Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Provendo Garantias de Confidencialidade e Integridade da Informação para Aplicações Publicar/Assinar

Fábio Fernando de Oliveira Silva

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Segurança da Informação

Andrey Elísio Monteiro Brito
(Orientador)

Campina Grande, Paraíba, Brasil

S586p Silva, Fábio Fernando de Oliveira.
Provendo garantias de confidencialidade e integridade da informação para aplicações publicar/assinar / Fábio Fernando de Oliveira Silva. - Campina Grande, 2020.
73 f. : il. color

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.
"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito.
Referências.

1. Computação na Nuvem. 2. Ambientes de Execução Confiável (Trusted Execution Environments). 3. TEEs. 4. Barramentos de Mensagens Baseados em Publicar/Assinar. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU 004.65(043)
FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO GUSTAVO DINIZ DO NASCIMENTO CRB-15/515

PROVENDO GARANTIAS DE CONFIDENCIALIDADE E INTEGRIDADE DA INFORMAÇÃO PARA APLICAÇÕES PUBLICAR/ASSINAR

FÁBIO FERNANDO DE OLIVEIRA SILVA

DISSERTAÇÃO APROVADA EM 27/03/2020

ANDREY ELÍSIO MONTEIRO BRITO, Dr., UFCG
Orientador(a)

REINALDO CÉZAR DE MORAIS GOMES, Dr., UFCG
Examinador(a)

EDUARDO DE LUCENA FALCÃO, Dr., UNIFACISA
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A adoção de computação na nuvem por empresas cresce a cada momento. Entre os principais motivos para este movimento "em direção à nuvem", destacam-se as suas características de escalabilidade e baixo custo. Com a crescente adoção do uso de serviços de provedores de computação na nuvem, aumentam também os riscos que se ocorram violações de dados, uma vez que informações de usuários, potencialmente confidenciais como informações de pagamentos, dados médicos e transações financeiras, passam a trafegar e a serem processadas em infraestruturas computacionais de terceiros. Para suportar a execução segura de software em um sistema computacional remoto administrado por terceiros, nós exploramos os ambientes de execução confiável (*Trusted Execution Environments*, ou TEEs), que é uma solução apoiada pela literatura.

Além de TEEs, outra tecnologia chave neste trabalho é a utilização de barramentos de mensagens baseados em publicar/assinar. Este paradigma de comunicação é adequado para aplicações distribuídas de larga escala pois facilita a implementação de propriedades como a escalabilidade e o baixo acoplamento. O paradigma publicar/assinar é tipicamente aplicado em sistemas com requisitos de disseminação e processamento de informações em larga escala e de forma contínua, como sistemas financeiros, sistemas de monitoramento de recursos, e aplicações de IoT. Considerando o cenário em que aplicações publicar/assinar estão sendo movidas para a nuvem, os sistemas devem ser modificados para que sejam garantidas a integridade e confidencialidade das informações neste novo ambiente administrado por terceiros. No entanto, a depender da quantidade de componentes e complexidade da aplicação em questão, as modificações necessárias podem ter um custo de desenvolvimento elevado. Neste trabalho propomos uma estratégia para prover garantias de integridade e confidencialidade para aplicações publicar/assinar, sem que seja necessário modificações nas mesmas.

Abstract

The enterprise adoption of the cloud computing is growing all the time. Among the main reasons for this movement to the cloud are its scalability and low-cost. With this growing adoption of services from cloud computing providers, the risks of data breaches also increase as potentially sensitive user information such as payment information, medical data, and financial transactions, traffic through and are processed on third-party computing infrastructures. To support the secure remote software execution on a third-party managed computing system, we exploit the trusted execution environments (TEEs), which is a solution supported by the literature.

In addition to TEEs, another key technology in this work is the use of publish/subscribe-based message buses. This communication paradigm is quite adequate for the nature of large-scale distributed applications. It facilitates the implementation of properties such as scalability and low coupling and is widely applied where there are large scale data dissemination and continuous processing, such as financial systems, resource monitoring systems, IoT applications, among others. Considering the scenario in which publish/subscribe applications are moving to the cloud, components must be modified to ensure the integrity and confidentiality of information in this new third-party-managed environment. However, depending on the number of components and complexity of the application in question, the necessary modifications can have a high development cost. In this work, we propose a strategy to provide guarantees of integrity and confidentiality for publish/subscribe applications, without requiring changes to their implementations.

Agradecimentos

Agradeço aos meus pais Genival e Dória, pela paciência e por sempre acreditarem no caminho da educação.

Agradeço à minha companheira Fran, que sempre esteve ao meu lado e me deu todo apoio nessa caminhada.

Agradeço aos meus irmãos Dayse, Bruno, Diogo e Diego, e aos meus sobrinhos Júlia, Lucas e Bruninho por sempre serem a minha inspiração.

Agradeço ao professor Dr. Andrey Brito pela orientação, paciência, oportunidades, pelas melhores ideias e por me manter motivado ao longo dessa jornada.

Agradeço aos amigos que em algum momento trabalhamos juntos, em especial Adbys, Amandio, Benardi, Clenimar, Eduardo, Gabriel Vinha (que foi), João Targino, Leo, Lilia, Lucas, Marcus Tenório, Matteus, Max, Maysa, Ricardo, Roberto, Rodolfo, Talita, Viviane, em ordem alfabética para não haver brigas. :) Com vocês aprendi bastante.

Ao Laboratório de Sistemas Distribuídos, por todo amparo profissional e tecnológico de qualidade.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao projeto GT LiteCampus (RNP) pelo financiamento desta pesquisa, e à Smartiks pela parceria e cooperação.

Sumário

Lista de Símbolos

ACL - *Access Control List*

AESM - *Application Enclave Service Manager*

API - *Application Programming Interface*

AVR - *Attestation Verification Report*

BIOS - *Basic Input/Output System*

BNF - *Backus-Naur Form*

CA - *Certification Authority*

CN - *Common Name*

CPU - *Central Processing Unit*

DN - *Distinguished Name*

DRAM - *Dynamic Random-Access Memory*

DSS - *Digital Signature Standard*

EPC - *Enclave Page Cache*

HDFS - *Hadoop Distributed File System*

IAS - *Intel Attestation Service*

IoT - *Internet of Things*

JSON - *JavaScript Object Notation*

MAC - *Message Authentication Code*

NILM - *Nonintrusive Load Monitoring*

OID - *Object Identifier*

PRM - *Processor Reserved Memory*

QE - *Quoting Enclave*

RA - *Remote Attestation*

REST - *Representational State Transfer*

RSA - *Rivest–Shamir–Adleman*

SASL - *Simple Authorization Service Layer*

SCBR - *Secure Content-Based Routing*

SCONE - *Secure CONtainer Environment*

SCRAM - *Salted Challenge Response Authentication Mechanism*

SDK - *Software Development Kit*

SGX - *Secure Guard eXtensions*

SO - Sistema Operacional

TCP - *Transmission Control Protocol*

TEE - *Trusted Execution Environment*

TLS - *Transport Layer Security*

vCPU - *Virtual Central Processing Unit*

VM - *Virtual Machine*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Neste capítulo, na Seção 1.1 são apresentadas as motivações que levaram ao desenvolvimento deste trabalho, dentro do contexto em que se insere, na Seção 1.2 são descritos os objetivos gerais e específicos e, por último, na Seção 1.3 é apresentada a organização deste documento de dissertação.

1.1 Motivação

A adoção de computação na nuvem por empresas cresce a cada momento. Entre os principais motivos para este movimento "em direção à nuvem", destacam-se as suas características de escalabilidade e baixo custo. Estima-se que, em 2020, 83% de toda a carga de trabalho de corporações serão executadas em nuvens, sendo 41% destas públicas [9]. Com a crescente adoção do uso de serviços de provedores de computação na nuvem, aumentam também os riscos que se ocorram violações de dados, uma vez que informações de usuários, potencialmente confidenciais como informações de pagamentos, dados médicos e transações financeiras passam a trafegar e a serem processadas em infraestruturas computacionais de terceiros.

Uma violação de dados pode ser definida como um incidente onde dados sensíveis, protegidos ou confidenciais podem ter sido vistos, roubados ou usados por um indivíduo não autorizado [1]. Para prevenir que tais violações ocorram, diferentes estratégias têm sido adotadas e combinadas, como o uso de canais de comunicação seguros e a criptografia de dados em descanso. Tais estratégias dificultam a violação de dados no processo de transmissão e recebimento, e no armazenamento, porém, alguém com acesso privilegiado à infraestrutura

que hospeda a aplicação, como um administrador malicioso, ainda é capaz de obter o acesso indevido à informações.

A Internet das Coisas (ou abreviadamente IoT, do inglês, *Internet of Things*) é o nome dado à rede de dispositivos computacionais interrelacionados, providos de um identificador único, que realizam a transmissão e o recebimento de informações através de uma infraestrutura de comunicação para prover serviços com valor agregado [17]. São exemplos de aplicações de IoT as redes de medidores de energia inteligentes (*smart grids*), aplicações de casas inteligentes (*smart homes*) e carros autônomos [28, 26, 24]. Uma violação de dados nesse tipo de aplicação pode expor informações potencialmente confidenciais, capazes de identificar usuários e seu comportamento [23, 42].

Estatísticas apontam que as aplicações de IoT em funcionamento possuem aproximadamente 26 bilhões de dispositivos e que produzem diariamente 4,6 *exabytes* de dados [25, 40]. Extrapola-se que em 2020 existirão conectados uma base de 30 bilhões de dispositivos de IoT [25]. Para suportar aplicação de IoT, o paradigma de comunicação publicar/assinar é adequado e largamente utilizado.

O paradigma de comunicação publicar/assinar, também conhecido como *publish/subscribe*, é bastante adequado para a natureza de aplicações distribuídas de larga escala devido às suas características de baixo acoplamento [13, 11]. Nos sistemas baseados no esquema *publish/subscribe*, os assinantes (*subscribers*) registram o seu interesse em um evento, ou um padrão de eventos e, à medida que estes eventos são produzidos e publicados pelos publicadores (*publishers*), eles são entregues aos assinantes. Esse paradigma para comunicação é bastante aplicado onde há disseminação e processamento de informações em larga escala e de forma contínua, como sistemas financeiros, sistemas de monitoramento de recursos, aplicações de IoT, entre outros.

Para a execução segura de software em um sistema computacional remoto administrado por terceiros, como um ambiente de computação na nuvem, uma solução é explorar a utilização de ambientes de execução confiável (*Trusted Execution Environments*, ou TEEs). Os ambientes de execução confiável proveem garantias de integridade e confidencialidade de dados e códigos ao carregá-los e executá-los em áreas seguras e isoladas do processador da máquina [35]. São exemplos de TEEs o Intel SGX [19] e o ARM TrustZone [5].

Considerando o cenário em que aplicações publicar/assinar estão sendo movidas para a

nuvem, os sistemas devem ser modificados para que sejam garantidas a integridade e confidencialidade neste novo ambiente, administrado por terceiros. A depender da quantidade de componentes e complexidade da aplicação em questão, as modificações necessárias podem ter um custo elevado. Entende-se por confidencialidade a proteção da informação contra acesso não autorizado, e integridade a garantia de que a informação não é alterada.

Neste trabalho propomos uma estratégia para prover garantias de integridade e confidencialidade para aplicações publicar/assinar, sem que sejam necessárias modificações nas mesmas. Para tanto, foram projetados um conjunto de componentes a serem executados junto à aplicação. Estes componentes são executados em ambientes de execução confiável e são responsáveis pelo gerenciamento de credenciais e pela confidencialidade e integridade das informações. Por fim, validamos a viabilidade de nossa proposta através da implementação dos componentes propostos e realização de experimentos.

Este estudo é parte do projeto GT-LiteCampus, financiado pela Rede Nacional de Pesquisa (RNP), que tem por objetivo principal a criação de uma plataforma de processamento para cidades universitárias, com garantias de segurança e privacidade dos dados.

1.2 Objetivos

É objetivo geral deste trabalho propor uma estratégia para prover garantias de integridade e confidencialidade para aplicações que utilizam o paradigma publicar/assinar, sem que seja necessário realizar modificações nas mesmas.

São objetivos específicos deste trabalho:

- adicionar características de confidencialidade e integridade nas informações de aplicações publicar/assinar, de forma transparente;
- desenvolver mecanismos para proteção de credenciais e senhas criptográficas, através do uso de propriedades de ambientes de execução confiável;
- elaborar um modelo de ameaça, com a finalidade de detectar quais são os pontos da aplicação que precisam de atenção;
- mitigar as ameaças apresentadas neste modelo, através do uso de componentes extra-aplicação.

1.3 Organização da Dissertação

Este trabalho de dissertação está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica utilizada para a elaboração da solução proposta neste trabalho. São apresentados conceitos importantes para o entendimento desse trabalho, como os sistemas publicar/assinar, que é o tipo de sistema suportado pela nossa proposta, a segurança de camada de transporte (TLS) que é utilizada pela nossa solução, é apresentado o Apache Kafka, que é a implementação de serviço de mensagens utilizada pela aplicação de exemplo, e, por fim, é apresentada a tecnologia de ambiente de execução confiável em hardware Intel SGX, e alguns conceitos chaves da mesma.

No Capítulo 3 é feita uma contextualização deste trabalho, onde é apresentada uma aplicação de exemplo e o seu modelo de ameaça, e são apresentados trabalhos relacionados a essa pesquisa. No Capítulo 4 é apresentada a solução proposta, baseada nos objetivos deste trabalho, onde é definida uma arquitetura e estratégias para mitigação das ameaças documentadas no modelo de ameaça. No Capítulo 5 é feita uma validação da proposta através da avaliação de desempenho da implementação dos componentes propostos, e no Capítulo 6 são discutidas as conclusões e contribuições deste trabalho.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados conceitos fundamentais para o desenvolvimento e entendimento deste trabalho. Na Seção 2.1 são apresentados os sistemas publicar/assinar, que caracterizam a classe de sistemas abarcados por este trabalho. Na Seção 2.2 é feita uma explanação sobre o protocolo de Segurança da Camada de Transporte (TLS), que é utilizado para prover características de segurança na comunicação e autenticação. Na Seção 2.3 é apresentado o barramento de mensagens Apache Kafka, largamente utilizado pela indústria e também utilizado pela aplicação de exemplo considerada neste trabalho. Na Seção 2.4 é apresentado o Intel SGX, que é uma tecnologia de segurança implementada em hardware, e explorada pela solução apresentada neste trabalho. E, por último, na Seção 2.5 apresentamos o SCONE, uma plataforma que explora a utilização do Intel SGX para a execução de aplicações de forma protegida.

2.1 Sistemas Publicar/Assinar

Também conhecidos como sistemas baseados em eventos distribuídos [14], os sistemas publicar/assinar (em inglês *publish/subscribe*, ou pub/sub) são caracterizados pelo baixo acoplamento das interações entre as diferentes entidades participantes, sendo esse um requisito de sistemas de grande escala [13]. Esse tipo de sistema é composto por publicadores (*publishers*), assinantes (*subscribers*) e um serviço de eventos, também conhecido como barramento de mensagens ou *broker*.

Os publicadores divulgam eventos estruturados para o *broker*, e assinantes expressam

interesse em eventos ou padrões de eventos específicos por meio de assinaturas [11]. Em outras palavras, publicadores publicam informações para o *broker*, e através de uma etapa de filtragem baseado nas assinaturas recebidas, é feito o roteamento e entrega dessas mensagens para os respectivos assinantes. Esse paradigma para comunicação é bastante aplicado onde há disseminação e processamento de informações em larga escala e de forma contínua, como sistemas financeiros, sistemas de monitoramento de recursos, aplicações de IoT, entre outros.

O desacoplamento provido pelos serviços de eventos em conjunto com os publicadores e assinantes ocorre em três dimensões: espaço, tempo e sincronização [13]. O desacoplamento espacial está relacionado com a ausência de comunicação direta entre os publicadores e assinantes. Mais especificamente, os publicadores sequer conhecem os endereços dos assinantes. O desacoplamento temporal está relacionado com a falta de necessidade das partes estarem interagindo ao mesmo tempo, uma vez que publicações podem ser feitas, e podem ser consumidas eventualmente. E, por fim, o desacoplamento de sincronização ocorre pois os consumidores são notificados de que existem novos eventos de maneira assíncrona, enquanto estão realizando atividades concorrentes.

Existem diferentes esquemas para assinaturas em sistemas publicar/assinar, onde os mais populares são o baseado em tópico (*topic-based*) e o baseado em conteúdo (*content-based*) [13].

O esquema **baseado em tópico** utiliza a noção de "assunto" ou "canal", onde os eventos publicados são associados a um tópico específico por seus respectivos publicadores, e os assinantes, por sua vez, assinam a tópicos. Na medida em que eventos vão sendo publicados, serão entregues para os assinantes dos tópicos o qual estes pertencem. Existem diferentes implementações de serviços de eventos que suportam o esquema baseado em tópico, como o RabbitMQ[1], HiveMQ[2] e o Apache Kafka[3].

