <mod>DD5B1C559FD5F27</mod>
      </file>
    – <file _id="3" name="fileout_3.txt">
        <mod>70F355102D6718F3</mod>
      </file>
```

Figura 5.6: Parâmetros utilizados na geração dos blocos codificados e MAC's de cada bloco codificado

O tamanho do arquivo F e os parâmetros escolhidos na fase de Codificação (δ,ϵ,k,n) refletem diretamente no tempo de processamento para geração dos blocos codificados. O ponto a ser perseguido nos resultados consiste em achar um valor adequado de "n", para cada tamanho de arquivo F, de forma que cada fase tenha um tempo otimizado para suas tarefas.

Com valores de $\delta = 0,05$ e $\epsilon = 0,10$, a Figura 5.7 mostram o tempo gasto na codificação de arquivos de 50 MB, 100 MB, 500 MB e suas relações diretas com os diferentes valores de n. Na Figura 5.7, os arquivos de 50 MB e 100 MB alcançam valores constantes a partir de 300 blocos de mensagens, porém com o arquivo de 500 MB o mesmo comportamento é alcançado com 800 blocos de mensagens. Em (MAYMOUNKOV, 2002) é mostrado que o tempo de codificação alcança tempo de execução constante, pois a medida que o número de blocos de mensagens aumenta o tamanho de cada bloco de mensagem diminui, levando a um tempo constante o cálculo dos blocos codificados.

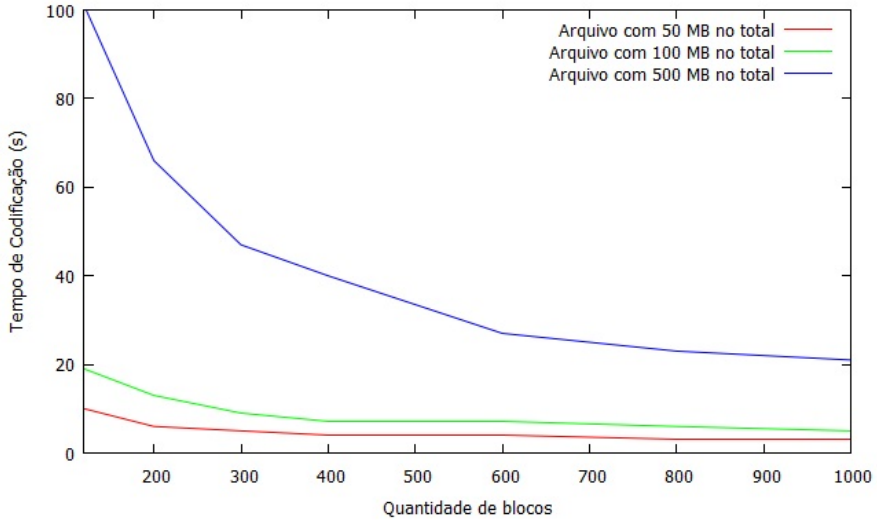

Figura 5.7: Tempo de Codificação em função do tamanho do bloco

5.2 DISTRIBUIÇÃO

O *Hy-SAIL* não possui um comportamento estático como ocorre nos *clusters* ou *grids*, onde os servidores envolvidos possuem um sistema operacional específico para executar alguma tarefa ou algum tipo de configuração que fora realizada anteriormente em cada servidor. Tanto para o envio como para o recebimento de arquivos, a única necessidade reside em estabelecer um canal autenticado entre usuário e o provedor de armazenamento.

Ao contrário do que é abordado no *HAIL* (BOWERS; JUELS; OPREA, 2009a), o *Hy-SAIL* não faz distinção entre servidores primários e secundários, e exige apenas os requisitos básicos para armazenamento de dados em um ambiente de rede, tais como conectividade e espaço em disco rígido. Por conta desta característica, o *Hy-SAIL*

pode enviar indistintamente quaisquer blocos codificados para quaisquer provedores de armazenamento.

Aproveitando esta característica, com o *Hy-SAIL* é possível implementar uma política para alocação de arquivos em redes *P2P*, onde a volatilidade dos recursos computacionais não afeta a disponibilidade dos arquivos, permitindo a recuperabilidade a qualquer instante.

Após a fase de Codificação, todos os blocos codificados são enviados aleatoriamente para os provedores de armazenamento $\mathcal{S}_i$, assegurando que os blocos codificados com grau 1 sejam os primeiros a serem recuperados para o início da fase de *Decodificação*. A intenção é manter os blocos codificados em provedores que forneçam um nível de recuperabilidade com tempo de resposta que seja viável do ponto de vista prático, funcional e econômico.

A seguir segue uma descrição dos diferentes tipos de distribuição dos blocos codificados, que facilitará a recuperação do arquivo original F armazenado nos provedores. A distinção entre os tipos considera como os blocos codificados estão mapeados, incluindo quais blocos os compõem e onde estão armazenados:

- Stateful: Nesta abordagem, o usuário sabe em qual servidor os blocos codificados estão armazenados, podendo realizar consultas acerca da integridade e extrair informações do conteúdo dos mesmos. É uma abordagem típica na qual um provedor de armazenamento é responsável por todas as fases da operação do *Hy-SAIL*.

- Stateless: O usuário, com alta probabilidade, não sabe onde um bloco codificado específico está localizado. A interação com os blocos codificados é possível através apenas de um *gateway*, que possui uma estrutura de dados identificando os blocos codificados. Este é o caso onde o provedor de armazenamento recebe todos os dados e os encaminha para seus parceiros, com o intuito de suportar demandas elásticas em sua infraestrutura.

- Peer-to-peer: Este cenário é semelhante aos caso do *Stateless*, onde o usuário se conecta a um *gateway* que controla as conexões entre diferentes clientes, informando a disponibilidade e conectando diretamente. É uma disposição onde cada nó possui funções de servidor e cliente ao mesmo tempo. Apesar de possuir recur-

sos diferenciados, todos os nós possuem as mesmas capacidades de encaminhar informações e responsabilidades na localização de dados armazenados.

Os blocos codificados são distribuídos de maneira aleatória juntos aos provedores de armazenamento $\mathcal{S}_i$'s. Após o envio, o *Hy-SAIL* armazena uma tabela de índices com informações dos blocos codificados que foram distribuídos nos diversos provedores de armazenamento, como no exemplo a seguir:

$$\langle \mathcal{S}_i, c_t \rangle$$

onde:

$$\left\{ \begin{array}{l} \mathcal{S}_i : \text{ Identificador do provedor.} \\ c_t : \text{ identificação dos blocos codificados distribuídos nos provedores } \mathcal{S}_i \end{array} \right\}$$

Na figura 5.8 é ilustrado o processo de distribuição dos blocos.

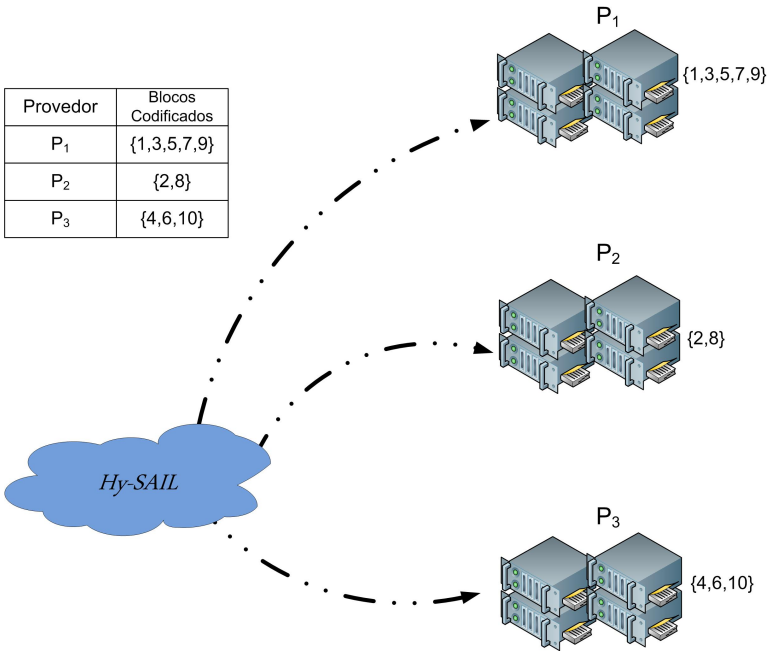

Figura 5.8: Distribuição dos blocos codificados nos provedores

5.3 AUDITORIA

Existem dois modelos utilizados no *Hy-SAIL* para validar a integridade dos blocos codificados: Limitado e Ilimitado. O diferencial entre estes modelos consiste na quan-

tidade de informações armazenadas pelo usuário e o tráfego gerado para a verificação de integridade.

O desafio submetido pelo usuário depende do modelo usado no protocolo de prova de recuperabilidade. Caso o usuário opte pelo Modelo Limitado (seção 4.1) de consultas, o usuário submete os seguintes componentes ao provedor de armazenamento:

- um polinômio primitivo aleatório ($P(x)$), de um conjunto de l polinômios já gerado anteriormente;
- os índices dos blocos codificados. Como resultado, o usuário recebe o resultado da equação apresentada no pseudo-código A.1.

No pseudo-código B.1 localizado no apêndice deste trabalho e na Figura 5.9 mostram o funcionamento do Modelo Limitado de verificações de integridade dos blocos codificados que estão armazenados no provedor.

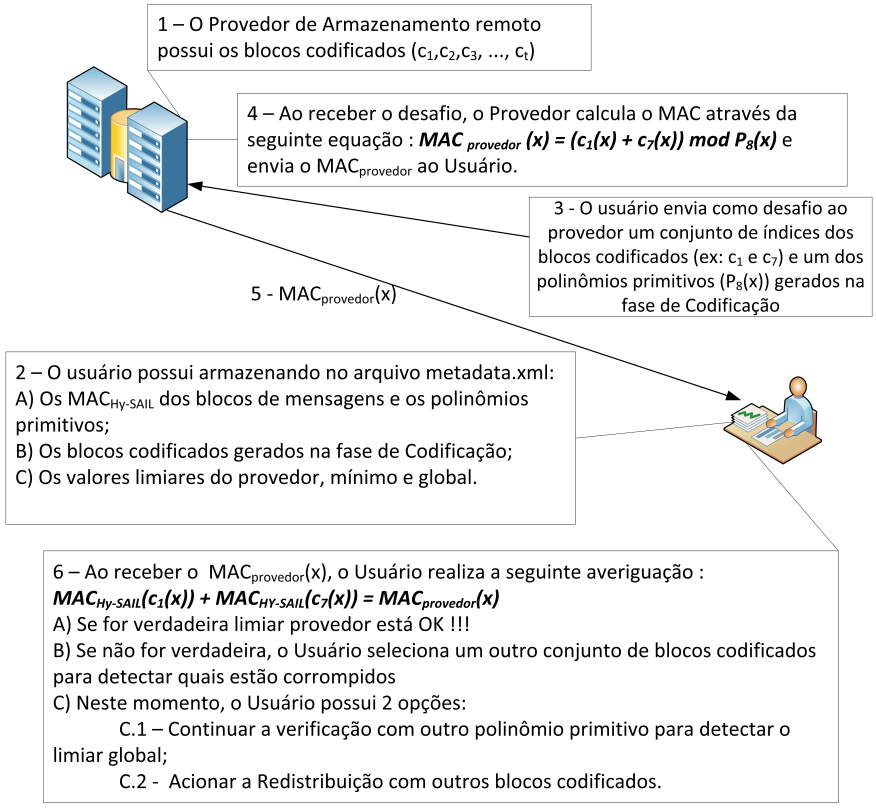

Figura 5.9: Modelo Limitado de verificação de integridade do Hy-SAIL

No pseudo-código C.1 localizado no apêndice deste trabalho e figura 5.10 são mostrados o funcionamento do Modelo Ilimitado, na qual o usuário envia ao provedor de arma-

zenamento apenas os índices dos blocos codificados e como resultado, o conteúdo do XOR destes blocos.