O esquema **baseado em conteúdo** utiliza eventos estruturados, geralmente no formato chave-valor, em que o serviço de eventos considera o seu conteúdo para a entrega dos mesmos. As assinaturas, neste caso, possuem uma expressão que deve ser avaliada, pelo serviço de eventos, em relação a cada evento, para que o mesmo seja considerado de interesse ou não do assinante. Existem diferentes implementações de serviços de eventos que suportam

[1] https://www.rabbitmq.com/

[2] https://www.hivemq.com/

[3] http://kafka.apache.org/

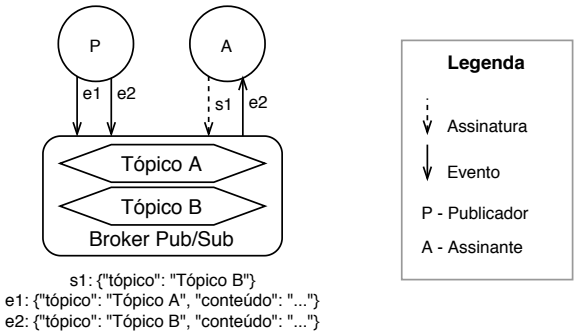

Figura 2.1: Exemplo de esquema de assinatura baseado em tópico

o esquema baseado em conteúdo, como o Elvin [37], IBM Gryphon[4], e o Siena[5].

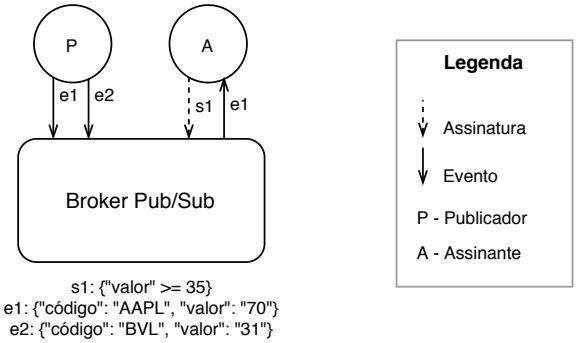

Figura 2.2: Exemplo de esquema de assinatura baseado em conteúdo

2.2 Segurança da Camada de Transporte (TLS)

O protocolo de Segurança da Camada de Transporte (do inglês *Transport Layer Security* - TLS) tem como finalidade prover um canal seguro de comunicação entre duas partes, cliente e servidor, com propriedades de **autenticação**, **confidencialidade** e **integridade** [33]. É requisito para o uso do TLS que o protocolo utilizado na camada de transporte de rede seja confiável e que garanta a entrega dos pacotes na mesma ordem em que foram enviados, como por exemplo o protocolo TCP.

A autenticação da identidade das partes é realizada através do uso de criptografia de chave pública (RSA [34], DSS [29], etc.), onde, de acordo com a especificação mais recente,

[4]https://engineering.purdue.edu/ sbagchi/research_ibm.html

[5]https://www.inf.usi.ch/carzaniga/siena/

o lado do servidor deve sempre ser autenticado pelo cliente, cabendo à implementação do servidor a decisão de autenticar ou não o cliente.

Para estabelecer o canal de comunicação seguro, ambas as partes da conexão devem concordar em um conjunto de protocolos (*ciphersuite*) que irá determinar o método de troca de chaves, o algoritmo de criptografia simétrica e o de autenticação de mensagens (ou MAC, do inglês *Message Authentication Code*). O protocolo TLS especifica uma sequência de passos bem definida, também conhecida como *handshake*, para realizar essa negociação. Através do uso de criptografia de chave pública, é possível negociar uma chave secreta compartilhada sem precisar haver nenhum conhecimento prévio entre as partes bem como o uso de um canal de comunicação não criptografado. Todo o tráfego de dados realizado após o processo de *handshake* é confidencial e íntegro, alcançados através do uso de criptografia e autenticação de mensagens, utilizando os parâmetros negociados.

Para associar a identidade à uma chave pública são utilizados certificados de chave pública, formatados de acordo com a recomendação X.509 [22]. De acordo com a especificação, um certificado é um documento que atesta que algo é verdadeiro ou quem é o proprietário de algo. Assim, este certificado atesta a relação de posse entre uma entidade qualquer e uma chave pública. Os certificados de chave pública podem ser auto-assinados ou emitidos e assinados por entidades/autoridades reconhecidas e confiadas por alguma comunidade de usuários, conhecidas como autoridades certificadoras (ou CAs, do inglês *Certification Authorities*). São exemplos de CAs populares a VeriSign, Entrust, Thawte e Geotrust. Uma nova CA pode ser definida a partir da criação de um certificado raíz auto-assinado, porém, os certificados emitidos por esta nova entidade só serão reconhecidos como válidos a partir da inclusão do certificado de chave pública desta CA na cadeia de certificados das CAs confiáveis do sistema que está realizando a validação.

```
1 Certificate ::= SIGNED { CertificateContent }
2
3 CertificateContent ::= SEQUENCE {
4     version                 Version DEFAULT v1,
5     serialNumber            CertificateSerialNumber,
6     signature               AlgorithmIdentifier{{SupportedAlgorithms}},
7     issuer                  Name,
8     validity                Validity,
```

```
9     subject                   Name,
10    subjectPublicKeyInfo      SubjectPublicKeyInfo,
11    issuerUniqueIdentifier    IMPLICIT UniqueIdentifier OPTIONAL,
12                              -- if present, version shall be v2 or v3
13    subjectUniqueIdentifier   IMPLICIT UniqueIdentifier OPTIONAL,
14                              -- if present, version shall be v2 or v3
15    extensions                Extensions OPTIONAL
16                              -- If present, version shall be v3 --
17 }
```

Código Fonte 2.1: Especificação parcial ASN.1 do certificado X.509

No Código Fonte 2.1 é apresentada uma especificação parcial, em ASN.1, de um certificado de chave pública X.509. Segue uma breve descrição de cada campo do certificado:

- **version** - um número que identifica a versão da especificação do certificado, onde a versão atual é '3';
- **serialNumber** - um número de série único, atribuído pela CA;
- **signature** - um identificador de objeto (do inglês *Object Identifier*, ou OID), utilizado para especificar o algoritmo de assinatura utilizado para assinar o certificado, por exemplo o "`sha256WithRSAEncryption`" e o "`sha384WithRSAEncryption`";
- **issuer** - o nome distinto (do inglês *Distinguished Name*, ou DN) da entidade que emitiu e assinou o certificado;
- **validity** - especifica o período de validade do certificado;
- **subject** - o DN que identifica o dono do certificado. Em um certificado auto-assinado o campo *subject* contém o mesmo DN encontrado no campo *issuer*;
- **subjectPublicKeyInfo** - contém a chave pública e informações sobre a mesma, como o OID do algoritmo utilizado e comprimento da chave (em bits);
- **issuerUniqueIdentifier** - presente a partir da versão 2, e opcional, contém informações que identificam unicamente o emissor;

- **subjectUniqueIdentifier** - presente a partir da versão 2, e opcional, contém informações que identificam unicamente o dono do certificado;

- **extensions** - presente a partir da versão 3, e opcional, permite adicionar novos campos ao certificado sem ser necessário modificar sua estrutura, como por exemplo o *Subject Alternative Name* (SAN), que pode ser preenchido com múltiplos nomes de hosts, e-mails, IPs, entre outros, permitindo assim utilizar um único certificado para representar diferentes recursos.

Um exemplo de certificado de chave pública X.509 pode ser encontrado no Código Fonte B.1, no Apêndice B.

2.3 Apache Kafka

Desenvolvido pelo LinkedIn[6] e posteriormente doado em 2011 para a Apache Software Foundation[7], o Apache Kafka é um sistema publicar/assinar de código aberto [16]. Ele utiliza o esquema baseado em tópicos, e pode ser executado na forma de *cluster*, neste caso, sendo distribuído em múltiplas máquinas. Possui também suporte à replicação [3].

O Apache Kafka é utilizado por diversas companhias[8], para diferentes casos de uso, por exemplo:

- **LinkedIn**[6] – é usado no LinkedIn para disseminação de informações de atividades e métricas operacionais. Isso alimenta vários produtos da empresa como o LinkedIn Newsfeed, o LinkedIn Today, além de sistemas de análise de dados offline como o Hadoop;

- **Uber**[9] – o Kafka é um componente fundamental na infraestrutura geral e alimenta vários sistemas online, para diversos casos de uso em tempo real;

- **Shopify**[10] – o Apache Kafka é responsável pelos *logs* de acesso, eventos de teste A/B, eventos de domínio ("uma compra aconteceu", etc.), métricas, entrega para o *Hadoop*

[6] https://www.linkedin.com/
[7] http://www.apache.org/
[8] https://kafka.apache.org/powered-by
[9] https://eng.uber.com
[10] https://www.shopify.com

Distributed File System (HDFS), relatórios de clientes, ferramentas de suporte e análise de fraude.

2.3.1 Terminologia

Esta seção contém a terminologia utilizada pelo Apache Kafka.

- ***Broker*** – enquanto sistema distribuído, o Kafka executa na forma de *clusters*. Cada nó de um *cluster* é chamado de *broker*.
- **Tópicos** – é o nome da abstração para o qual as mensagens são armazenadas e publicadas. Todas as mensagens Kafka são organizadas em tópicos, e cada tópico está associado a um nome.
- **Produtor** – entidades que publicam mensagens em tópicos.
- **Consumidor** – entidades que consomem mensagens de tópicos.

2.3.2 Tópicos e Partições

No Apache Kafka, tópico é o nome dado à entidade para a qual as mensagens são publicadas e consumidas, e é dividido em partições. As partições dividem os dados de um tópico, para permitir a escrita de vários produtores e leituras de vários consumidores no mesmo. Toda mensagem em uma partição possui um identificador chamado *offset*, que é sequencial, monotonicamente crescente e denomina a posição de uma mensagem numa partição. Consequentemente, toda mensagem pode ser identificada unicamente em um *cluster* Kafka através da 3-upla (*Tópico*, *Partição*, *Offset*).

Um tópico possui um fator de replicação, que determina a quantidade de réplicas, que será distribuída nos *brokers* do *cluster*. Um *broker* pode ser eleito o líder de uma determinada partição, o que implica que todas as leituras e escritas em uma partição devem ser feitas exclusivamente a partir dele. É o líder que também coordena a atualização das réplicas com novas mensagens, e se um líder falha, uma réplica do *broker* se torna o novo líder. A Figura 2.4 apresenta um tópico com fator de replicação igual a 2, e suas lideranças. Essa divisão de trabalho permite uma maior escalabilidade do sistema.

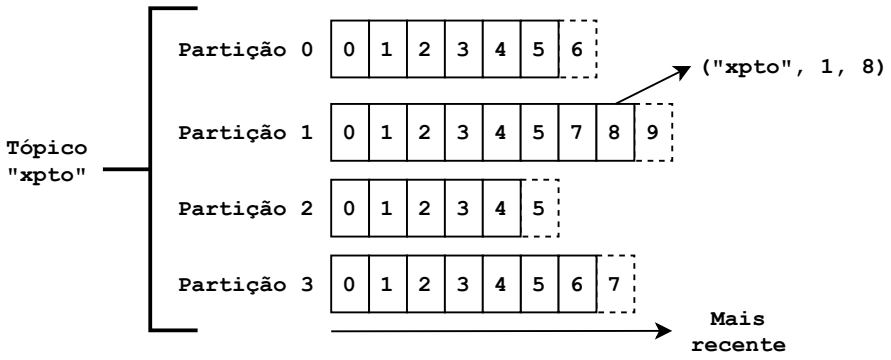

Figura 2.3: Anatomia de um tópico Kafka

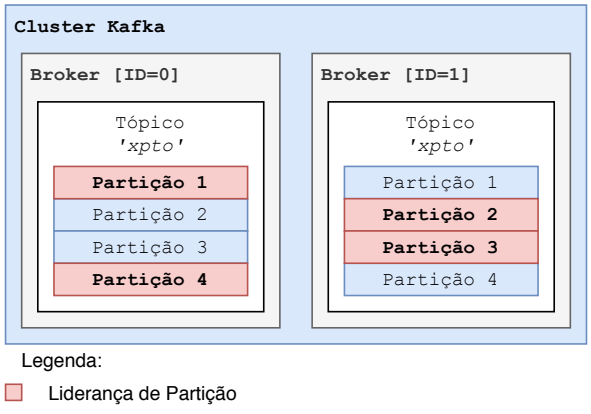

Figura 2.4: Exemplo de um tópico Kafka com replicação

Os produtores publicam mensagens para um tópico através dos líderes de suas partições. Supondo que os líderes das partições são *brokers* distintos, a vazão de produção é aumentada, uma vez que ocorre um balanceamento da carga. Os *brokers* e partições vão ser escolhidos com base na resposta de uma requisição de metadados (*MetadataRequest*) feita para um dos *brokers* do cluster. Na resposta está contida a lista de *brokers*, partições e lideranças. A escolha da partição é feita utilizando uma heurística definida pelo produtor.

De forma simétrica aos produtores, os consumidores leem mensagens dos tópicos através dos líderes de suas partições, provendo aumento de vazão da mesma forma que ocorre com os produtores, quando os líderes são *brokers* distintos. Consumidores se organizam na forma de grupos de consumidores (*ConsumerGroups*) para um dado tópico, onde cada consumidor dentro do grupo lê de uma única partição, de maneira que o grupo como um todo consome todas as mensagens de um tópico.

É delegado a um dos *brokers* do *cluster* Kafka o papel chamado coordenador de grupo (*GroupCoordinator*), para cada novo grupo de consumidores que passa a fazer parte do sistema. Dentre as responsabilidades, o coordenador de grupo detecta através do uso de pacotes de controle, se todos os consumidores de um grupo estão em funcionamento. Caso algum venha a parar de funcionar, ele é responsável por engatilhar o rebalanceamento através do líder do grupo.

É considerado líder do grupo o primeiro consumidor a juntar-se ao grupo. O líder recebe do coordenador de grupo uma lista de todos os consumidores do grupo, e é responsável por atribuir um subconjunto de partições a cada consumidor. Se existem mais consumidores do que partições, alguns deles ficarão em espera, mas se existem mais partições do que consumidores, alguns consumidores irão ler mensagens de mais de uma partição. A atuação do líder do grupo é feita de forma transparente, de acordo com a implementação do cliente ou biblioteca Kafka utilizada pela aplicação.

A Figura 2.5 apresenta algumas configurações de grupos de consumidores com diferentes balanceamentos.

2.3.3 Consistência e Disponibilidade dos Dados

Sobre consistência e disponibilidade dos dados, o Kafka garante:

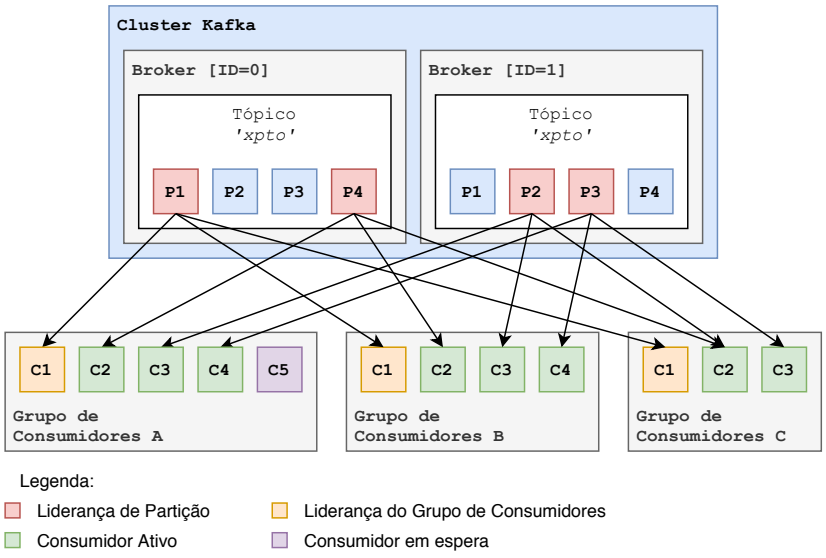

Figura 2.5: Balanceamento de grupos de consumidores

- mensagens enviadas para uma partição serão anexadas à partição na ordem em que foram enviadas;
- uma instância do consumidor irá receber as mensagens na ordem em que elas aparecem na partição;
- uma mensagem é dita entregue (*committed*) quando todas as réplicas sincronizadas anexaram a mensagem ao seu registro. Nenhuma mensagem entregue será perdida, desde que pelo menos uma réplica sincronizada esteja ativa.

2.3.4 Mecanismos de Segurança

Em sua configuração padrão, o Apache Kafka permite o acesso de qualquer cliente sem a necessidade de autenticação, podendo este publicar ou consumir informações em tópicos sem autorização, e todas as informações em trânsito através do canal de comunicação são visíveis. Em um modelo onde há a participação de diferentes clientes Kafka, publicando e consumindo informações, com o tráfego de informações possivelmente sensíveis, existe a necessidade da utilização de mecanismos para implementar o controle de acesso ao sistema.

A partir da versão 0.9 do Apache Kafka, diferentes recursos de segurança vêm sendo introduzidos para proteger *clusters* Kafka de acessos indevidos e vazamentos de informações. Através do uso desses mecanismos é possível se obter uma comunicação segura entre os

clientes e *brokers*, e restringir quem e como podem ser acessados os tópicos. Estes recursos podem ser agrupados nas categorias: criptografia, autenticação e autorização.

Comunicação Segura com TLS

O Kafka, por meio do uso de canais de comunicação protegidos através do uso de TLS, garante que os dados transferidos entre os brokers e clientes sejam criptografados, além de também proteger contra Ataques do Homem do Meio (do inglês, *man-in-the-middle*).

Com o uso de TLS, apenas as duas pontas da comunicação são capazes de descriptografarem o conteúdo do canal de comunicação, conforme visto na Seção 2.2. Um atacante capaz de interceptar e observar os dados que foram enviados pelas pontas da comunicação, dificilmente irá extrair o conteúdo original das mensagens, levando em consideração que sejam utilizados algoritmos criptográficos que não possuam vulnerabilidades conhecidas e os certificados utilizados e suas chaves privadas não tenham sidos de alguma maneira vazados.