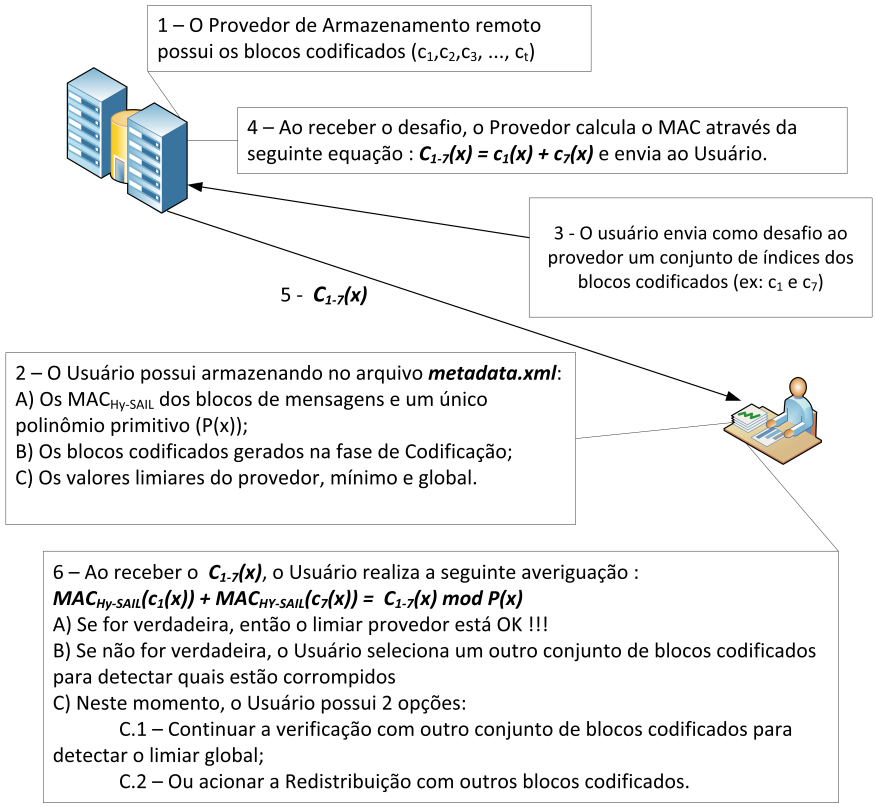

Figura 5.10: Modelo Ilimitado de verificação de integridade do Hy-SAIL

Nesta fase de Auditoria, seja qual for o modelo a ser utilizado, o usuário deve ter garantias da integridade de $(1 - \delta)$ da mensagem composta, então faz-se necessário quantificar os valores dos limiares de cada provedor de armazenamanto e o limiar global do arquivo para garantir a recuperabilidade do arquivo F com alta probabilidade. Para esta fase de Auditoria, tem-se os limiares:

- $\tau_{provedor}$: é a quantidade de blocos codificados que estão corrompidos de cada provedor;
- τ_{minimo}: é a quantidade mínima de blocos codificados que cada provedor deve ter para não prejudicar a recuperabilidade do arquivo.
- τ_{global} : é o somatário dos τ_{minimo}'s de todos os provedores e que garante a recuperabilidade do arquivo original com alta probabilidade.

Para questões de entendimento desta fase, os modelos de Auditoria desenvolvidos no *Hy-SAIL* foram detalhados nas seções 4.1 e 4.2.

Com bases nos limiares mostrados, na fase de Auditoria é possível obter informações quanto a recuperabildade do arquivo original. O *Hy-SAIL* permite que seja usada uma quantidade aleatória de blocos codificados para reconstruir o arquivo original, desde que essa quantidade seja maior que um valor mínimo definido. De acordo com as equações apresentadas na seção 2.2, é mostrado na Figura 5.11 a relação existente entre a probabilidade de falha na recuperação do arquivo original dada uma quantidade mínima de blocos codificados que é exigida para esta tarefa, confirmando que aumentando o valor de δ, que representa o percentual de perda no canal, é necessária uma quantidade maior de blocos codificados para garantir a recuperabilidade do arquivo original.

Para efeito desta ilustração, foram usados como parâmetros valores fixos de $k = 3$ e $n = 500$ para obter as probabilidades de falha na recuperação de acordo com a quantidade de blocos codificados. A probabilidade de falha na recuperação é representada pela fórmula δ^k, e a quantidade de blocos codificados por $(1 - \delta)(1 + k\delta)n$.

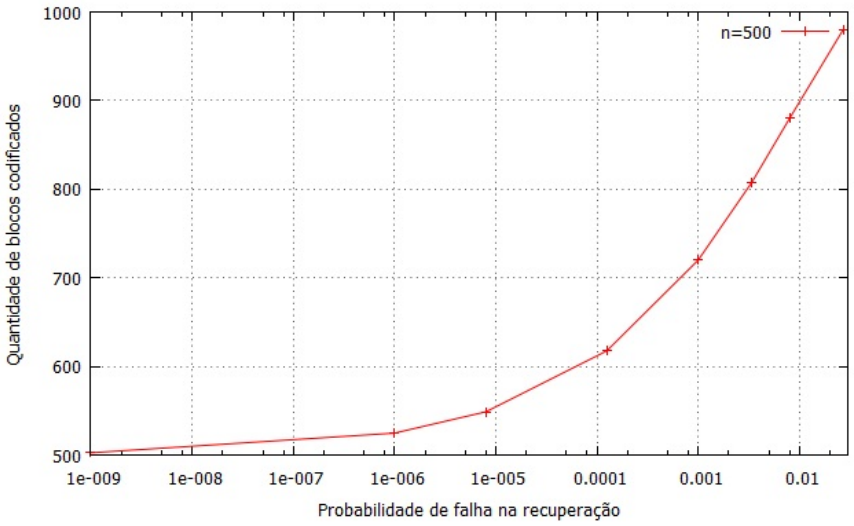

Figura 5.11: Quantidade de blocos X Probabilidade de falha na recuperação

5.4 DECODIFICAÇÃO

O principal objetivo desta fase é reconstruir a mensagem original F com, no mínimo, $(1 - \delta)(1 + k\delta)n$ blocos codificados quaisquer. Dependendo do modelo usado para executar o *PoR*, se limitado ou ilimitado, o usuário envia desafios aos provedores de armazenamento para a realização de verificação de integridade de um conjunto de blocos codificados, como descrito na seção 5.3, na fase de *Auditoria*. Se a integridade

dos blocos codificados for confirmada, o usuário solicita a cópia de, no mínimo, $(1 - \delta)(1 + k\delta)n$ blocos codificados. Após todos os blocos codificados serem copiados, o usuário inicia o processo de decodificação do arquivo F.

Primeiramente, deve-se procurar pelos blocos codificados com grau 1, ou seja, estes blocos codificados são cópia simples de um bloco da mensagem. Estes blocos de mensagem são marcados como decodificados. Em seguida, serão analisados os blocos codificados que possuem todos os blocos de mensagens já decodificados, com a exceção de um deles.

A decodificação é efetuada da seguinte maneira: suponha que $c_{12} = m_1 \oplus m_5 \oplus m_9 \oplus m_{13}$ e que m_{13}, m_5 e m_9 já são decodificados. Então o bloco de mensagem m_1 será recuperado através da seguinte equação : $m_1 = c_{12} \oplus m_5 \oplus m_9 \oplus m_{13}$.

A cada bloco de mensagem que é recuperado, o mesmo é marcado como decodificado. Este processo é realizado repetidamente até que os blocos que compõem a mensagem F sejam decodificados e concatenados, formando o arquivo original $F=\{m_1||m_2||\ldots||m_n\}$.

Nos experimentos realizados com o *Hy-SAIL*, pode-se afirmar que a complexidade de tempo de execução é linear, sendo diretamente proporcional à quantidade de blocos de mensagens. Mesmo havendo variações no tamanho dos arquivos, a Figura 5.12 comprova esta afirmação.

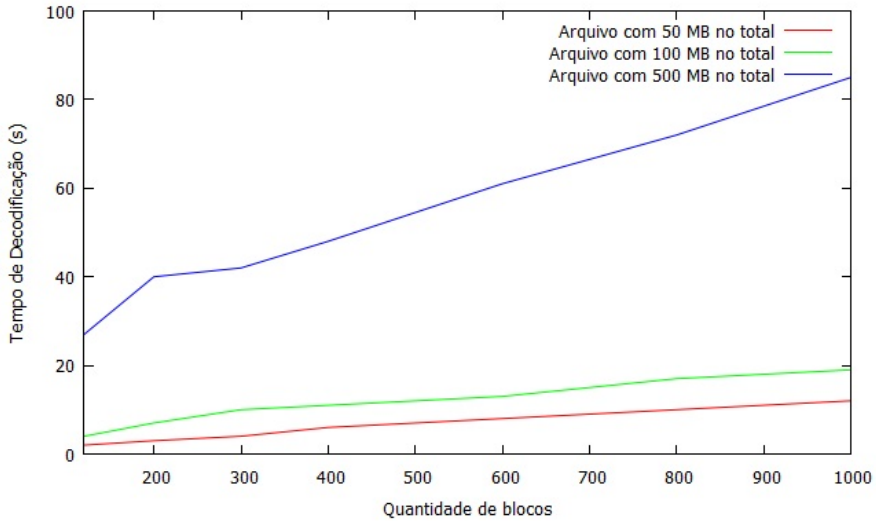

Figura 5.12: Tempo de Decodificação em função do tamanho do bloco

5.5 REDISTRIBUIÇÃO

Como já mostrado na seção 5.3, o *Hy-SAIL* mantém três limiares que são checados a cada rodada na verificação de integridade dos blocos codificados, o que constitui um diferencial em relação aos PoR's apresentados no capítulo 3.

Nesta fase, após uma rodada de verificação e ter calculado os limites apresentados, o usuário pode requerer ao provedor uma nova composição dos blocos que compõem os blocos corrompidos. Na abordagem adotada pelo *Hy-SAIL* não se faz necessário efetuar o *download* do arquivo inteiro para reconstruir os blocos codificados corrompidos. O objetivo desta fase é manter os valores de distribuição de probabilidade dos blocos codificados. De maneira simples, são mapeados os outros blocos codificados que também possuem blocos de mensagens em sua composição e são executadas operações XOR *bit a bit* entre eles, que gera um novo bloco codificado.

A Figura 5.13 ilustra esta ação de reconstruir e redistribuir novos blocos codificados:

- Passo 1: Se algum problema ocorrer com o dispostivo que armazena algum bloco codificado, o mecanismo de redistribuição é acionado e uma nova composição do bloco codificado é construída.
- Passo 2: Após receber alguma informação sobre falha na integridade de um determinado bloco codificado, é realizada uma operação XOR *bit a bit* com o conteúdo de dois ou mais blocos codificados de quaisquer dispositivo para recompor o bloco corrompido (Ex.: $c_{j+1} = c_{j-6} \oplus c_{j-1}$).
- **Passo 3**: O novo bloco codificado(c_{j+1}) é enviado para outro dipositivo.