Autenticação (TLS & SASL)

Os recursos de autenticação do Kafka têm como objetivo garantir que apenas os clientes que podem provar sua identidade podem conectar ao *cluster* Kafka. A autenticação pode ser realizada através do uso de certificados de chave pública X.509 ou através de autenticação SASL.

A autenticação TLS ocorre através da utilização de certificados pelos clientes, estes emitidos por uma autoridade certificadora (CA) de confiança do *broker*, e que irá permitir o Kafka verificar a identidade dos clientes. Os nomes de usuário são derivados a partir do campo ***subject*** do certificado de chave pública X.509.

Já na autenticação SASL, acrônimo para *Simple Authorization Service Layer*, o mecanismo de autenticação é separado do protocolo Kafka, e que podem ser:

- **PLAIN**: este mecanismo utiliza nome de usuário/senha para autenticação. Estes nomes de usuários e senhas devem ser guardados nos *brokers* Kafka com antecedência, e a cada alteração o *cluster* deve ser reiniciado;
- **SCRAM (*Salted Challenge Response Authentication Mechanism*)**: similar ao método anterior, a diferença é que os nomes de usuários e *hashs* de senhas são armazena-

dos combinados com um dado adicional aleatório (*salt*);

- **Kerberos**: como o nome sugere, utiliza uma implementação de Kerberos para autenticação (ex., Microsoft Active Directory).

Autorização

Uma vez que a autenticação está configurada, é através do uso de listas de controle de acesso (*Access Control List*, ACL) que são definidas quais operações podem ser realizadas e por quem. Através de ACLs, por exemplo, é possível definir quais usuários podem consumir ou publicar mensagens em um determinado tópico, quem pode criar novos tópicos. É através de ACLs que também são definidas as operações entre *brokers*, por exemplo restringindo quais *brokers* fazem parte do *cluster*. Em linhas gerais, uma entrada na lista de controle de acessos pode ser lida como: "O usuário **U** [está/não está] autorizado a realizar a operação **O**, a partir do *host* **H** no recurso **R**, coincidindo o padrão de recurso **RP**".

```
1 $ ./kafka-acls --bootstrap-server localhost:9092 \
2                --add --allow-principal User:Bob \
3                --consumer --allow-host 198.51.100.1 \
4                --topic test-topic
```

Código Fonte 2.2: Exemplo de inclusão de ACL

No exemplo do Código Fonte 2.2, podemos interpretar como: "O usuário **Bob** está autorizado a consumir do *host* **198.51.100.1** no recurso **tópico**, coincidindo o padrão de recurso **test-topic**". A lista completa de operações e tipos de recursos que podem ser gerenciados através de ACLs encontram-se na Tabela A.1, no Apêndice A.

2.4 Intel SGX

O Intel *Software Guard eXtensions* (SGX) é um conjunto de extensões introduzidas em CPUs da Intel, que têm como objetivo prover garantias de integridade e confidencialidade. O Intel SGX foi desenvolvido como o intuito de resolver o problema de computação remota segura, ou seja, a execução segura de software em um sistema computacional remoto administrado por terceiros [19].

Considerando este modelo de ameaça, se assume que todo o software executado em um sistema remoto é potencialmente malicioso, o que inclui o próprio sistema operacional (SO), o hipervisor no caso de máquinas virtuais (VMs), ou até mesmo a BIOS. O SGX fornece proteção a programas executados em nível de usuário contra software não confiável em execução no nível do usuário, bem como em níveis de privilégios mais altos [10].

As garantias de confidencialidade e integridade providas pelo SGX se dão por meio do uso de hardware confiável, que isola um programa em nível de usuário em um contêiner seguro chamado enclave, onde, através de um processo de atestação de software, é capaz de fornecer uma prova ao usuário de que ele está executando software não modificado e protegido em um enclave por um hardware seguro [19]. Para fornecer a atestação de software, o sistema fornece uma assinatura criptográfica certificando o *hash* de medição de um enclave (MRENCLAVE), que é calculado no conteúdo das páginas de memória de um enclave, após todos os dados necessários terem sido carregados da memória desprotegida em páginas de enclave protegidas na Memória Reservada do Processador (em inglês *Processor Reserved Memory*, ou PRM), e o processo de inicialização foi concluído [10].

O usuário remoto pode verificar essa assinatura com base em um certificado de assinatura fornecido pela Intel e pode se recusar a carregar seus dados em um enclave cujo MRENCLAVE não corresponda ao valor esperado [19, 10]. Sob este modelo de ameaça, é necessário que o usuário remoto confie no fabricante do hardware responsável por fornecer o hardware seguro e o certificado de assinatura.

Os meta-dados e conteúdos de um enclave são armazenados em um subconjunto da PRM, chamado de *Enclave Page Cache* (EPC), este dividido em páginas de 4 KB. O gerenciamento de páginas de enclave é delegado ao SO host ou hipervisor não confiável, que pode gerenciar páginas por meio de uma interface limitada por meio do uso de instruções de microcódigo SGX, o que limita o acesso direto à essa região de memória. As páginas do enclave são criptografadas enquanto estão na Memória de Acesso Aleatório Dinâmica (em inglês *Dynamic Random-Access Memory, ou DRAM*) e são descriptografadas no hardware conforme são carregadas no cache pela CPU.

Além disso, as decisões de gerenciamento de memória feitas pelo software do sistema são rastreadas para manter o isolamento da memória protegida, verificando se as páginas do enclave só podem ser acessadas pelo código protegido que está sendo executado no enclave

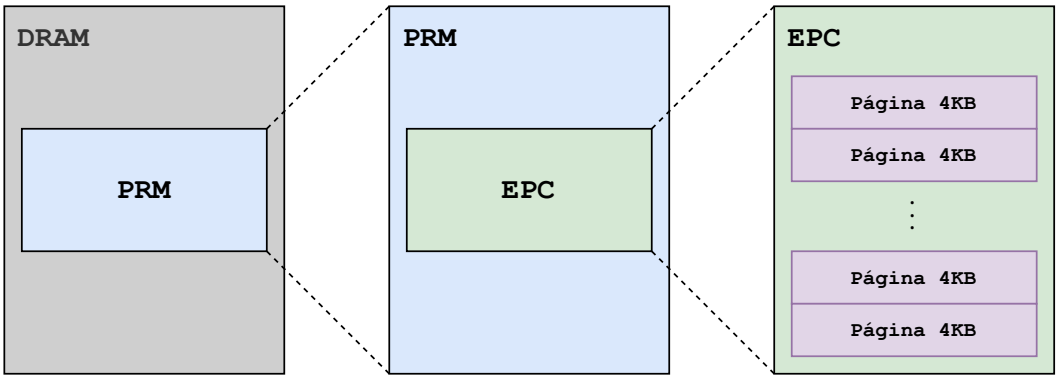

Figura 2.6: Hierarquia de memória do Intel SGX

associado a uma determinada página deste [19, 10, 27]. Vale destacar que a quantidade máxima de memória reservada ao uso pelo PRM na maioria dos processadores disponíveis no mercado é de $128\ MB$, tornando este um recurso que deve ser utilizado de maneira eficiente. Apenas em novembro de 2019 foram lançados os primeiros processadores com suporte à alocação de $256\ MB$ para PRM [20].

Para utilizar os recursos providos pelo SGX, as aplicações devem ser desenvolvidas utilizando ferramentas específicas. A Intel fornece um kit de desenvolvimento de software para esta finalidade, o Intel SGX SDK [21]. Com o SGX SDK é possível desenvolver enclaves utilizando as linguagens de programação C e C++, com suporte a um subconjunto dos serviços disponíveis na biblioteca padrão de C (`libstdc`) e na biblioteca padrão de C++ (`libstdc++`). Além do SGX SDK, outras ferramentas também permitem a criação de aplicações SGX, como o SCONE [7], o Graphene-SGX [41] e o Open Enclave[11].

2.4.1 Medições de Enclaves

No SGX, medição de enclave é o nome dado ao processo em que se é estabelecida uma identidade do mesmo. Todo enclave possui duas medições distintas: o MRENCLAVE e o MRSIGNER.

O **MRENCLAVE** está relacionado com a identidade do enclave. É o resultado de uma função *hash* SHA-256 aplicada sobre o registro de todas as atividades enquanto o enclave é carregado. Fazem parte desse registro a posição relativa de páginas de memória no enclave, as *flags* de segurança associadas a essas páginas, e o seu conteúdo (código, dados, pilha e

[11] https://openenclave.io/sdk/

heap).

O **MRSIGNER** está relacionado com a procedência do código. É o resultado de uma função *hash* SHA-256 aplicada sobre a chave pública da identidade de quem assinou o enclave no momento de sua construção, tipicamente o seu desenvolvedor.

2.4.2 Atestação Remota de Enclaves

O processo de atestação remota (*Remote Attestation*, ou RA) permite provar que um determinado programa está sendo executado dentro de um enclave SGX, bem como checar a sua identidade (MRENCLAVE) e a sua procedência (MRSIGNER).

Para que tal processo seja realizado, é necessário que o *Quoting Enclave* (QE) esteja sendo executado na plataforma remota. O QE é parte integrante do gerente de serviços de enclave de aplicações (*Application Enclave Service Manager*, AESM). Ele é responsável pela emissão do *QUOTE*, que é uma estrutura que contém, entre outras informações, o MRENCLAVE e MRSIGNER do enclave que está sendo atestado, e que é assinada com uma chave assimétrica acessível apenas por ele. Além do QE, a Intel provê um serviço de atestação (*Intel Attestation Service*, IAS), que é capaz de verificar a autenticidade da assinatura de um *QUOTE*.

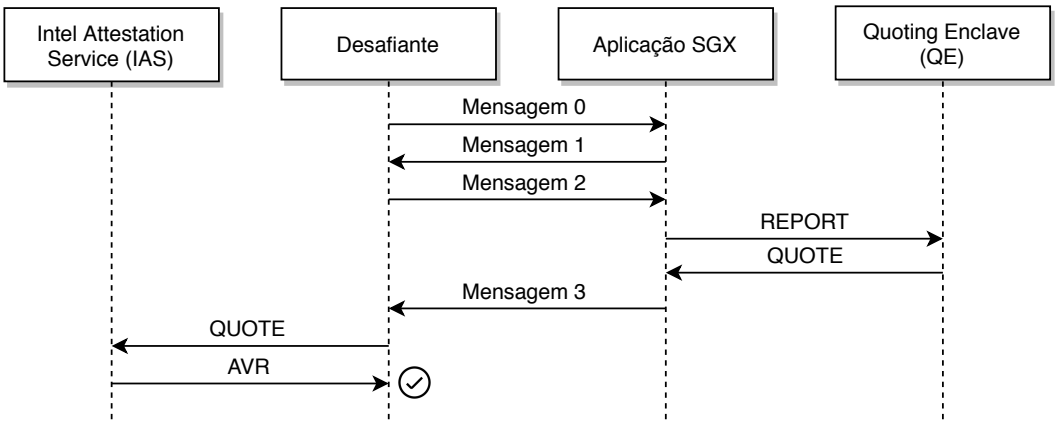

Figura 2.7: Atestação remota

A Figura 2.7 ilustra, em alto nível, o processo de atestação remota, que acontece da seguinte maneira:

1. Um desafiante estabelece uma conexão com a aplicação SGX. É solicitada uma prova

de que a aplicação está sendo executada dentro de um enclave SGX. A **mensagem 0** representa essa solicitação, e contém a chave pública do desafiante.

2. Ao receber a solicitação, a aplicação SGX cria a **mensagem 1** contendo a sua chave pública e a envia para o desafiante.

3. O desafiante, em posse da chave pública da aplicação SGX, deriva a chave simétrica SMK através do uso do método *Diffie-Hellmann*, usando a sua chave privada e a chave pública da aplicação SGX. Após derivar a chave simétrica, o desafiante cria e envia a **mensagem 2**, que contém:

   a) a chave pública do desafiante;

   b) o identificador de provedor de serviços (*Service Provider ID*, ou SPID), emitido previamente, durante o processo de registro para uso do IAS;

   c) a concatenação de ambas as chaves públicas (do desafiante e da aplicação SGX) assinada com a chave privada do desafiante;

   d) um código de autenticação de mensagem (MAC) de toda a mensagem, usando a chave simétrica SMK derivada, que pode ser verificado pelo enclave.

4. A aplicação SGX gera uma estrutura chamada *REPORT* e a envia para o QE. Essa estrutura contém todas as informações necessárias para a emissão de um *QUOTE*.

5. O QE devolve para a aplicação SGX um *QUOTE* assinado.

6. A aplicação SGX prepara a **mensagem 3** contendo o QUOTE e o seu MAC, que foi criado usando a chave simétrica SMK, e a envia para o desafiante.

7. O desafiante verifica o MAC, para garantir que ainda está se comunicando com a mesma aplicação SGX remota. Após essa verificação, o *QUOTE* é enviado para o IAS.

8. O IAS verifica a estrutura do *QUOTE* e sua assinatura. É enviado como resultado o relatório de verificação de atestação (*Attestation Verification Report*, ou AVR).

9. Após a validação do IAS, o desafiante verifica se as medições de enclave contidos no *QUOTE* eram os esperados, completando o processo de atestação remota.

Completado o processo de atestação remota com sucesso, o desafiante e a aplicação SGX podem utilizar a chave simétrica SMK para troca de mensagens segura.

2.5 SCONE

O SCONE [7] (*Secure CONtainer Environment*) é uma plataforma que facilita a execução de aplicações de forma protegida, utilizando enclaves SGX. O SCONE é baseado em uma versão modificada da biblioteca `musl libc`[12], permitindo que aplicações possam se beneficiar dos recursos de proteção em hardware providas pelo Intel SGX, sem que sejam necessário a realização de modificações em seu código-fonte.

Uma das implicações do uso do SCONE, por exemplo, é a possibilidade da utilização de outras linguagens de programação como C, C++, Python, Go, Fortran, RUST, entre outras, ao contrário do SGX SDK que suporta apenas C e C++. Para utilizar o SCONE, é necessário que o *host* utilize o sistema operacional Linux, possua suporte ao Intel SGX e um ambiente Docker configurado.

Assim como as aplicações desenvolvidas utilizando o Intel SGX SDK, as aplicações que utilizam SCONE também suportam atestação remota, o que permite a verificação da identidade da aplicação. Através do uso do SCONE *Configuration and Attestation Service* (CAS), além da verificação de identidade, é possível provisionar configurações e segredos para serem entregues à aplicação após o processo de atestação remota, de maneira segura.

[12] https://www.musl-libc.org/

Capítulo 3

Contextualização

Neste capítulo é feita uma contextualização da pesquisa. Na Seção 3.1 é apresentada a aplicação LiteMe, que utiliza o paradigma publicar/assinar. A partir dessa aplicação de exemplo, na Seção 3.2 são feitas considerações e elaborado um modelo de ameaças. E, por último, na Seção 3.3 são apresentados trabalhos relacionados à essa pesquisa.

3.1 Aplicação de Exemplo: LiteMe

O LiteMe é um sistema de apoio à gestão de consumo de energia elétrica. Ele possui funcionalidades que permitem aos usuários, por exemplo, acompanhar o histórico de consumo com diferentes resoluções de tempo, identificar o consumo por equipamento sem que seja necessário um sensor por equipamento, etc.

Esta aplicação foi projetada utilizando o modelo publicar/assinar, onde os diferentes componentes interagem através do uso de um barramento de mensagens Kafka. Os componentes da aplicação utilizam certificados de chave pública X.509 para autenticação. Uma visão geral da sua arquitetura pode ser vista na Figura 3.1. Através do barramento de mensagens são disseminadas informações potencialmente confidenciais. Aspectos relacionados a confidencialidade e integridade da informação devem ser observados.

A escolha desta aplicação foi feita baseada no contexto em que esta pesquisa está inserida, parte do projeto GT LiteCampus (RNP), que propõe uma plataforma de processamento com garantias de segurança e privacidade e uma aplicação de gerência de consumo de energia elétrica e água. O LiteMe é um produto da Smartiks Tecnologia da Informação LTDA.,

participante do projeto.

Arquitetura e Componentes

Segue uma breve descrição dos diferentes componentes presentes na arquitetura da aplicação.