5.6 COMPARAÇÃO DE FUNCIONALIDADES DO Hy-SAIL

Nesta seção é apresentada uma comparação das características do *Hy-SAIL* em relação aos modelos de *PoR* mostrados na seção 3: Modelo Juels-Kalilski (JUELS; KALISKI, 2007), Modelo Shacham-Waters (SHACHAM; WATERS, 2008) e HAIL (BOWERS; JUELS; OPREA, 2009a). O objetivo é mostrar a eficiência do Hy-SAIL nos requisitos

Itens	Juels-Kaliski	Shacham-Waters	HAIL	Hy-SAIL
Complexidade de Codificação do arquivo a ser armazenado no provedor remoto	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}$(n)
Complexidade de Decodificação do arquivo armazenado no provedor remoto	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(1)$
Armazenamento do Provedor	$\mathcal{O}$(2n)	$\mathcal{O}$(2n)	$\mathcal{O}$(n)	$\mathcal{O}$(n)
Armazenamento em Múltiplos Provedores	NÃO	NÃO	SIM	SIM
Acesso aos blocos dos arquivos na realização do PoR	PARCIAL	PARCIAL	TOTAL	TOTAL
Redistribuição dos blocos corrompidos	NÃO	NÃO	NÃO	SIM
Quantidade ilimitada de vezes para consulta de integridade dos blocos	NÃO	SIM	NÃO	SIM

Tabela 5.1: Comparação dos Modelos de PoR. “n”é o tamanho da mensagem e “k” é o parâmetro de segurança.

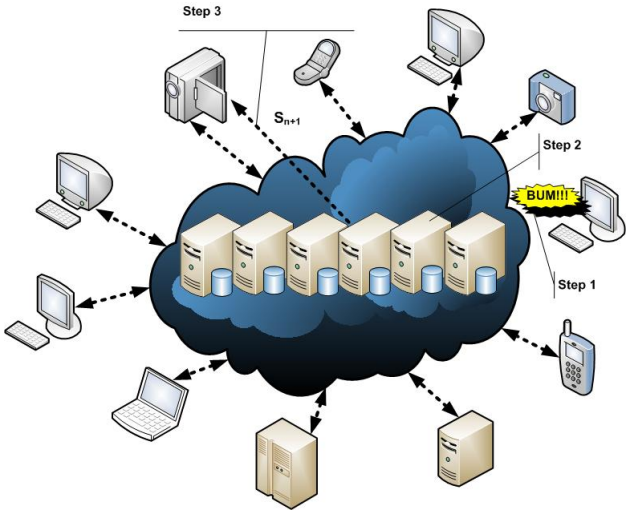

Figura 5.13: Redistribuição de um novo bloco

computacionais para o processamento, armazenamento e consumo de largura de banda na realização do "desafio-reposta" do PoR.

6 CONCLUSÕES

Neste trabalho, foi apresentado o *Hy-SAIL*, que foca em duas importantes preocupações para armazenamento de dados em provedores remotos: disponibilidade e integridade. O objetivo deste trabalho é fornecer suporte a esta demanda desenvolvendo um protocolo de prova de recuperabilidade seguro, utilizando sólidos conceitos de álgebra polinomial e código corretor de erros.

Para garantir a disponibilidade dos dados, foi utilizado o conceito de código corretor de erros, pois caso haja uma perda de parte dos dados será possível recuperar o arquivo original. O *Hy-SAIL* utiliza *Online Codes*, uma classe especial de *Fountain Codes*, que apresenta um avanço em relação aos outros modelos de *PoR*'s devido à sua flexibliidade no gerenciamento dos arquivos. Com *Online Codes* é possível reconstruir blocos codificados corrompidos sem a necessidade de efetuar *download* do arquivo original. Com a camada de abstração criada pelo *Hy-SAIL*, os provedores de armazenamento possuem flexibilidade na adição e/ou remoção de novas unidades de armazenamento, mesmo após o término da distribuição dos blocos codificados. Isso se deve ao fato de não haver uma distinção entre os diferentes provedores de armazenamento, os quais podem receber qualquer tipo de bloco codificado.

Para a realização de checagem de integridade dos blocos codificados, a criação de *MAC* com propriedade *XOR* homomórfica proporcionou uma evolução em relação aos outros modelos de *PoR*'s. O MAC_{hysail} possui características de função de *hash* 2-universal com segurança desmonstrável, o que garante uma propabilidade desprezível para qualquer atacante que deseje forjar um *MAC*. A disponibilidade dos blocos codificados também possui segurança desmostrável, com probabilidade desprezível no que tange a falha da recuperabilidade de tais blocos.

O *Hy-SAIL* implementa dois modelos de verificação de integridade: Limitado e Ilimitado. O modelo Limitado armazena uma quantidade de desafios e possui pouco consumo de largura de banda na resposta. Já o modelo Ilimitado armazena um único desafio e o consumo de largura de bansa corresponde ao envio de um único bloco codificado. O *Hy-SAIL* apresenta um *PoR* robusto, flexível e escalável para arquivos armazenados em ambientes não confiáveis.

Outro ponto relevante é a quantidade de informações coletadas para a realização da verificação de integridade, visto que o *Hy-SAIL* permite que o arquivo por inteiro seja verificado com apenas uma única consulta.

A implementação de um protótipo do *Hy-SAIL* em Python permitiu aferir o funcionamento e medir os desempenhos na codificação e decodificação de arquivos no esquema proposto.

6.1 TRABALHOS FUTUROS

O esquema proposto possui algumas limitações como o processamento de arquivo de tamanho da ordem de *gigabytes*. A utilização de métodos que possam paralelizar esta tarefa de codificação torna-se necessária para fornecer um nível de escalabilidade maior que o já alcançado neste trabalho. O paradigma *MapReduce* é uma alternativa para a otimização da codificação e decodificação do arquivo, visto que a sua utilização está relacionada à atividades que envolvem grande quantidades de dados.

Outro ponto a ser explorado é a possibilidade de delegar a tarefa de verificar a integridade para terceiros. O esquema proposto carece de uma abordagem com verificação pública de integridade dos blocos codificados. Tal funcionalidade permitiria desonerar recursos computacionais por parte do usuário que muitas das vezes não dispõe de recursos do porte de provedores.

Com a utilização de *Online Codes* para a aplicação do código corretor de erros, foi possível desenvolver uma metodologia que alcance condições favoráveis para a codificação e decodificação dos arquivos, porém a verificação de integridade ainda é objeto de pesquisa através de uma outra função *MAC XOR* homomórfica que atenda os requisitos de escalabilidade e otimização para uma abordagem de Computação em Nuvem.

REFERÊNCIAS BIBLIOGRÁFICAS

ALSHIBAMI, O.; BOUSSAKTA, S.; AZIZ, M. Fast algorithm for the 2-d new mersenne number transform. *Signal Processing*, n. 8, p. 1725–1735, ago. Disponível em: <http://www.sciencedirect.com/science/article/B6V18-43HT727-D/1/504431827211f768bbe977be7ee4734e>.

ANDERBERG, M. R. *Cluster Analysis for Applications*. [S.l.]: Academic Press, 1973.

ARMBRUST, M. et al. *Above the Clouds: A Berkeley View of Cloud Computing*. [S.l.], February 2009. Disponível em: <http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html>.

ATENIESE, G. et al. Provable data possession at untrusted stores. In: *Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007. (CCS '07), p. 598–609. ISBN 978-1-59593-703-2. Disponível em: <http://doi.acm.org/10.1145/1315245.1315318>.

BELLARE, M.; ROGAWAY, P. Random oracles are practical: a paradigm for designing efficient protocols. In: *Proceedings of the 1st ACM conference on Computer and communications security*. New York, NY, USA: ACM, 1993. (CCS '93), p. 62–73. ISBN 0-89791-629-8. Disponível em: <http://doi.acm.org/10.1145/168588.168596>.

BERENSON, A. *Snowden, Through the Eyes of a Spy Novelist*. 2013. Http://www.nytimes.com/2013/06/25/opinion/snowden-through-the-eyes-of-a-spy-novelist.html. Accessed: 2013-09-29.

BLODGET, H. *Amazon's cloud crash disaster permanently destroyed many customers' data*. 2011. Accessed: 2013-09-15. Disponível em: <http://www.businessinsider.com/amazon-lost-data-2011-4>.

BONEH, D.; LYNN, B.; SHACHAM, H. Short signatures from the weil pairing. In: *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*. London, UK: Springer-Verlag, 2001. (ASIACRYPT '01), p. 514–532. ISBN 3-540-42987-5. Disponível em: <http://portal.acm.org/citation.cfm?id=647097.717005>.

BORTHAKUR, D. *The Hadoop Distributed File System: Architecture and Design*. [S.l.], 2007.

BOWERS, K. D.; JUELS, A.; OPREA, A. Hail: A high-availability and integrity layer for cloud storage. In: *Proceedings of the 16th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2009. (CCS '09), p. 187–198. ISBN 978-1-60558-894-0. Disponível em: <http://doi.acm.org/10.1145/1653662.1653686>.

BOWERS, K. D.; JUELS, A.; OPREA, A. Proofs of retrievability: theory and implementation. In: *Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009. (CCSW '09), p. 43–54. ISBN 978-1-60558-784-4. Disponível em: <http://doi.acm.org/10.1145/1655008.1655015>.

CANEDO, E. D. *Modelo de confiança para a troca de arquivos em uma nuvem privada*. Tese (Doutorado) — Universidade de Brasília, Agosto 2012.

CAO, N. et al. Lt codes-based secure and reliable cloud storage service. In: GREENBERG, A. G.; SOHRABY, K. (Ed.). *INFOCOM*. IEEE, 2012. p. 693–701. ISBN 978-1-4673-0773-4. Disponível em: <http://dblp.uni-trier.de/db/conf/infocom/infocom2012.html>.

CARTER, J. L.; WEGMAN, M. N. Universal classes of hash functions (extended abstract). In: *Proceedings of the ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1977. (STOC '77), p. 106–112. Disponível em: <http://doi.acm.org/10.1145/800105.803400>.

CHANG, V. et al. A categorisation of cloud computing business models. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (CCGRID '10), p. 509–512. ISBN 978-0-7695-4039-9. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2010.132>.

CRAMER, R.; SHOUP, V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: BILARDI, G. et al. (Ed.). *Advances in Cryptology – CRYPTO'98*. [S.l.]: Springer, 1998. (Lecture Notes in Computer Science, v. 1462), p. 13–25.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

DILLON, T.; WU, C.; CHANG, E. Cloud computing: Issues and challenges. In: *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*. Washington, DC, USA: IEEE Computer Society, 2010. (AINA '10), p. 27–33. ISBN 978-0-7695-4018-4. Disponível em: <http://dx.doi.org/10.1109/AINA.2010.187>.

DODIS, Y.; VADHAN, S. P.; WICHS, D. Proofs of retrievability via hardness amplification. In: REINGOLD, O. (Ed.). *TCC*. Springer, 2009. (Lecture Notes in Computer Science, v. 5444), p. 109–127. ISBN 978-3-642-00456-8. Disponível em: <http://dblp.uni-trier.de/db/conf/tcc/tcc2009.html/DodisVW09>.

ELIAS, P. Coding for two noisy channels. In: *Information Theory, The 3rd London Symposium*. [S.l.]: Buttersworth's Scientific Publications, 1955. p. 61–76.

GANTZ, J.; REINSEL, D. Big data, bigger digital shadows, and biggest growth in the far east. In: . [s.n.], 2012. Disponível em: <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf>.

GAO, J. et al. Saas performance and scalability evaluation in clouds. In: GAO, J. Z. et al. (Ed.). *SOSE*. IEEE, 2011. p. 61–71. ISBN 978-1-4673-0411-5. Disponível em: <http://dblp.uni-trier.de/db/conf/sose/sose2011.html>.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 37, n. 5, p. 29–43, out. 2003. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1165389.945450>.

GUAJARDO, J. et al. Efficient hardware implementation of finite fields with applications to cryptography. *Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications*, v. 93, n. 1, p. 75–118, 2006.

HALEVI, S. et al. Proofs of ownership in remote storage systems. In: *Proceedings of the 18th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2011. (CCS '11), p. 491–500. ISBN 978-1-4503-0948-6. Disponível em: <http://doi.acm.org/10.1145/2046707.2046765>.

JADEJA, Y.; MODI, K. Cloud computing - concepts, architecture and challenges. In: *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*. [s.n.], 2012. p. 877–880. Disponível em: <http://dx.doi.org/10.1109/ICCEET.2012.6203873>.