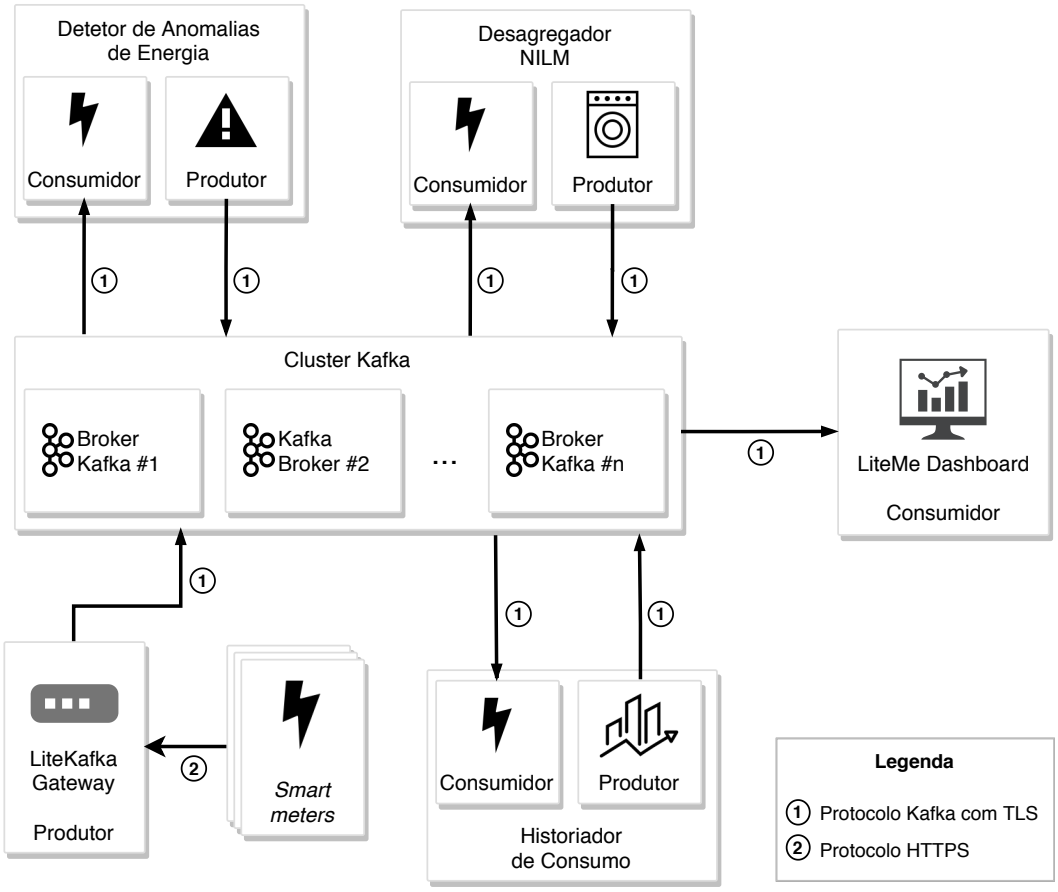

Figura 3.1: Arquitetura da Aplicação LiteMe

Smart meters

Também conhecidos como medidores de energia inteligentes, são dispositivos computacionais capazes de monitorar características de uma rede elétrica, e transmiti-las através de um canal de comunicação. Neste caso, os *smart meters* estão instalados nos quadros elétricos de cada cliente, e transmitem as informações relativas ao seu consumo para o LiteKafka

Gateway, através do uso de uma conexão segura HTTPS (HTTP sobre TLS). Cada medidor possui identificador único e um certificado de chave pública que o identifica.

LiteKafka Gateway

Provê uma API REST, utilizada pelos medidores para publicarem as suas informações. Ele é necessário pois os medidores, por serem dispositivos limitados, não suportam a complexidade do protocolo Kafka [4]. Essas informações, na medida que são recebidas, são transformadas em mensagens Kafka e publicadas em um tópico no barramento de mensagens.

Cluster Kafka

O cluster Kafka é um conjunto de *brokers*, instalações do Apache Kafka com propriedades de replicação. Essa replicação, além de prover o aumento da disponibilidade, aumenta também a vazão do sistema, uma vez que os publicadores e assinantes devem ser capazes de se conectarem simultaneamente a todos os *brokers* que são líderes das partições do tópico de interesse dessas aplicações.

Detetor de Anomalias de Energia

Possui a função de analisar o perfil de consumo de energia do cliente e emitir alertas quando são detectadas anomalias relacionadas à própria qualidade da energia, bem como quando há algum comportamento de consumo fora do padrão do cliente. Este componente é assinante do tópico de medições, publicadas pelo LiteKafka Gateway, e publica os alertas em um tópico específico para os mesmos.

Desagregador NILM

Possui a função de identificar, dado um perfil de consumo, quais equipamentos estão consumindo energia naquele instante, através da aplicação de uma técnica conhecida por monitoramento de carga não-intrusivo (Nonintrusive Load Monitoring, ou NILM). Este componente é assinante do tópico de medições, publicadas pelo LiteKafka Gateway, e publica as decomposições em um tópico específico para as mesmas.

Historiador de Consumo

Este componente tem a função de agregar conjuntos de medições para diferentes intervalos de tempo. Este componente é assinante do tópico de medições, publicadas pelo LiteKafka Gateway, e publica as agregações em um tópico destinado para as mesmas.

LiteMe Dashboard

É a interface com o usuário final, onde é agregada todas as funcionalidades da aplicação. Este componente é assinante de praticamente todos os tópicos publicados pelos diversos componentes da aplicação.

3.2 Considerações e Modelo de Ameaça

Com base na aplicação de exemplo LiteMe, foram enumeradas algumas considerações sobre as aplicações das quais a abordagem proposta neste trabalho se aplica e se limita:

1. as aplicações devem ser baseadas no modelo publicar/assinar com o esquema de assinaturas baseado em tópico, e utilizar certificados de chave pública X.509 para autenticação;

2. o domínio de proteção dos dados publicados é o tópico, onde cada tópico deverá ser isolado dos demais através do uso de mecanismos de criptografia e autenticação;

3. o principal tipo de mensagens a ser transportado pelo barramento é JSON com tamanho médio de 1 KB (por exemplo, informações de consumo e qualidade de energia elétrica);

4. deve ser suportada a conexão simultânea com múltiplos *brokers* replicados, já que em ambientes de produção esta é a regra.

O nosso modelo de ameaça assume o adversário como sendo alguém que possui nível de acesso privilegiado na infraestrutura computacional remota que está executando o *cluster* Kafka. Ele tem acesso a todo o software que está sendo executado nas máquinas, bem como a todos os dados em repouso, que o permite realizar a alteração, visualização e replicação

dessas informações. O Apache Kafka, que é barramento de mensagens utilizado pela aplicação LiteMe, não foi projetado considerando este modelo de adversário, podendo serem comprometidas a confidencialidade e integridade dos dados através de um ataque feito por este tipo de adversário.

Também se assume que as credenciais de acesso ao *broker* dos sistemas publicadores e assinantes podem ser vazadas. De forma semelhante, novas credenciais poderiam ser criadas e autorizadas a acessar tópicos sensíveis pelo administrador da infraestrutura ou por alguém que conseguiu controlar a sua identidade. Isso permitiria o ingresso de publicadores e assinantes maliciosos no barramento de mensagens, que poderiam além de vazar dados, publicar mensagens inválidas ou falsas, prejudicando o funcionamento da aplicação.

3.3 Trabalhos Relacionados

O paradigma publicar/assinar se apresenta como uma solução adequada para ser utilizado na construção de aplicações distribuídas e de larga escala, devido às suas propriedades de desacoplamento de espaço, tempo e sincronização [13]. Apesar de ser possível utilizar certificados de chave pública para autenticação [18] das entidades participantes de um sistema publicar/assinar, o modelo de ameaça apresentado na Seção 3.2 considera que essa abordagem isoladamente não é uma solução suficiente.

O PubSub-SGX [6] e o SCBR [31] apresentam implementações de sistemas publicar/assinar baseado em conteúdo explorando os recursos de proteção em hardware providos pelo Intel SGX para garantir a confidencialidade e a integridade dos dados, bem como manter anônima a identidade dos publicadores e assinantes. Nessas implementações, as etapas de filtragem e roteamento são feitas dentro de enclaves SGX. No projeto SecureCloud[1], uma solução baseada no SCBR foi desenvolvida para disseminar dados, inclusive dados de energia. No entanto, esta abordagem sofre problemas de escalabilidade, já que é baseada no SCBR, que executa em uma única máquina e é orientado para dados de controle. Além disso, ele usa o modelo baseado em conteúdo, que requer que toda a infraestrutura de roteamento (*brokers*) execute completamente dentro de ambientes confiáveis. No nosso trabalho exploramos o uso de ambientes de execução confiáveis para garantir confidencialidade e integridade dos

[1] https://www.securecloud-project.eu

dados, mas sem modificar os componentes da aplicação. A identidade dos publicadores e assinantes não estão no escopo do nosso trabalho.

O Graphene-SGX [41] e o SGX-LKL [32] são bibliotecas de sistema operacional, que permitem a execução de binários não-modificados em enclaves SGX. Ao contrário do SCONE [7], que foi introduzido na Seção 2.5, a utilização dessas abordagens resultam em uma superfície de ataque maior, uma vez que a biblioteca de sistema operacional também é carregada dentro do enclave SGX. No Capítulo 5 fazemos a comparação da execução de uma instância de nossa solução utilizando Intel SGX SDK e SCONE.

Um processo não trivial é a atestação remota de enclaves, que é facilitado quando se utiliza abordagens como o Graphene-SGX, SGX-LKL e SCONE. Para Intel SGX SDK, o Squad [12] se apresenta como uma solução para o provisionamento de segredos e configurações para aplicações desenvolvidas utilizando o Intel SGX SDK. Nesta abordagem, o operador do sistema provisiona no Squad as configurações e segredos que devem ser entregues para as aplicações. As aplicações ao inicializarem, se comunicam com o Squad e são submetidas a um processo de atestação remota, onde é verificada a sua identidade e se ela é elegível para que os segredos sejam entregues. Uma parte da solução proposta no nosso trabalho utiliza uma abordagem análoga ao Squad, para proteger as credenciais de acesso e chaves criptográficas no contexto de sistemas publicar/assinar.

Silva [38] propõe um modelo de arquitetura de software com foco no aumento da segurança dos dados processados para aplicações publicar/assinar, em termos de confiabilidade e integridade, com a criação de domínios de privacidade, explorando recursos de ambientes de execução confiável. Porém, a sua proposta não atende a aplicações que já estão implantadas, em produção.

Capítulo 4

Solução Proposta

Para mitigar as ameaças descritas na Seção 3.2, cumprindo com os objetivos específicos deste trabalho, propomos um conjunto de componentes. A utilização destes componentes, em conjunto com as aplicações publicar/assinar existentes, proveem a proteção das credenciais bem como a confidencialidade e integridade dos dados, atendendo ao modelo de ameaças.

A Figura 4.1 apresenta uma visão geral da arquitetura, contendo os componentes propostos. O componente Sistema de Gerenciamento de Chaves é responsável por manter as configurações sensíveis, e é também responsável por autenticar e realizar a entrega dessas configurações. Esse componente é detalhado na Seção 4.1. Já o interceptador de mensagens é o componente responsável por filtrar as assinaturas e publicações realizadas pelos assinantes, bem como adicionar, de forma transparente, as características de confidencialidade e integridade às mensagens. Esse componente é detalhado na Seção 4.2.

4.1 Sistema de Gerenciamento de Chaves

O sistema de gerenciamento de chaves é a parte responsável por manter as configurações sensíveis das aplicações clientes do barramentos de dados. São consideradas configurações sensíveis as credenciais e chaves criptográficas utilizadas pelo processo de autenticação e criptografia das mensagens. O sistema é composto pelos componentes **catálogo** e **autenticador**, e utilizam uma base de dados compartilhada. A implementação utilizada para a base de dados é customizável, e está fora do escopo do sistema.

Por não ser um sistema crítico para o desempenho, uma vez que é usado apenas no

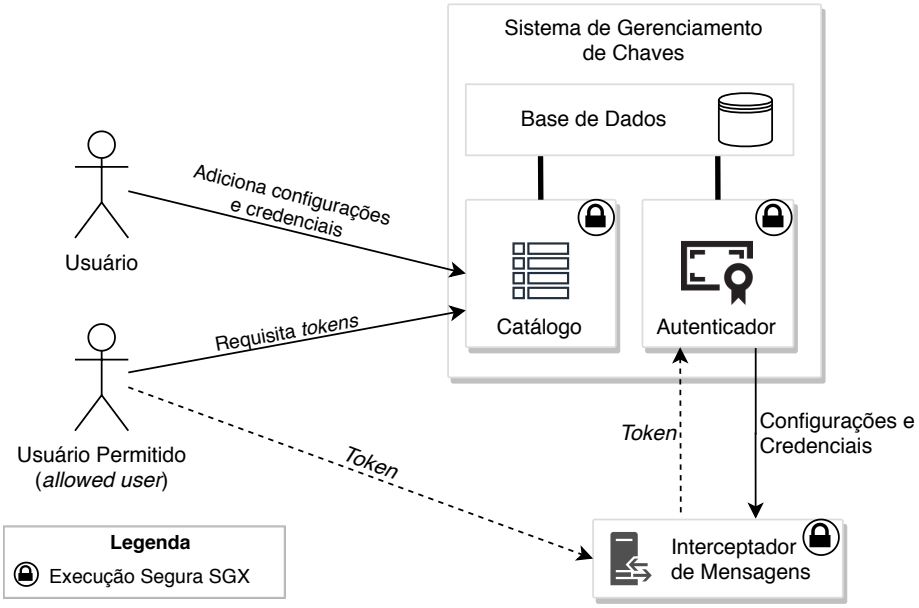

Figura 4.1: Arquitetura da Proposta

momento do registro e resgate de configurações, os seus componentes foram implementados utilizando a linguagem de programação Python, se beneficiando assim das características de uma linguagem de alto nível. Então, para se beneficiar das garantias de segurança em hardware providas pelo Intel SGX, a sua execução é feita através da plataforma SCONE.

4.1.1 Catálogo

O catálogo é o componente utilizado para o provisionamento de configurações e credenciais para acesso ao barramento de mensagens. É através dele que, também, são emitidos os *tokens* para que possam ser acessadas essas configurações pelas aplicações. Ele se apresenta como uma API REST e que possui em sua interface dois *endpoints*: `/catalog` e `/token`.

Acessível apenas por meio de uma conexão segura TLS, os usuários são identificados por meio de seu certificado de chave pública X.509, que devem ser assinados por uma autoridade certificadora (CA) reconhecida pelo sistema de gerenciamento de chaves. De forma simétrica ao que ocorre no Apache Kafka, as identificações dos usuários e aplicações são derivadas do campo **subject** do certificado. Por motivo de simplificação, derivamos os nomes de usuários e aplicações como sendo o conteúdo do *Common Name* (CN), parte do **subject**.

Através do *endpoint* `/catalog` são feitas operações sobre as entradas no catálogo. Uma

entrada contém configurações e metadados, e está associada a um tópico e a uma chave criptográfica escolhida aleatoriamente, permitindo um isolamento de diferentes aplicações ou domínio dos dados. No Código Fonte 4.1 é apresentado um exemplo de entrada.

```
{
  "data-description": "Smartmeter data",
  "topic-name": "sm.data",
  "allowed-users": [
    "sample-user", "another-user"
  ],
  "allowed-applications": [
    "aa25d6e1863819fca72f4f3315131ba4a438d1e1643c030190ca665215912465",
    "9c56db536e046a5fb84a5c482ce86e6592071dff75dc0e3eb27d701cf2c40508"
  ],
  "broker-credentials": {
    "cert-pem": "<base64 encoded PEM X.509 certificate>",
    "cert-pkey": "<base64 encoded certificate private key>",
    "cert-pkey-secret": "<private key secret - optional>"
  },
  "app-pubkeys": [
    {
      "name": "kafka-proxy-v1",
      "pubkey": "<base64 encoded PEM public key>",
      "name": "kafka-proxy-v2",
      "pubkey": "<base64 encoded PEM public key>"
    }
  ]
}
```

Código Fonte 4.1: Exemplo de uma entrada de catálogo

As entradas são definidas utilizando o formato JSON[1] (*JavaScript Object Notation*), que é um formato de fácil manutenção por desenvolvedores, e fácil interpretação e manipulação pelas aplicações. Uma entrada deve ter os seguintes campos:

- "`data-description`" – uma descrição alto nível da entrada, como por exemplo uma descrição do domínio dos dados o qual essa entrada se refere.

[1] https://www.json.org/json-en.html

- "`topic-name`" – o identificador do tópico no sistema publicar/assinar.
- "`allowed-users`" – lista de usuários que podem fazer solicitações de *tokens* para utilizar a entrada.
- "`allowed-applications`" – lista de MRENCLAVEs das aplicações que podem acessar as configurações e credenciais. Para facilitar o processo de desenvolvimento, quando as aplicações tem o seu MRENCLAVE constantemente modificados ou ainda não possuem suporte a atestação, essa lista pode conter uma entrada "$\star$", o que significa que qualquer aplicação SGX atestável remotamente pode acessar as configurações, ou uma lista vazia, onde neste caso o processo de entrega de configurações não utilizará atestação remota, porém, essas opções não devem ser consideradas em um ambiente de produção.
- "`broker-credentials`" – as credenciais para acessar o barramento de mensagens. A nossa abordagem se limita ao uso de certificados de chave pública X.509 para autenticação, logo este campo deve conter um certificado X.509 ("`cert-pem`"), sua chave privada ("`cert-pkey`"), e o segredo para descriptografar a chave privada ("`cert-pkey-secret`"), quando necessário. As chaves pública e privada devem estar no formato PEM, codificadas utilizando Base64.
- "`app-pubkeys`" – lista de chaves públicas das aplicações que irão se autenticar para receber as configurações. No exemplo apresentado no Código Fonte 4.1, a aplicação Kafka Proxy é listada.
- "`token-expiration-time`" – tempo, em minutos, de validade de um *token* emitida para acessar essa configuração. Esse campo é opcional, e quando não preenchido, é utilizado o valor padrão de 1440 minutos (1 dia).

Ao registrar uma nova entrada no catálogo, é devolvido como resposta um identificador único para a entrada criada e uma chave criptográfica simétrica de 128 bits (`"aes128-encryption-key"`), codificada em Base64, que deverá ser utilizada pelas aplicações para os processos que envolverem criptografia. Esta chave simétrica é guardada no armazenamento utilizado pelo catálogo criptografada utilizando criptografia assimétrica,

utilizando as chaves públicas das aplicações, logo apenas as aplicações que possuem as chaves privadas correspondentes são capazes de descriptografar o seu conteúdo. Um exemplo da resposta do serviço de catálogo pode ser vista no Código Fonte 4.2.

```
1 {
2   "aes128-encryption-key": "rATwOgjpqaYVWAZgfTVAdg==",
3   "entry-id": "6c974c76-b513-4a9c-8360-070ec1257f2d"
4 }
```

Código Fonte 4.2: Exemplo da resposta do serviço de catálogo após o registro de uma entrada

É de extrema importância que esta chave seja guardada pelo usuário que fez o registro da entrada no catálogo, pois ela será necessária para realizar qualquer atualização futura na entrada, bem como para recuperar as informações que venham a ser criptografadas utilizando a mesma como chave. Atualizações e a exclusão de uma entrada no catálogo devem ser feitas exclusivamente se autenticando utilizando credenciais que fizeram o registro.

Através do *endpoint* `/token` são feitas as emissões e revogações de *tokens* de acesso às configurações. Para fazer uma solicitação, é necessário que o usuário apresente um identificador de entrada e esteja listado no campo `"allowed-users"` da mesma. Um exemplo da resposta do serviço de solicitação de *tokens* pode ser vista no Código Fonte 4.3.

```
1 {
2   "expiration": "2020-02-06T18:42:54.683896",
3   "token": "0724c92d-d279-4aa3-b024-3d4f9a4fe777"
4 }
```

Código Fonte 4.3: Exemplo da resposta do serviço de solicitação de *token*

4.1.2 Autenticador

O autenticador é o componente responsável pela autenticação e entrega de configurações e credenciais para as aplicações. O autenticador utiliza um protocolo próprio, que deve ser implementado pelas aplicações que irão utilizá-lo. O fluxograma apresentado na Figura 4.2 representa o processo de autenticação. A gramática utilizada pelo protocolo de autenticação se encontra no Código Fonte C.1, no Apêndice C.

Utilizando comunicação segura por TLS, a aplicação deve apresentar um certificado de

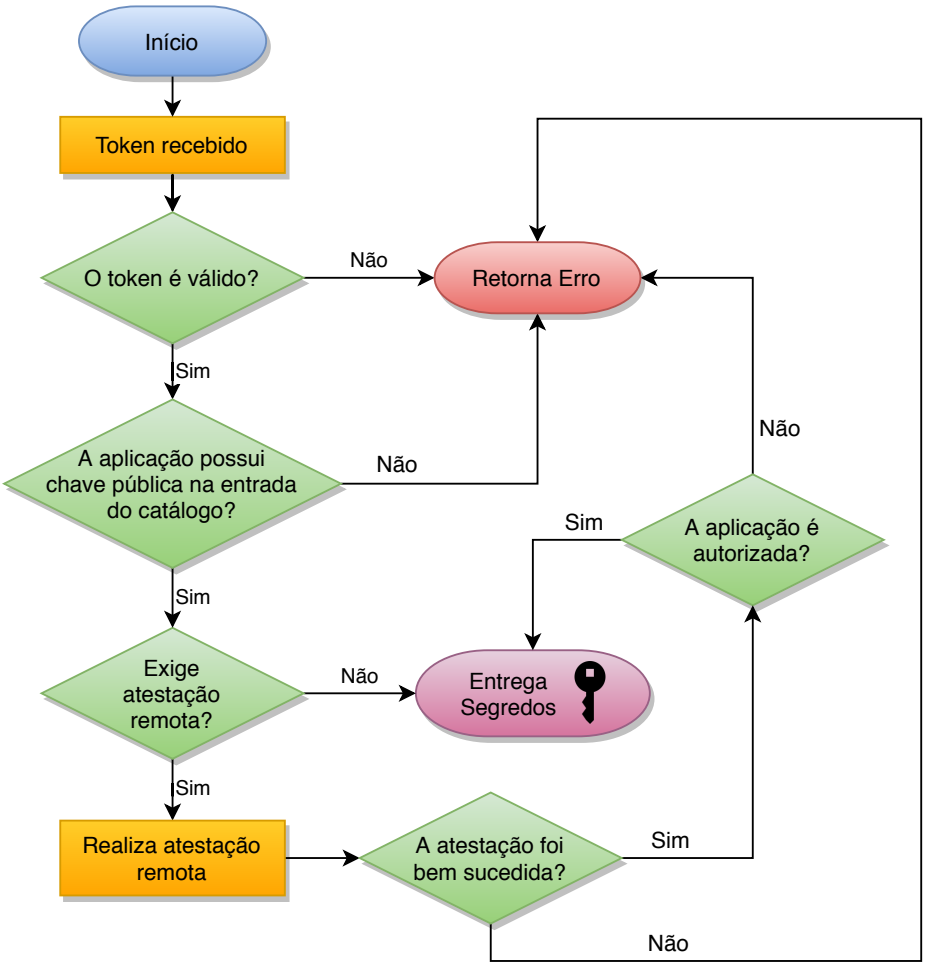

Figura 4.2: Fluxograma do protocolo de autenticação

chave pública assinada por uma CA reconhecida pelo sistema de gerenciamento de chaves, e um *token* válido, sendo este o primeiro fator utilizado pelo protocolo de autenticação. Um *token* é considerado válido se (i) ele consta na base de dados e (ii) se está dentro do período de validade. Sendo considerado válido, se avança para a próxima etapa que é verificar se a aplicação possui sua chave pública na entrada do catálogo à qual o *token* se refere.

Essa verificação é feita em cima do certificado de chave pública apresentado no *handshake* TLS, na abertura da conexão. Caso a chave pública conste na entrada do catálogo, é decidido se é necessário realizar atestação remota. Caso contrário, um erro é retornado.

Essa decisão, de haver ou não atestação remota, se baseia no conteúdo do campo "`allowed-applications`" da entrada do catálogo. Caso não seja necessária a atestação remota, os segredos são entregues nesse momento para a aplicação. Caso contrário, se inicia um processo de atestação remota, conforme descrito na Seção 2.4.2. É verificada a identidade da aplicação e caso a aplicação seja autorizada, ou seja, seu MRENCLAVE ou uma entrada "$\star$" constam no campo "`allowed-applications`", os segredos são entregues, criptografados utilizando a chave simétrica *SMK*, derivada no processo de atestação.

É importante ressaltar que a utilização da autenticação sem atestação remota ou com o campo "`allowed-applications`" preenchido com "$\star$" deve ser utilizada **apenas** em um ambiente de desenvolvimento, uma vez que aplicações maliciosas poderiam facilmente se passar por aplicações legítimas e obterem acesso as configurações e credenciais.

A Figura 4.3 representa a troca de mensagens que ocorre entre a aplicação e o autenticador, no processo de autenticação, quando há atestação remota.

O tempo gasto para a realização do processo de autenticação e entrega de configurações e credenciais é apresentado na Figura 4.4. Para a construção do gráfico foram consideradas a execução de 200 processos de autenticação, com e sem atestação remota. A Tabela 4.1 apresenta um sumário dos dados apresentados no gráfico.

Segurança	**Min.**	**1º Quartil**	**Mediana**	**Média**	**3º Quartil**	**Max.**
Sem atestação remota	62,11 ms	66,31 ms	68,67 ms	71,24 ms	73,79 ms	91,33 ms
Com atestação remota	912,3 ms	949,4 ms	1011,3 ms	1048,2 ms	1119,1 ms	1433,0 ms

Tabela 4.1: Sumário do tempo gasto para autenticação

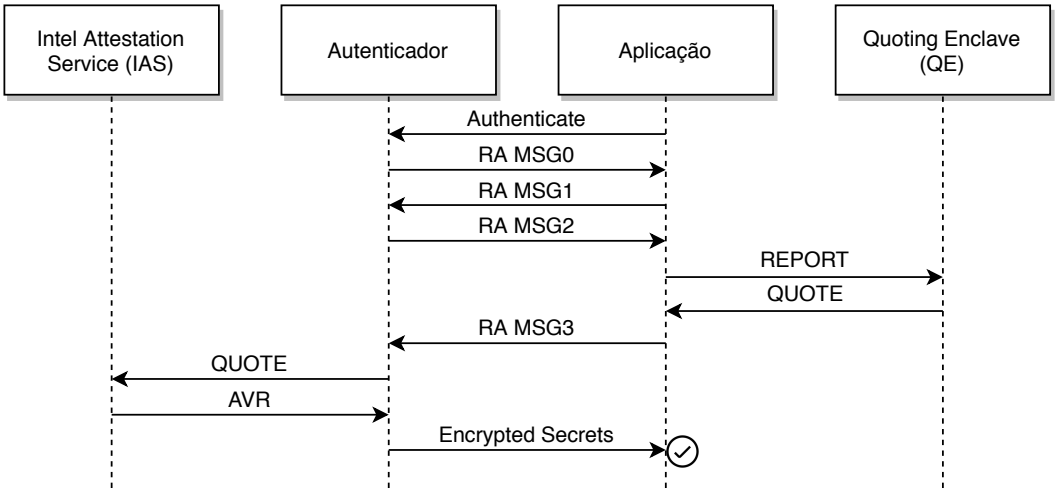

Figura 4.3: Autenticação com atestação remota

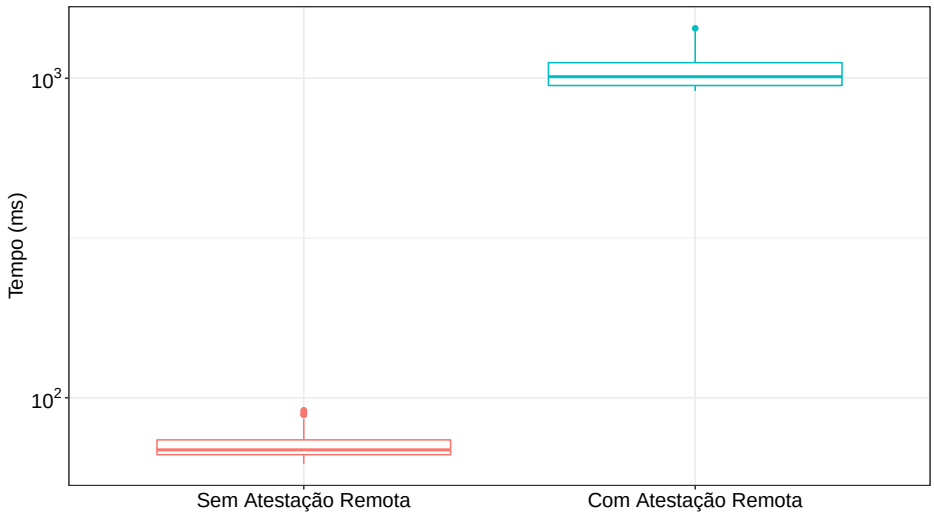

Figura 4.4: Gráfico do tempo gasto para autenticação

4.2 Interceptador de Mensagens

O interceptador de mensagens é o componente responsável por filtrar as assinaturas e publicações realizadas pelos assinantes, bem como adicionar, de forma transparente, as características de confidencialidade e integridade às mensagens. Este componente deve ser interposto entre o barramento de mensagens e os componentes publicadores e assinantes, conforme a Figura 4.5.

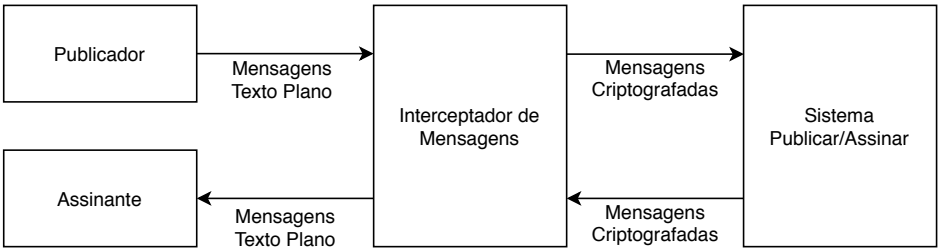

Figura 4.5: Configuração de publicadores e assinantes com interceptador de mensagens

As configurações de credenciais e chaves criptográficas que serão utilizadas devem ser obtidas através do processo de autenticação junto ao sistema de gerenciamento de chaves. É recomendável que o interceptador de mensagens seja executado dentro de um ambiente de execução confiável Intel SGX, uma vez que ele irá lidar com informações potencialmente confidenciais, além das provas de sua identidade, intrínsecas da tecnologia, que ele é capaz de fornecer.

O interceptador de mensagens é o responsável pela autenticação no barramento de mensagens, através do uso de certificados de chave pública X.509, conforme as considerações expostas na Seção 3.2. Desta forma, a conexão entre o interceptador de mensagens e o barramento de mensagens é assegurada por TLS, adicionando garantias de integridade e confidencialidade ao tráfego de mensagens.

Caso o interceptador de mensagens seja executado em um ambiente de execução externo ao da aplicação publicadora ou assinante, é mandatório o uso do TLS também na sua conexão com a aplicação cliente, uma vez que irá existir tráfego de informações externo ao ambiente de execução.

Ainda que a carga útil da mensagem esteja criptografada, caso não seja utilizado TLS, considerando um adversário capaz de inspecionar o tráfego, é possível obter detalhes sobre os dados que estão sendo transferidos, revelando, por exemplo, a estrutura de tópicos utilizada

pelas aplicações, uma vez que a estrutura do pacote é mantida, de acordo com o barramento de mensagens utilizado.

A filtragem de publicações e assinaturas restringe aos publicadores e assinantes o acesso ao tópico definido na entrada do catálogo, no campo "`topic-name`". As mensagens publicadas e assinaturas realizadas para tópicos diferentes do que foi definido na entrada são descartadas. Desta forma, conseguimos restringir o acesso dos publicadores e assinantes ao seus domínios de mensagens.

As características de integridade e confidencialidade são obtidas através do uso da criptografia simétrica AES-GCM[2] (*Galois/Counter Mode*), que além de criptografar, possui suporte a autenticação das mensagens, através do uso de MAC (*Message Authentication Code*). Desta forma, todas as mensagens que atravessam o interceptador de mensagem, dependendo da direção, são criptografadas ou descriptografadas, de forma transparente. Como consequência, as mensagens que irão trafegar através do barramento de mensagens estarão criptografadas e passíveis de verificação de integridade.

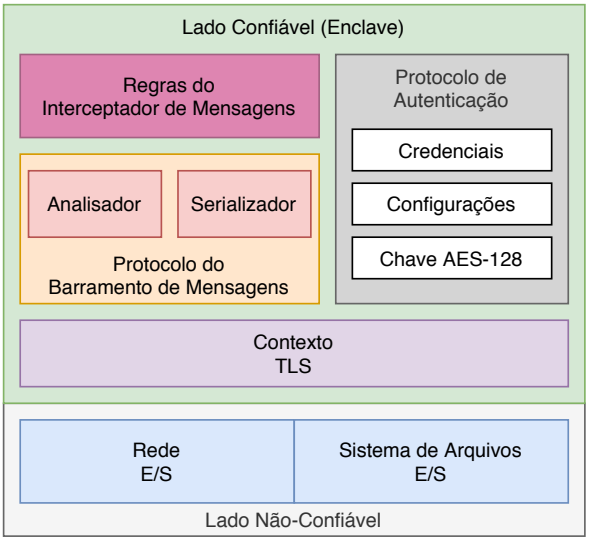

Figura 4.6: Particionamento dos blocos de montagem do interceptador de mensagens

O particionamento do interceptador de mensagens, conforme a Figura 4.6, é necessário para que as operações que exigem características de confidencialidade e integridade do processamento sejam feitas na região isolada da memória e processador. Implementando desta forma, em nenhum momento são expostas as credenciais, chaves criptográficas e configu-

[2]https://csrc.nist.gov/publications/detail/sp/800-38d/final

rações recebidas no processo de autenticação. A implementação específica para diferentes sistemas publicar/assinar deve ser feita no bloco do protocolo do barramento de mensagens, bem como as regras do interceptador de mensagens. Uma implementação específica para Apache Kafka foi desenvolvida para ser utilizada na aplicação de exemplo LiteMe.

4.3 Integração com a Aplicação de Exemplo

Para integrar a proposta com a aplicação de exemplo, foi desenvolvido um interceptador de mensagens específico para o Apache Kafka, o qual chamamos de Kafka Proxy. Para implementar o analisador e serializador do protocolo do Kafka, foi utilizado um compilador desenvolvido para esta finalidade, apresentado no Apêndice D. A Figura 4.7 apresenta a arquitetura do LiteMe com a integração dos componentes propostos.

Inicialmente foram registradas 4 entradas no catálogo, uma para cada domínio de mensagens: dados dos *smart meters*, alertas de anomalias, desagregador NILM e historiador de consumo. Neste momento, as credenciais de acesso ao *broker* são retiradas dos componentes, para serem exclusivamente acessadas através do autenticador.

Nos ambientes em que executam os componentes consumidores e produtores, foram inicializadas instâncias do Kafka Proxy, sinalizados na Figura 4.7 pela sigla "KP". Apesar da atomicidade dos componentes, quando há a utilização de mais de um domínio de mensagens diferente, é necessário instanciar um Kafka Proxy para cada domínio. Cada instância do Kafka Proxy deve ser configurada para se conectar com o *broker*, e as conexões dos componentes da aplicação devem ser realizadas através do endereço de escuta do Kafka Proxy.

Por serem executados nos mesmos ambientes que os componentes da aplicação, as instâncias do Kafka Proxy foram configuradas para escutarem conexões inseguras (sem TLS), uma vez que não há tráfego externo ao ambiente de execução. Apenas a conexão com o *broker*, com o uso das credenciais recebidas através do processo de autenticação, é assegurada com TLS. Essa decisão foi tomada para reduzir a sobrecarga adicionada pela segurança.