JUELS, A.; KALISKI, B. S. Pors: proofs of retrievability for large files. In: *In CCS ?07: Proceedings of the 14th ACM conference on Computer and communications security*. [S.l.]: ACM, 2007. p. 584–597.

KAMARA, S.; LAUTER, K. Cryptographic cloud storage. In: *Proceedings of the 14th international conference on Financial cryptograpy and data security*. Berlin, Heidelberg: Springer-Verlag, 2010. (FC'10), p. 136–149. ISBN 3-642-14991-X, 978-3-642-14991-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1894863.1894876>.

KAUFMAN, L. M. Data security in the world of cloud computing. *IEEE Security and Privacy*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 7, n. 4, p. 61–64, jul. 2009. ISSN 1540-7993. Disponível em: <http://dx.doi.org/10.1109/MSP.2009.87>.

LILLIBRIDGE, M. et al. A cooperative internet backup scheme. In: *In Proceedings of the 2003 USENIX Annual Technical Conference*. [S.l.: s.n.], 2003. p. 29–41.

LUBY, M. Tornado codes: Practical erasure codes based on random irregular graphs. In: LUBY, M.; ROLIM, J. D. P.; SERNA, M. J. (Ed.). *Randomization and Approximation Techniques in Computer Science, Second International Workshop, RANDOM 98, Barcelona, Spain, October 8-10, 1998, Proceedings*. [S.l.]: Springer, 1998. (Lecture Notes in Computer Science, v. 1518), p. 171. ISBN 3-540-65142-X.

LUBY, M. Lt codes. In: *Proceedings of the 43rd Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2002. (FOCS '02), p. 271–. ISBN 0-7695-1822-2. Disponível em: <http://dl.acm.org/citation.cfm?id=645413.652135>.

LUBY, M. G.; MITZENMACHER, M.; SHOKROLLAHI, M. A. Analysis of random processes via and-or tree evaluation. In: *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*. [S.l.: s.n.], 1998. p. 364–373.

LUO, J. et al. Efficient software implementations of large finite fields gf(2n) for secure storage applications. *Trans. Storage*, ACM, New York, NY, USA, v. 8, n. 1, p. 2:1–2:27, fev. 2012. ISSN 1553-3077. Disponível em: <http://doi.acm.org/10.1145/2093139.2093141>.

MACKAY, D. J. C. Fountain codes. *IEE Communications*, v. 152, p. 1062–1068, 2005.

MARTINIAN, E. *Error Correcting Codes*. 2002. Http://web.mit.edu/ emin/. Accessed: 2012-12-01.

MAYMOUNKOV, P. Online codes. *Technical Report TR2002-833*, New York University, New York, NY, November 2002.

MAYMOUNKOV, P.; MAZIÈRES, D. Rateless codes and big downloads. In: KAASHOEK, M. F.; STOICA, I. (Ed.). *IPTPS*. Springer, 2003. (Lecture Notes in Computer Science, v. 2735), p. 247–255. ISBN 3-540-40724-3. Disponível em: <http://dblp.uni-trier.de/db/conf/iptps/iptps2003.html>.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, September 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.

MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of Applied Cryptography*. CRC Press, 2001. Disponível em: <http://www.cacr.math.uwaterloo.ca/hac/>.

NEPAL, S. et al. Diaas: Data integrity as a service in the cloud. In: LIU, L.; PARASHAR, M. (Ed.). *IEEE CLOUD*. IEEE, 2011. p. 308–315. ISBN 978-1-4577-0836-7. Disponível em: <http://dblp.uni-trier.de/db/conf/IEEEcloud/IEEEcloud2011.html>.

REED, I. S.; SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, SIAM, v. 8, n. 2, p. 300–304, 1960. Disponível em: <http://dx.doi.org/10.1137/0108018>.

RISTENPART, T. et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: AL-SHAER, E.; JHA, S.; KEROMYTIS, A. D. (Ed.). *ACM Conference on Computer and Communications Security*. ACM, 2009. p. 199–212. ISBN 978-1-60558-894-0. Disponível em: <http://dblp.uni-trier.de/db/conf/ccs/ccs2009.html/RistenpartTSS09>.

SHACHAM, H.; WATERS, B. Compact proofs of retrievability. In: PIEPRZYK, J. (Ed.). *Proceedings of Asiacrypt 2008*. [S.l.]: Springer-Verlag, 2008. (LNCS, v. 5350), p. 90–107.

SHOKROLLAHI, A. Raptor codes. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. SI, p. 2551–2567, jun. 2006. ISSN 1063-6692. Disponível em: <http://dx.doi.org/10.1109/TIT.2006.874390>.

SHVACHKO, K. et al. The hadoop distributed file system. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Washington, DC, USA: IEEE Computer Society, 2010. (MSST '10), p. 1–10. ISBN 978-1-4244-7152-2. Disponível em: <http://dx.doi.org/10.1109/MSST.2010.5496972>.

THAIN, D.; LIVNY, M. Building reliable clients and servers. In: FOSTER, I.; KESSELMAN, C. (Ed.). *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 2003.

UNDHEIM, A.; CHILWAN, A.; HEEGAARD, P. Differentiated availability in cloud computing slas. In: *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2011. (GRID '11), p. 129–136. ISBN 978-0-7695-4572-1. Disponível em: <http://dx.doi.org/10.1109/Grid.2011.25>.

WANG, C. et al. Ensuring data storage security in cloud computing. In: *Proceeding of the 17th IEEE International Workshop on Quality of Service (IWQoS)*. [S.l.: s.n.], 2009.

WANG, C. et al. Privacy-preserving public auditing for data storage security in cloud computing. In: *Proceedings of the 29th conference on Information communications*. Piscataway, NJ, USA: IEEE Press, 2010. (INFOCOM'10), p. 525–533. ISBN 978-1-4244-5836-3. Disponível em: <http://portal.acm.org/citation.cfm?id=1833515.1833620>.