Para cada instância do Kafka Proxy, foi solicitado um *token* para ser utilizado por ele. É feito dessa forma pois, caso seja detectado qualquer comportamento anormal por parte de algum dos componentes da plataforma, é possível revogar o seu acesso isoladamente, sem afetar outras partes do sistema.

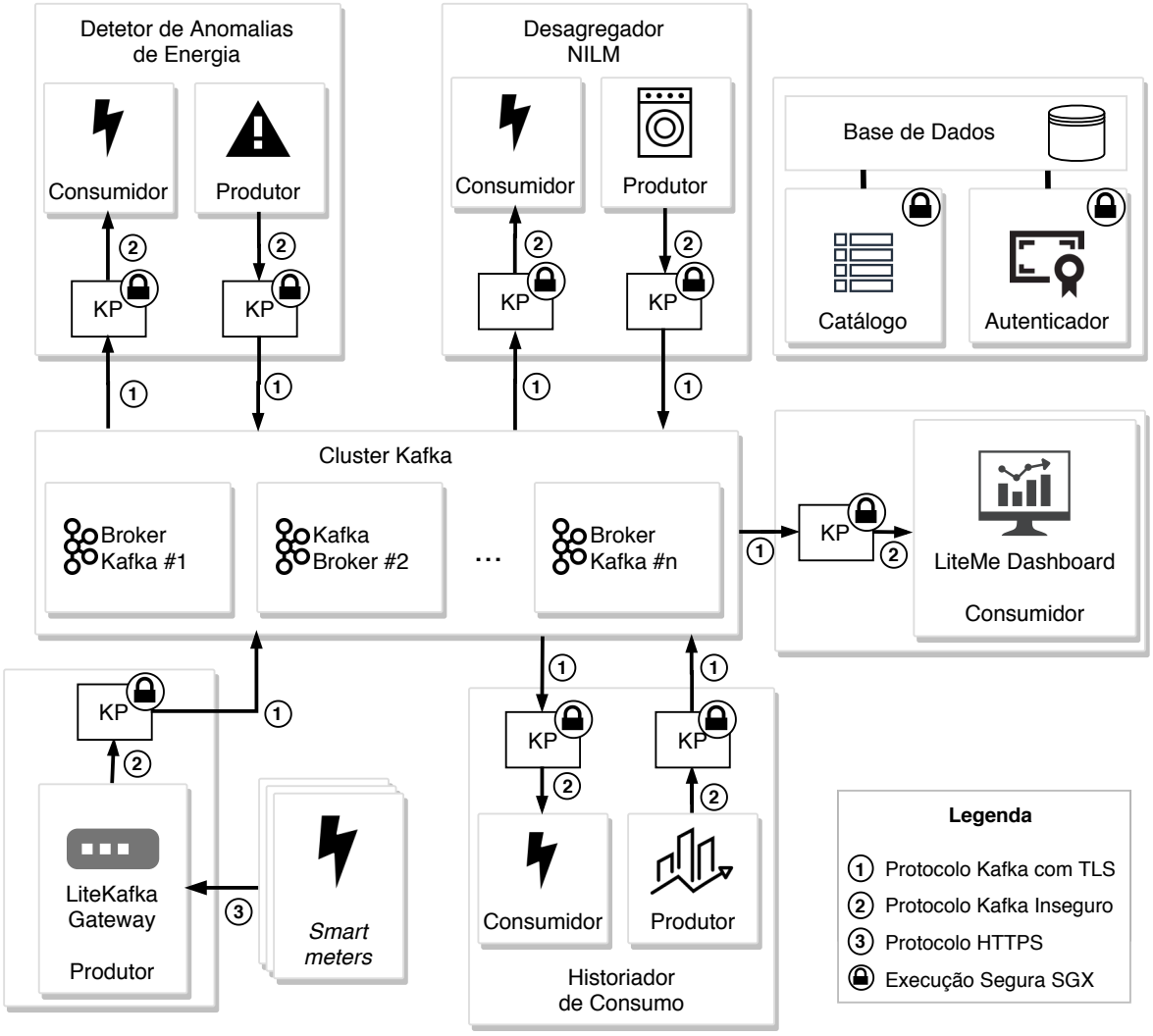

Figura 4.7: Integração dos novos componentes com a arquitetura do LiteMe

Desta forma, sem modificações na aplicação, estendemos as funcionalidades do sistema publicar/assinar, acrescentando propriedades de confidencialidade e integridade nas informações.

Capítulo 5

Avaliação de Desempenho

Neste capítulo é apresentada a avaliação de desempenho da solução proposta nesta dissertação. Na Seção 5.1 é feita uma descrição do experimento realizado, na Seção 5.2 é dada uma visão geral das ferramentas para avaliação de desempenho que foram utilizadas, na Seção 5.3 é feita uma descrição do ambiente utilizado, a Seção 5.4 descreve como foram conduzidos os experimentos, na Seção 5.6 são apresentados e discutidos os resultados obtidos, e na Seção 5.7 são listadas as ameaças de validade.

5.1 O Experimento

O experimento realizado tem como objetivo avaliar o custo de desempenho e de recursos pela utilização do **Kafka Proxy**. Este componente é executado durante todo o ciclo de vida da aplicação, em conjunto com os produtores e consumidores de mensagens. As métricas de vazão de produção e de consumo de mensagens e a latência são utilizadas para avaliar o custo. O custo está relacionado com a razão entre a métrica observada sem e com a utilização do Kafka Proxy.

Em linhas gerais, o experimento visa responder às questões de pesquisa P1 e P2:

- P1 – Qual o custo em termos de desempenho (latência e vazão) para as diferentes configurações?
- P2 – Qual o custo em termos de recursos adicionais de CPU para as diferentes configurações?

Um *design* fatorial completo foi escolhido para guiar o experimento. Foram considerados os fatores (variáveis independentes), a configuração da aplicação e o tamanho da mensagem. A configuração da aplicação está relacionada com a utilização ou não do Kafka Proxy e, quando utilizado, à forma com que a aplicação foi construída, onde:

- Proxy Inseguro - a aplicação é compilada ligada à biblioteca GNU C Library[1], sem oferecer qualquer tipo de garantia de segurança em hardware;
- SGX SDK - a aplicação é compilada ligada às bibliotecas do Intel SGX SDK, e é executada no ambiente de tempo de execução provido pelo mesmo;
- SCONE - a aplicação é compilada utilizando o compilador `scone-g++` e ligada à biblioteca SCONE `musl libc`, e é executada no ambiente de tempo de execução SCONE.

A Tabela 5.1 apresenta todos os níveis escolhidos para os fatores do experimento.

Fator	**Níveis**
Configuração da aplicação	Sem Proxy, Proxy Inseguro, SGX SDK, SCONE
Tamanho da mensagem (bytes)	128, 256, 512, 1024, 2048, 4096, 8192

Tabela 5.1: Definição dos níveis dos fatores do experimento.

5.2 Ferramentas Utilizadas

Para avaliar o desempenho, foram utilizadas ferramentas distribuídas em conjunto com o Apache Kafka: `kafka-producer-perf-test`, `kafka-consumer-perf-test` e `EndToEndLatency`. Estas mesmas ferramentas foram utilizadas em avaliações similares na literatura [8, 30, 43, 44].

A ferramenta `kafka-producer-perf-test` simula um produtor Kafka, e é utilizada para medir a vazão de produção de mensagens. Para a execução dessa ferramenta, é necessário definir o endereço de um dos *brokers* do *cluster* Kafka, o nome tópico para o qual

[1] https://www.gnu.org/software/libc/

serão feitas as publicações, a quantidade de mensagens a serem produzidas, o tamanho de cada uma e a taxa de produção por segundo.

A ferramenta `kafka-consumer-perf-test` simula um consumidor Kafka, e é utilizada para medir a vazão de ingestão de mensagens. Para a execução dessa ferramenta, é necessário definir o endereço de um ou mais *brokers* do *cluster* Kafka, o nome do tópico o qual as mensagens serão consumidas e a quantidade de mensagens a serem consumidas.

A ferramenta `EndToEndLatency` simula um produtor e consumidor Kafka simultaneamente, e é utilizada para medir a latência ponto-a-ponto, ou seja, o tempo gasto para que uma mensagem seja publicada e, posteriormente, entregue pelo *broker* Kafka. Para a execução dessa ferramenta, é necessário definir o endereço de um ou mais *brokers* do *cluster* Kafka, o nome tópico o qual as mensagens serão publicadas e consumidas, e a quantidade de mensagens a serem publicadas e consumidas.

Para a coleta de informações referente ao consumo de recursos de CPU foi utilizada a ferramenta `pidstat`. Parte do pacote `sysstat` [15], esta ferramenta processa as estatísticas por processo disponibilizadas pelo *Kernel GNU/Linux*, identificadas pelo número identificador do processo (PID, ou do inglês *Process Identification Number*) disponibilizadas no caminho '/proc/**PID #**/stat', onde 'PID #' é o número identificador do processo.

Os comandos utilizados para a avaliação de performance encontram-se no Apêndice E.

5.3 Ambiente Utilizado

Para este experimento foram utilizadas cinco máquinas virtuais (VMs) instanciadas em um ambiente de computação na nuvem privada, onde foram implantados e executados os componentes necessários para a realização desta avaliação. A Tabela 5.2 apresenta, em detalhes, as configurações das VMs utilizadas no experimento.

Nome	**Quantidade**	**Configurações**
lsd.t2.large	3	4 vCPUs, 8GB RAM
sgx.epc16.medium	1	2 vCPUs, 4GB RAM, SGX (16MB EPC)
sgx.epc32.medium	1	2 vCPUs, 4GB RAM, SGX (32MB EPC)

Tabela 5.2: Configurações de VMs

Foram utilizadas três VMs do tipo `lsd.t2.large` para a implantação do *cluster* do Apache Kafka, para os componentes Catálogo e Autenticador foi utilizada uma VM do tipo `sgx.epc16.medium`, e para as ferramentas utilizadas para avaliação de desempenho e o Kafka Proxy foi utilizada uma VM do tipo `sgx.epc32.medium`. No *cluster* Kafka foi criado um tópico chamado 'proxy-test' com fator de replicação igual a 3, e 9 a quantidade de partições, para ser utilizado nas avaliações. O fator de replicação foi definido igual à quantidade de nós para garantir que o tópico seja replicado por todos os *brokers* do cluster, e a quantidade de partições múltipla, e maior que o fator de replicação, para reforçar a distribuição das mensagens ao longo dos *brokers*. A distribuição de partições e suas respectivas lideranças estão detalhadas na Figura 5.1.

A configuração de um ambiente de nuvem como o OpenStack para o uso de Intel SGX exige configurações adicionais, esses passos foram resultado do trabalho descrito em [36].

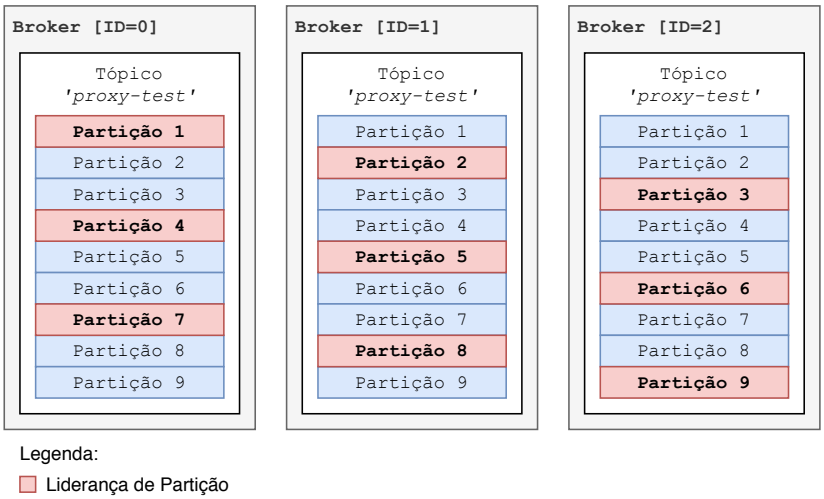

Figura 5.1: Configuração do tópico 'proxy-test' utilizado no experimento

5.4 Metodologia

A avaliação de desempenho de produção e consumo de mensagens foi realizada utilizando as ferramentas `kafka-producer-perf-test` e `kafka-consumer-perf-test`, e a avaliação de latência ponto-a-ponto foi realizada utilizando a ferramenta `EndToEndLatency`. Nos tratamenos onde a configuração da aplicação possui o nível

"Sem Proxy", as ferramentas se conectam diretamente ao *cluster* Kafka, enquanto nos outros níveis o Kafka Proxy é executado em conjunto com as ferramentas, na mesma máquina.

A escolha dos níveis para tamanho de mensagem foi feita a partir de uma avaliação prévia do *cluster* utilizado no experimento, para identificar o comportamento da vazão de dados para diferentes tamanhos de mensagem, apresentada na Seção 5.5.

Antes de começar a execução do experimento, foi armazenado o estado de todo o *cluster*. Após a execução de cada réplica, esse estado inicial é recarregado, com o objetivo de evitar que alguma mudança no estado do *cluster* influencie nos resultados. Para cada tratamento foram executadas 30 réplicas do experimento. Nos experimentos de produção e consumo de mensagens foram produzidas e consumidas um total de 2.500.000 mensagens por réplica, e para latência foram enviadas 200 mensagens em cada réplica do experimento, com uma taxa de 1 mensagem por segundo.

5.5 Avaliação do Cluster

Foi conduzida uma avaliação de desempenho do *cluster* utilizado no experimento, com o objetivo de determinar os níveis de tamanhos de mensagens que serão utilizados. Esta avaliação consistiu em executar as ferramentas `kafka-producer-perf-test` e `kafka-consumer-perf-test`, para diferentes tamanhos de mensagem, observando a vazão do *cluster* em termos de $bytes/s$.

Foi determinado como tamanho de mensagem inicial igual à 128 bytes, e este valor foi sendo duplicado, enquanto a vazão de dados do *cluster* aumentava. Para cada tamanho de mensagem foram executadas 30 réplicas. Na Figura 5.2 é observado o comportamento da vazão do *cluster* com a variação dos tamanhos de mensagens.

Ao chegar no tamanho de mensagem 2048 bytes, é possível observar que é atingido o ponto ótimo na capacidade de ingestão e entrega de dados do *cluster*. A partir desse ponto, começa a ocorrer a degradação do desempenho. Então, foram escolhidos como níveis de tamanhos de mensagens, em bytes, para o experimento: 128, 256, 512, 1024, 2048, 4096 e 8192.

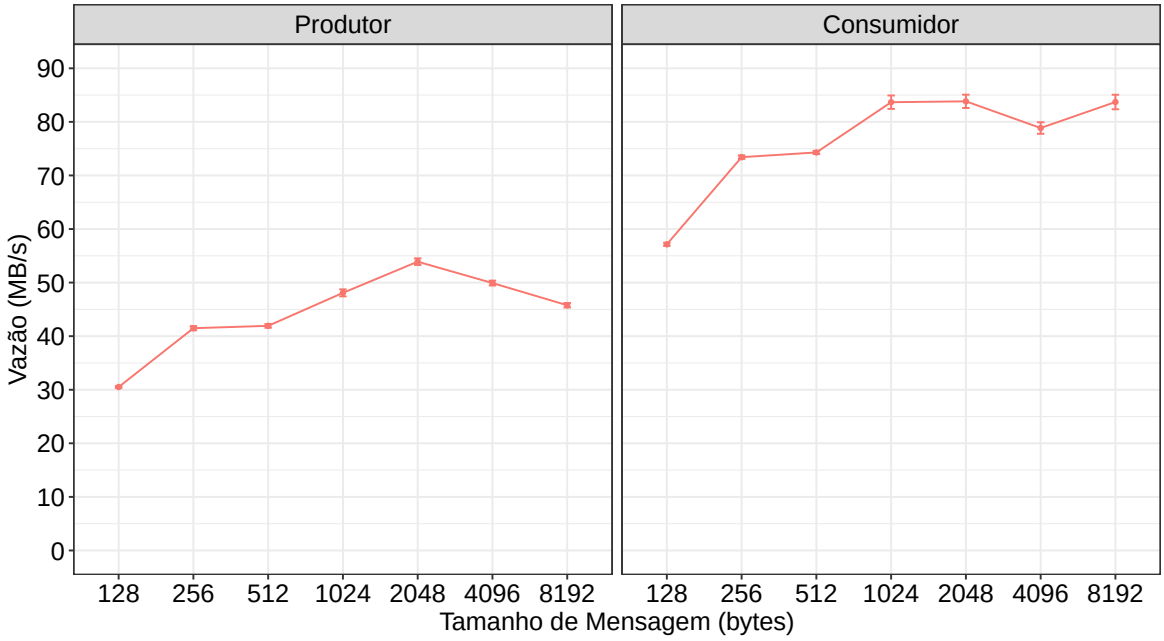

Figura 5.2: Gráfico de vazão (MB/s) do *cluster* por tamanho de mensagem

5.6 Discussão

Na Figura 5.3 é possível observar que a utilização do Kafka Proxy, independente de sua configuração, gera uma sobrecarga, tanto na produção quanto no consumo de mensagens. A utilização do Kafka Proxy sem SGX reduz a vazão de produção de mensagens 42,89% em média, considerando os diferentes tamanhos de mensagens. Já para a vazão de consumo essa redução é menor, para 24,83%, em média.

Ao executar o Kafka Proxy utilizando SGX, a configuração que utiliza SGX SDK possui uma redução de vazão menor do que a que utiliza SCONE. Utilizando SGX SDK, a redução de vazão é, em média, de 50,96% para produção e 48,62% para consumo. Já utilizando SCONE, a redução de vazão passa a ser, em média, de 79,56% para produção e 70,83% para consumo. A Tabela 5.3 apresenta a redução percentual da vazão por tamanho de mensagem e configuração.

A Figura 5.4 apresenta um gráfico de vazão, desta vez utilizando a métrica ‘megabytes por segundo’ no eixo das ordenadas. Nele é possível observar que, utilizando o tamanho de mensagem de 2048 bytes, ocorre o melhor rendimento dentre os tamanhos de mensagens escolhidos. Ou seja, maximiza o uso do canal de comunicação. Este comportamento é

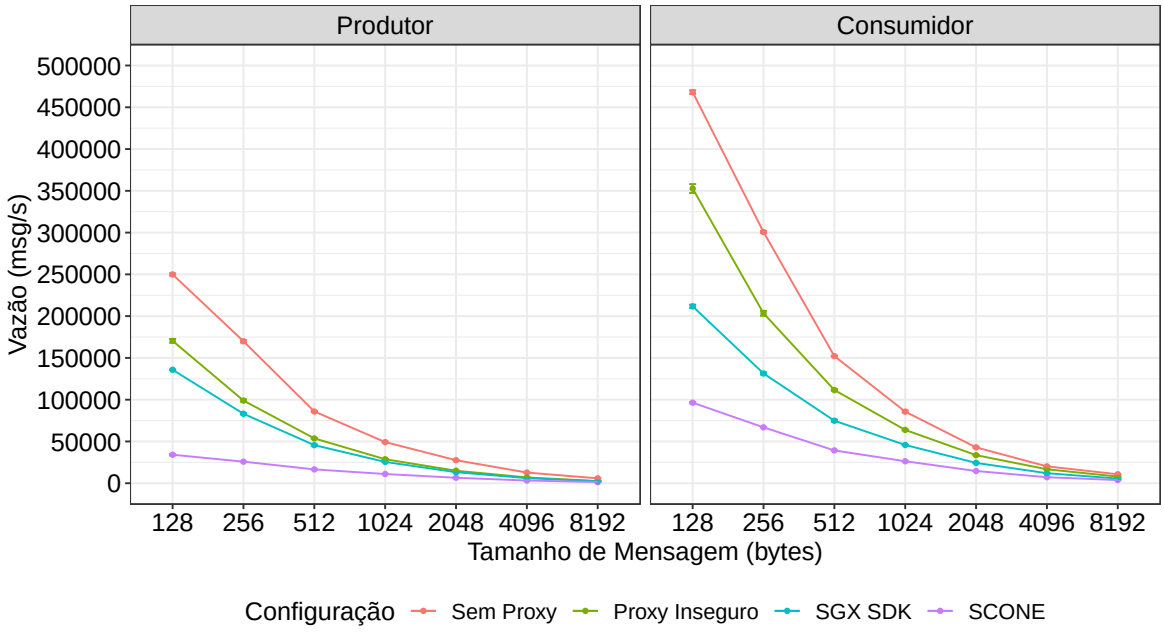

Figura 5.3: Gráfico de vazão (msg/s) por tamanho de mensagem

Tam. de Mensagem	**Produtor**			**Consumidor**		
	Inseguro	**SGX SDK**	**SCONE**	**Inseguro**	**SGX SDK**	**SCONE**
128	31,82%	45,67%	86,35%	24,65%	54,77%	79,40%
256	41,76%	51,08%	84,81%	32,35%	56,31%	77,76%
512	37,51%	46,89%	80,71%	26,68%	50,83%	74,19%
1024	41,66%	48,22%	77,63%	25,53%	46,52%	69,30%
2048	45,73%	52,07%	76,32%	21,71%	43,33%	65,95%
4096	45,74%	52,34%	74,65%	17,31%	40,67%	64,41%
8192	55,98%	60,41%	76,46%	25,59%	47,90%	64,79%

Tabela 5.3: Redução percentual da vazão com o uso do proxy

observado com presença e ausência de proxy, em todas as configurações.

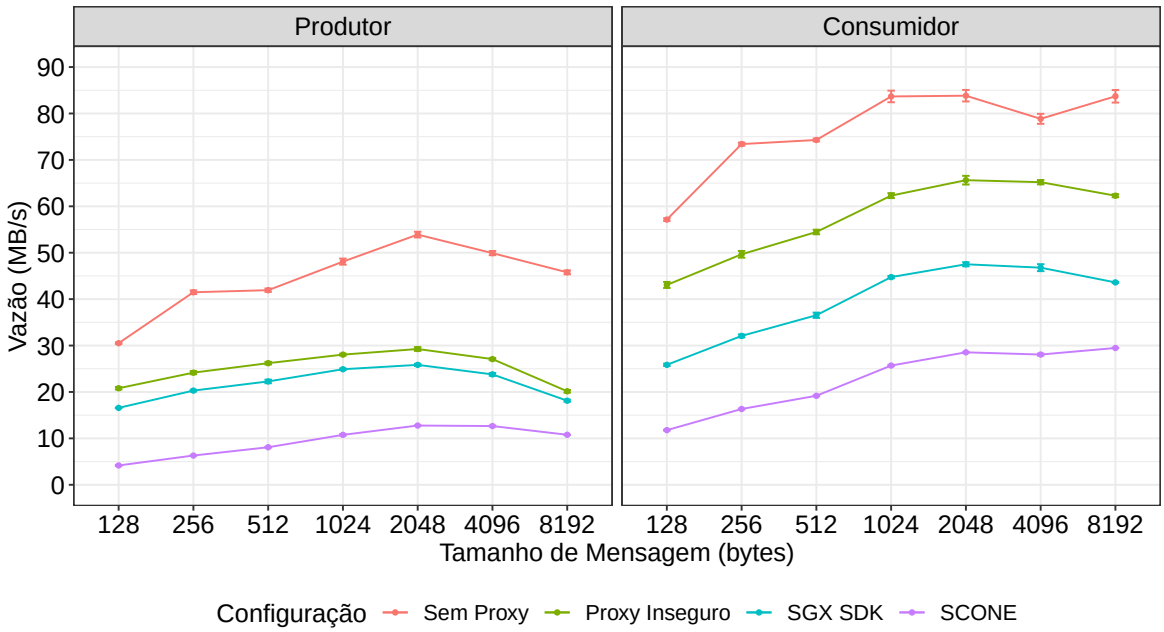

Figura 5.4: Gráfico de vazão (MB/s) por tamanho de mensagem

Portanto, em termos de capacidade, o Kafka Proxy é uma alternativa viável para o uso no contexto da aplicação de exemplo, o LiteMe, comportando aproximadamente 25.000 medidores por instância, considerando o tamanho de mensagem de 1024 bytes. Para aumentar esta vazão, é possível a utilização de estratégias com múltiplas instâncias do proxy e balanceadores de carga[2].

Na Figura 5.5 temos o perfil do tempo de CPU do proxy ao longo da execução, para o tamanho de mensagem 2048 bytes. Foi escolhido esse tamanho pois, conforme foi constatado anteriormente, é onde ocorre o melhor rendimento. É mostrado apenas os 25 segundos iniciais da execução pois é o tempo necessário para que ocorra a estabilização do tempo de CPU. Constata-se que as configurações que utilizam SGX, apesar de ter um desempenho inferior, possuem um tempo de CPU maior.

A Figura 5.6 apresenta a latência ponto-a-ponto, por tamanho de mensagem. A latência

[2]É importante lembrar também que grandes instalações provavelmente terão diferentes tópicos que agrupam diferentes domínios de mensagens, assim cada domínio já seria naturalmente mapeado para uma instância do Kafka Proxy.

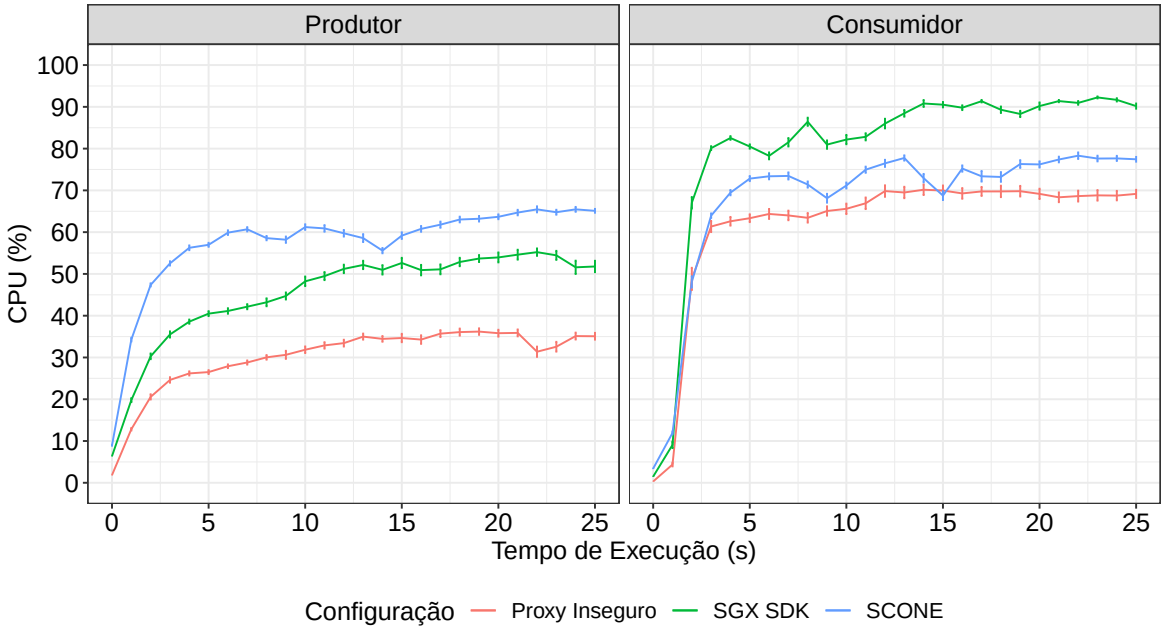

Figura 5.5: Gráfico de tempo de CPU (%) no tempo (s)

adicionada pelo uso do Kafka Proxy é de aproximadamente $1\ ms$ quando utilizado sem SGX, $2\ ms$ utilizando SGX SDK, e $16{,}55\ ms$ utilizando SCONE. É possível observar que a variação do tamanho da mensagem, nos níveis avaliados, não influencia na latência ponto-a-ponto. A Tabela 5.4 apresenta as latências médias para cada tratamento do experimento.

Tam. de Mensagem	**Configuração**			
	Sem Proxy	**Inseguro**	**SGX SDK**	**SCONE**
128	13,53 ms	14,54 ms	15,95 ms	27,33 ms
256	11,35 ms	12,01 ms	12,76 ms	26,24 ms
512	11,38 ms	11,79 ms	13,11 ms	29,17 ms
1024	10,53 ms	11,73 ms	12,77 ms	28,80 ms
2048	10,48 ms	11,45 ms	12,68 ms	26,84 ms
4096	10,27 ms	11,62 ms	12,41 ms	30,74 ms
8192	11,83 ms	13,32 ms	13,66 ms	26,11 ms

Tabela 5.4: Latência média, em diferentes configurações

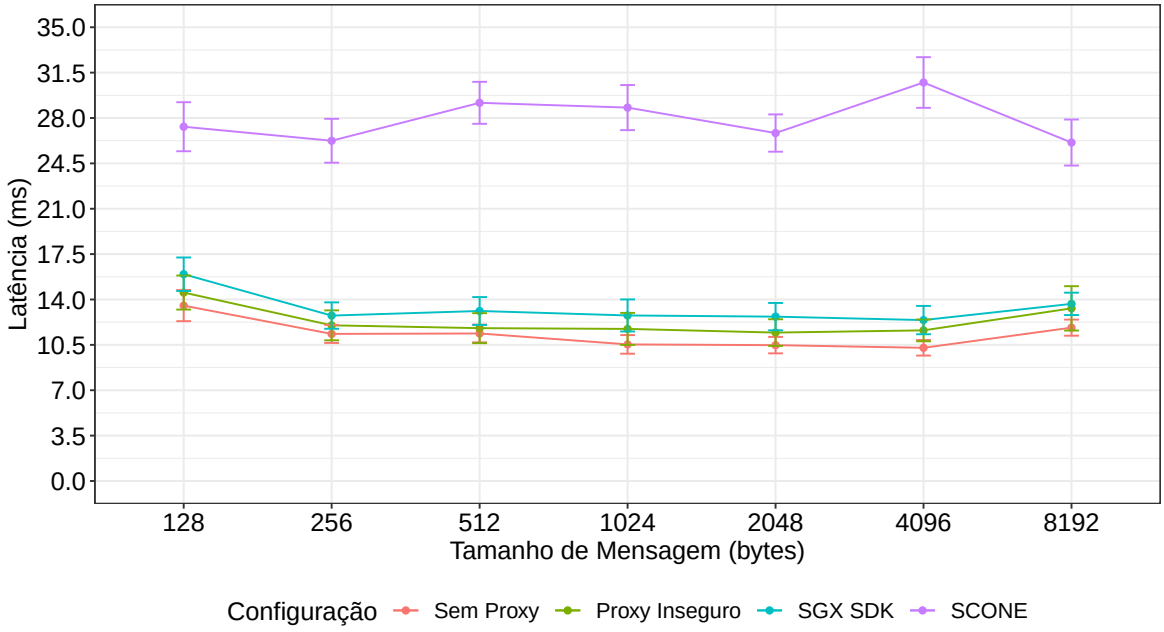

Figura 5.6: Latência (ms) ponto-a-ponto por tamanho de mensagem (bytes)

5.7 Ameaças à Validade

As seguintes ameaças à validade foram identificadas:

- **Validade de constructo**: o experimento realizado não considera fatores importantes em sistemas distribuídos, como por exemplo: (i) capacidade de comunicação (rede), (ii) desempenho do armazenamento.

- **Validade de constructo**: o experimento realizado não considera outras configurações do SCONE, como suas configurações internas para execução de chamadas do sistema, utilizando configuração padrão.

Capítulo 6

Conclusão

Neste capítulo, na Seção 6.1 é apresentado um sumário do que foi proposto e realizado nesta pesquisa e algumas considerações sobre as análises de desempenho feitas, e na Seção 6.2 são apresentados possíveis trabalhos futuros.

6.1 Considerações finais

Neste trabalho foi apresentada uma abordagem para adicionar garantias de confidencialidade e integridade a sistemas publicar/assinar. Foi proposto um conjunto de componentes para se integrarem a aplicações existentes. Estes componentes exploram o uso da tecnologia de segurança em hardware Intel SGX, para prover garantias de confidencialidade e integridade aos dados, além da proteção de credenciais. O sistema estende as capacidades de um barramento de mensagens publicar/assinar sem alterar a sua interface de comunicação, o que se apresenta como uma solução bastante atraente, uma vez que não são necessárias mudanças nas implementações existentes. Acreditamos que o LiteMe é uma aplicação que reflete vários dos desafios das aplicações de IoT e Smart Cities.

Os experimentos mostram que há um custo médio de aproximadamente $49{,}79\%$ da vazão e um aumento da latência de $2\ ms$, em média, com a utilização de nossa abordagem em uma implementação específica para o Apache Kafka. A nossa solução se apresenta viável para a utilização em conjunto com a aplicação de exemplo LiteMe, considerando as características dos dados utilizados por este sistema.

Além disso, como consequência direta ou indireta deste trabalho, artigos foram publica-

dos: "*Federated and secure cloud services for building medical image classifiers on an intercontinental infrastructure*" publicado no *Journal: Future Generation Computer Systems*, e o "*Secure and Privacy-Aware Data Dissemination for Cloud-Based Applications*", publicado no "*Proceedings of the 10th International Conference on Utility and Cloud Computing*".

6.2 Trabalhos Futuros

Entre as diversas linhas que podem ser seguidas, algumas são mais promissoras. Por exemplo, a integração das aplicações de catálogo e autenticação com um sistema de armazenamento robusto, descentralizado e imutável (como uma *blockchain*) adicionaria a possibilidade de auditoria confiável, uma vez que registraria os acessos e modificações de forma robusta. Desta forma, os proprietários dos dados teriam maior controle sobre a utilização dos mesmos.

Outro caminho seria adicionar suporte a publicadores e assinantes atestáveis remotamente. Desta maneira, seria possível verificar a identidade e a procedência do código que de fato processa as informações, podendo-se limitar a priori como e quais operações estão sendo executadas sobre os dados.

Finalmente, uma implantação em um ambiente mais realista, com centenas de sensores, seria desejável. Isso permitiria uma melhor avaliação da robustez e do custo de gerência e processamento de dados e segredos.

Bibliografia

[1] A. O. Adebayo. “A foundation for breach data analysis”. Em: *Journal of Information Engineering and Applications* 2.4 (2012), pp. 17–23.

[2] A. V. Aho, M. S. Lam, R. Sethi e J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[3] Apache Software Foundation. *Apache Kafka: Documentation*. Acesso em: 29/12/2019. URL: `https://kafka.apache.org/23/documentation`.

[4] Apache Software Foundation. *Kafka Protocol Guide*. Acesso em: 14/02/2020. URL: `https://kafka.apache.org/23/protocol`.

[5] ARM, Architecure. “Security technology building a secure system using TrustZone technology”. Em: *ARM Technical White Paper* (2009).

[6] S. Arnautov, A. Brito, P. Felber, C. Fetzer, F. Gregor, R. Krahn, W. Ozga, A. Martin, V. Schiavoni, F. Silva et al. “PubSub-SGX: Exploiting Trusted Execution Environments for Privacy-Preserving Publish/Subscribe Systems”. Em: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2018, pp. 123–132.