WANG, Q. et al. Enabling public verifiability and data dynamics for storage security in cloud computing. In: *Proceedings of the 14th European conference on Research in computer security*. Berlin, Heidelberg: Springer-Verlag, 2009. (ESORICS'09), p. 355–370. ISBN 3-642-04443-3, 978-3-642-04443-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1813084.1813114>.

WEISS, A. Computing in the clouds. *Networker*, ACM, New York, NY, USA, v. 11, n. 4, p. 16–25, dez. 2007. ISSN 1091-3556. Disponível em: <http://doi.acm.org/10.1145/1327512.1327513>.

WHITE, T. *Hadoop: The Definitive Guide*. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2009. ISBN 0596521979, 9780596521974.

ZHU, Y. et al. Efficient provable data possession for hybrid clouds. In: *Proceedings of the 17th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2010. (CCS '10), p. 756–758. ISBN 978-1-4503-0245-6. Disponível em: <http://doi.acm.org/10.1145/1866307.1866421>.

APÊNDICE

Neste apêndice são mostrados os seguintes pseudo-códigos:

- •Criação MAC_{hysail} de cada bloco de mensagem m_i
- •Modelo limitada de verificações de integridade dos blocos codificados
- •Modelo Ilimitada de verificações de integridade dos blocos codificados

A Cálculo do MAC do Hy-SAIL

Pseudo-Código A.1 MAC_{hysail}: Calcula o MAC de cada bloco m_i.

Um arquivo $F = \{m_i\}_{i=1}^{n} = \{m_1(x)||m_2(x)||\ldots||m_n(x)\}$.

Uma lista aleatória de polinômios primitivos $P = \{P_1(x), P_2(x), \ldots, P_l(x)\}$

Uma lista com MAC: $MAC_{Hy-SAIL}(F) = \{h_{i,j}\}_{i=1,j=1}^{n,l} = \{h_{1,1}(x), \ldots, h_{i,j}(x)\}$.

$P(x) \longleftarrow$ PrimitivoPol($\mathbb{F}_{2^p}(x), R$).
for $i = 1$ to n
  for $j = 1$ to l
  $h_{i,j}(x) \longleftarrow m_i(x) \mod P_j(x)$
  end for
end for
$\{h_{1,1}(x), \ldots, h_{i,j}(x)\}$
Segundo etapa na fase de Codificação.
Caso seja utilizado o Modelo Ilimitado, o valor de "l" é igual a 1.

B Modelo Limitado de Auditoria

Pseudo-Código B.1 Fase de Auditoria no Modelo Limitado

Dados de Entrada:

Uma lista de código autenticador $MAC_{hysail}(x) = \{h_{1,1}(x), \ldots, h_{i,j}(x)\}$.

Uma lista de blocos codificados c_t's e polinômios primitivos $P(x)$.

Valores de $\tau_{provedor}$, τ_{minimo} e τ_{global}.

Objetivo: Verificar a integridade de blocos codificados.

Usuário

1 - O usuário escolhe um polinômio primitivo aleatório $P(x)$, de um conjunto já gerado anteriormente.

2 - O usuário escolhe um subconjunto aleatório de c_t's.

$lista \longleftarrow c_j$.

3 - O usuário envia o desafio $\{lista, P(x)\}$ para o provedor de armazenamento $\mathcal{S}_i$.

Provedor de Armazenamento

1 - $MAC_{provedor}(x) \longleftarrow (\bigoplus_{j \in lista} c_j(x)) \bmod P(x)$.

2 - Envia a resposta $MAC_{provedor}(x)$ para o usuário.

Usuário

if $\left[\left(\bigoplus_{i \in list} c_j(x) \bmod P(x)\right)\right] = MAC_{provedor}(x)$

$\tau_{provedor} \longleftarrow$ OK!!!

else if

$\tau_{provedor} < \tau_{minimo}$ **then**

Seleciona um outro conjunto de c_j's no mesmo provedor para a verificação.

Repita o algoritmo para detectar quais blocos codificados estão corrompidos.

else

Neste momento, o usuário possui 2(duas) opções:

a - Continuar a verificação de integridade dos blocos codificados em outro provedor e verificar o valor de τ_{global}.

b - Ou iniciar a fase de Redistribuição com outros blocos codificados.

end if

C Modelo Ilimitado de Auditoria

Pseudo-Código C.1 Fase de Auditoria no Modelo Ilimitado

Dados de Entrada:

Uma lista de código autenticador $MAC_{hysail}(x) = \{MAC_i\}_{i=1}^{n} = \{h_1(x), \ldots, h_n(x)\}$.

Uma lista de blocos codificados c_t's gerados na fase de Codificação.

Um único polinômio irredutível $P(x)$ armazenado com o usuário.

Valores de $\tau_{provedor}$, τ_{minimo}e τ_{global}.

Objetivo: Verificar a integridade de blocos corrompidos.

Usuário

1 - O usuário escolhe um subconjnunto aleatório de c_t's.

$lista \longleftarrow c_j$.

2 - O usuário envia o desafio $\{lista\}$ para o provedor de armazenamento $\mathcal{S}_i$.

Provedor de Armazenamento

1 - $c_t(x) \longleftarrow (\bigoplus_{j \in lista} c_j(x))$.

2 - Envia a resposta $c_t(x)$ para o usuário.

Usuário

if $\left[\left(\bigoplus_{j \in list} h_j(x)\right)\right] = c_t(x) \bmod P(x)$

A integridade está satisfatória !!!

$\tau_{provedor} \longleftarrow$ OK!!!

else if

$\tau_{provedor} < \tau_{minimo}$ **then**

Repita o algoritmo para um novo valor de $\tau_{provedor}$.

Seleciona um outro conjunto de c_j's no mesmo provedor para a verificação.

else

Neste momento, o usuário possui 2(duas) opções:

a - Continuar a verificação de integridade dos blocos codificados em outro provedor e verificar o valor de τ_{global}

b - Ou iniciar a fase de Redistribuição com outros blocos codificados

end if