[7] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O’keeffe, M. L. Stillwell et al. “SCONE: Secure Linux Containers with Intel SGX”. Em: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 689–703.

[8] M. Bacou, G. Todeschi, D. Hagimont e A. Tchana. “Nested Virtualization Without the Nest”. Em: *Proceedings of the 48th International Conference on Parallel Processing*. 2019, pp. 1–10.

[9] L. Columbus. *83% Of Enterprise Workloads Will Be In The Cloud By 2020*. Acesso em: 20/02/2020. URL: https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/.

[10] V. Costan e S. Devadas. "Intel SGX Explained." Em: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[11] G. Coulouris, J. Dollimore, T. Kindberg e G. Blair. *Sistemas Distribuídos: Conceitos e Projeto*. 5ª ed. 2013, 2013.

[12] M. S. L. da Silva, F. F. de Oliveira Silva e A. Brito. "Squad: A Secure, Simple Storage Service for SGX-Based Microservices". Em: *2019 9th Latin-American Symposium on Dependable Computing (LADC)*. Nov. de 2019, pp. 1–9.

[13] P. T. Eugster, P. A. Felber, R. Guerraoui e A.-M. Kermarrec. "The many faces of publish/subscribe". Em: *ACM computing surveys (CSUR)* 35.2 (2003), pp. 114–131.

[14] P. P. Gero Mühl Ludger Fiege. *Distributed Event-Based Systems*. 1ª ed. Springer-Verlag, 2006.

[15] S. Godard. *System Statistics (sysstat)*. URL: http://sebastien.godard.pagesperso-orange.%20fr/.

[16] D. Gutierrez. *A Brief History of Kafka, LinkedIn's Messaging Platform - insideBIGDATA*. Acesso em: 17/01/2020. URL: https://insidebigdata.com/2016/04/28/a-brief-history-of-kafka-linkedins-messaging-platform/.

[17] M. Henshaw. "Systems of Systems, Cyber-Physical Systems, The Internet-of-Things... Whatever Next?" Em: *Insight* 19.3 (2016), pp. 51–54.

[18] R. Housley, T. Polk, D. W. S. Ford e D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3280. Maio de 2002.

[19] Intel Corporation. *Intel Software Guard Extensions*. Acesso em: 17/01/2020. URL: https://software.intel.com/en-us/sgx.

[20] Intel Corporation. *Intel Xeon E-2200 Processors for Servers*. Acesso em: 07/01/2020. URL: https://www.intel.com/content/www/us/en/products/docs/processors/xeon/xeon-e-2200-server-processor-brief.html (acesso em).

[21] Intel Software. *SDK for Intel Software Guard Extensions*. Acesso em: 23/02/2020. URL: https://software.intel.com/en-us/sgx/sdk (acesso em).

[22] ITU-T. *Recommendation x.509: Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*. https://www.itu.int/ITU-T/recommendations/rec.aspx?id=9590. Acesso em: 15/02/2020.

[23] W. Z. Khan, M. Y. Aalsalem e M. K. Khan. "Five acts of consumer behavior: A potential security and privacy threat to Internet of Things". Em: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2018, pp. 1–3.

[24] X. Krasniqi e E. Hajrizi. "Use of IoT technology to drive the automotive industry from connected to full autonomous vehicles". Em: *IFAC-PapersOnLine* 49.29 (2016), pp. 269–274.

[25] S. Lucero et al. "IoT platforms: Enabling the Internet of Things". Em: *White paper* (2016).

[26] M. Miller. *The internet of things: How smart TVs, smart cars, smart homes, and smart cities are changing the world*. Pearson Education, 2015.

[27] A. Moghimi, G. Irazoqui e T. Eisenbarth. "CacheZoom: How SGX Amplifies The Power of Cache Attacks". Em: *arXiv:1703.06986 [cs]* (mar. de 2017). arXiv: 1703.06986.

[28] R. Morello, C. De Capua, G. Fulco e S. C. Mukhopadhyay. "A smart power meter to monitor energy flow in smart grids: The role of advanced sensing and IoT in the electric grid of the future". Em: *IEEE Sensors Journal* 17.23 (2017), pp. 7828–7837.

[29] C. NIST. "The digital signature standard". Em: *Communications of the ACM* 35.7 (1992), pp. 36–40.

[30] C. d. l. C. Pinto. "Big Data en Seguridad". Diss. de mestrado. Universitat Oberta de Catalunya, 2018.

[31] R. Pires, M. Pasin, P. Felber e C. Fetzer. “Secure Content-Based Routing Using Intel Software Guard Extensions”. Em: *Proceedings of the 17th International Middleware Conference*. 2016, pp. 1–10.

[32] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov e P. Pietzuch. “SGX-LKL: Securing the host OS interface for trusted execution”. Em: *arXiv preprint arXiv:1908.11143* (2019).

[33] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Ago. de 2018.

[34] R. L. Rivest, A. Shamir e L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. Em: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[35] M. Sabt, M. Achemlal e A. Bouabdallah. “Trusted execution environment: what it is, and what it is not”. Em: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE. 2015, pp. 57–64.

[36] L. Sampaio. “Estratégias para o suporte a ambientes de execução confiável em sistemas de computação na nuvem.” Diss. de mestrado. 2018.

[37] B. Segall, D. Arnold, J. Boot, M. Henderson e T. Phelps. “Content based routing with elvin4”. Em: *Proceedings AUUG2k*. 2000.

[38] L. Silva. “Construção de aplicações seguras orientadas a eventos considerando aspectos de confidencialidade e integridade”. Tese de doutorado. Universidade Federal de Campina Grande, 2019.

[39] M. Sipser. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013.

[40] T. Stack. *Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How?* `https://bit.ly/2INWbs2`. Acesso em: 29/01/2020. Fev. de 2018.

[41] C.-C. Tsai, D. E. Porter e M. Vij. “Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX”. Em: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017, pp. 645–658.

[42] A. Ukil, S. Bandyopadhyay e A. Pal. “Privacy for IoT: Involuntary privacy enablement for smart energy systems”. Em: *2015 IEEE International Conference on Communications (ICC)*. IEEE. 2015, pp. 536–541.

[43] W. Wang, L. Fan, P. Huang e H. Li. “A New Data Processing Architecture for Multi-Scenario Applications in Aviation Manufacturing”. Em: *IEEE Access* 7 (2019), pp. 83637–83650.

[44] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy e T. Anderson. “Slim:OS kernel support for a low-overhead container overlay network”. Em: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 2019, pp. 331–344.

Apêndice A

Configurações de Controle de Acesso do Apache Kafka

Operação	Tipo do Recurso	API Kafka
ALTER	Cluster	AlterReplicaLogDirs
ALTER	Cluster	CreateAcls
ALTER	Cluster	DeleteAcls
ALTER	Topic	CreatePartitions
ALTER_CONFIGS	Cluster	AlterConfigs
ALTER_CONFIGS	Topic	AlterConfigs
CLUSTER_ACTION	Cluster	Fetch (for replication only)
CLUSTER_ACTION	Cluster	LeaderAndIsr
CLUSTER_ACTION	Cluster	OffsetForLeaderEpoch
CLUSTER_ACTION	Cluster	StopReplica
CLUSTER_ACTION	Cluster	UpdateMetadata
CLUSTER_ACTION	Cluster	ControlledShutdown
CLUSTER_ACTION	Cluster	WriteTxnMarkers
CREATE	Cluster	CreateTopics
CREATE	Cluster	Metadata if auto.create.topics.enable
CREATE	Topic	Metadata if auto.create.topics.enable
CREATE	Topic	CreateTopics

DELETE	Group	DeleteGroups
DELETE	Topic	DeleteRecords
DELETE	Topic	DeleteTopics
DESCRIBE	Cluster	DescribeAcls
DESCRIBE	Cluster	DescribeLogDirs
DESCRIBE	Cluster	ListGroups
DESCRIBE	DelegationToken	DescribeTokens
DESCRIBE	Group	DescribeGroup
DESCRIBE	Group	FindCoordinator
DESCRIBE	Group	ListGroups
DESCRIBE	Topic	ListOffsets
DESCRIBE	Topic	Metadata
DESCRIBE	Topic	OffsetFetch
DESCRIBE	Topic	OffsetForLeaderEpoch
DESCRIBE	TransactionalId	FindCoordinator
DESCRIBE_CONFIGS	Cluster	DescribeConfigs
DESCRIBE_CONFIGS	Topic	DescribeConfigs
IDEMPOTENT_WRITE	Cluster	InitProducerId
IDEMPOTENT_WRITE	Cluster	Produce
READ	Group	AddOffsetsToTxn
READ	Group	Heartbeat
READ	Group	JoinGroup
READ	Group	LeaveGroup
READ	Group	OffsetCommit
READ	Group	OffsetFetch
READ	Group	SyncGroup
READ	Group	TxnOffsetCommit
READ	Topic	Fetch
READ	Topic	OffsetCommit

READ	Topic	TxnOffsetCommit
WRITE	Topic	Produce
WRITE	TransactionalId	Produce
WRITE	Topic	AddPartitionsToTxn
WRITE	TransactionalId	AddPartitionsToTxn
WRITE	TransactionalId	AddOffsetsToTxn
WRITE	TransactionalId	EndTxn
WRITE	TransactionalId	InitProducerId
WRITE	TransactionalId	TxnOffsetCommit

Tabela A.1: Lista de operações e tipos de recursos gerenciáveis por ACLs

Apêndice B

Exemplo de Certificado de Chave Pública X.509

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      33:9b:15:29:da:52:2b:91:ef:6e:87:50:07:b5:2f:57:17:2f:b7:1d
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = BR, ST = Paraiba, L = Campina Grande, O = SmartCampus,
            OU = LiteCampus, CN = *.smartiks.com,
            emailAddress = fabiosilva@lsd.ufcg.edu.br
    Validity
      Not Before: Feb  7 01:58:24 2020 GMT
      Not After : Feb  2 01:58:24 2040 GMT
    Subject: C = BR, ST = Paraiba, L = Campina Grande, O = SmartCampus,
             OU = LiteCampus, CN = localhost,
             emailAddress = fabiosilva@lsd.ufcg.edu.br
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
        Modulus:
          00:c0:01:df:3f:42:13:39:aa:d3:67:8c:3a:f7:13:
          e5:f0:a0:d1:0b:0d:da:d6:e6:87:2d:aa:53:59:b9:
          5f:37:72:09:4a:68:eb:0b:6d:2e:76:88:45:f6:a9:
          66:44:50:41:eb:3f:ae:a7:69:29:45:dc:62:9a:28:
```

```
24          a1:fc:db:5a:6b:61:ab:51:7b:32:91:e4:66:e5:39:
25          a2:22:ec:f7:a2:b5:62:91:25:db:65:6a:c9:85:4c:
26          eb:da:81:cd:62:05:6d:d3:07:1a:70:e5:a4:35:72:
27          43:bb:93:5c:9b:f0:77:f8:f7:71:68:df:2e:85:ab:
28          0f:29:1b:58:8f:99:72:04:0c:34:4c:76:6f:a3:25:
29          51:05:47:b1:52:b9:c5:01:ca:fe:2a:18:e7:21:82:
30          c8:1b:c8:56:ab:5f:83:5b:9e:e2:69:26:50:4a:49:
31          92:a5:fd:a5:53:43:51:e0:ea:03:f7:76:03:a5:e7:
32          3e:57:43:c6:00:5c:36:ce:40:b5:aa:4a:7f:92:42:
33          21:97:49:72:2e:e6:fd:75:7c:b4:09:b2:95:da:67:
34          91:e2:eb:3e:f8:98:41:69:d9:b5:0d:0d:e4:38:00:
35          05:a7:64:20:a5:ab:00:5e:88:45:f1:1e:6e:ad:60:
36          f6:35:19:4d:3f:5d:bf:8e:a5:23:a7:4d:a8:e8:14:
37          a0:71
38        Exponent: 65537 (0x10001)
39    X509v3 extensions:
40      X509v3 Subject Alternative Name:
41        DNS:tokman.lsd.ufcg.edu.br, DNS:localhost,
42        IP Address:0.0.0.0, IP Address:127.0.0.1
43  Signature Algorithm: sha256WithRSAEncryption
44     44:81:45:35:fe:25:e2:57:8d:90:74:8e:f4:b6:c3:6d:ef:16:
45     a3:93:2d:bf:d7:df:a1:43:cd:26:24:28:3e:e9:50:df:06:76:
46     e8:60:8d:7e:8d:03:06:6e:95:ed:72:f1:f1:36:5d:a8:2e:b2:
47     92:4f:d1:81:25:fc:04:03:26:ac:02:46:cd:8f:81:50:e8:fd:
48     ae:9f:c4:10:55:19:63:11:f5:a7:ab:dc:be:2b:55:71:3c:16:
49     55:60:8b:d2:d7:96:9e:af:d7:31:fa:a7:8d:44:1e:b3:34:cf:
50     28:93:a5:48:62:f4:ea:a6:f9:56:d9:56:ed:ce:66:22:12:9a:
51     03:90:1a:ad:4f:85:67:b3:ed:9f:9d:6d:f4:aa:76:6c:f9:14:
52     76:a4:1b:39:da:c4:15:bb:ab:8a:eb:76:59:74:f7:84:91:02:
53     4f:03:04:e7:47:c2:43:1d:0c:8d:bc:2e:c5:99:b0:f6:51:72:
54     1c:ca:bb:d1:07:c9:f3:25:2b:51:58:bf:c2:7b:a7:44:f1:39:
55     00:7e:ba:68:8f:c6:1f:6e:da:a0:1f:12:b8:71:9c:06:af:af:
56     72:d7:cf:37:b3:ac:3e:98:a9:d3:84:dd:3f:1b:15:3b:58:bd:
57     6d:22:e3:90:41:fb:b1:7f:a8:64:20:6c:50:1b:d0:ee:9e:60:
58     ac:19:83:5d
```

Código Fonte B.1: Exemplo de certificado de chave pública X.509

Apêndice C

Gramática BNF do Protocolo de Autenticação

```
1  Authenticate => token
2    token => STRING /* Base64 encoded authentication token */
3
4  RA MSG0 => sp_pub_key
5    sp_pub_key => gx gy /* Service provider elliptic curve public key */
6      gx => BYTES /* X coordinate (little-endian) */
7      gy => BYTES /* Y coordinate (little-endian) */
8
9  RA MSG1 => pub_key epid_group_id
10   pub_key => gx gy /* Application enclave elliptic curve public key */
11     gx => BYTES /* X coordinate (little-endian) */
12     gy => BYTES /* Y coordinate (little-endian) */
13   epid_group_id => BYTES /* EPID Groupd ID (little-endian) */
14
15 RA MSG2 => payload
16   payload => BYTES /* Byte-encoded sgx_ra_msg2_t message */
17
18 RA MSG3 => payload
19   payload => BYTES /* Byte-encoded sgx_ra_msg3_t message */
20
21 Secrets => encrypted_skey kafka_topic app_pubkey kafka_credentials
22   encrypted_skey => STRING /* Base64 encoded symmetric key (encrypted
       with application public key) */
```

```
23   kafka_topic => STRING /* Kafka topic */
24   app_pubkey => STRING /* Application public key */
25   kafka_credentials => cert_pem cert_pkey cert_pkey_secret /* Base64
         encoded Kafka Credentials (encrypted with symmetric key) */
26   cert_pem => STRING /* Encrypted Certificate (PEM format) */
27   cert_pkey => STRING /* Encrypted Certificate Private Key (PEM format)
         */
28   cert_pkey_secret => NULLABLE_STRING /* Encrypted Cert. Priv. Key secret
          */
29
30 Encrypted Secrets => encrypted_secrets
31   encrypted_secrets => BYTES /* AES128-GCM encrypted secrets using Diffie
         -Hellman shared secret SMK derived key. */
```

Código Fonte C.1: Gramática BNF do protocolo de autenticação

Apêndice D

Compilador para Protocolos Doc2C

Visando otimizar o processo de implementação de suporte a protocolos, um compilador foi desenvolvido para esta finalidade. Escrito utilizando a linguagem de programação Python, o Doc2C utiliza como entrada uma espeficiação de protocolo, definida através de uma gramática com notação BNF (do inglês *Backus-Naur Form* ou Formalismo de Backus-Naur), e produz código-fonte em linguagem C++ na saída. A notação BNF é bastante utilizada para especificar a sintaxe de gramáticas livres-de-contexto [2].

Figura D.1: Compilador Doc2C

Sipser (2013) define uma gramática livre-de-contexto como sendo uma 4-upla (V, Σ, R, S), onde [39]:

- V é um conjunto finito denominado **variáveis**;
- Σ é um conjunto finito, disjunto de V, denominado **terminais**;
- R é um conjunto de **regras**, com cada regra sendo uma variável e uma cadeia de variáveis e terminais; e

- $S \in V$ é uma variável inicial.

A sintaxe utilizada para especificar gramáticas bem como o conjunto de símbolos terminais suportados foram baseados na especificação do protocolo de comunicação [4] do Apache Kafka. O formato original foi estendido com o suporte à documentação, onde, no processo de compilação, a mesma é transformada em blocos de comentário no código-fonte C++ de saída.

(**variável**) => (**variável | terminal**) ... (**variável | terminal**) /* **<comentários>** */ (D.1)

O código-fonte C++ gerado define um conjunto de tipos T, representado através de estruturas de variáveis (*structs*). Para cada tipo definido, um conjunto de funções H é gerado, responsáveis por:

- realizar o processo de **leitura**, onde, dada uma cadeia de bytes que representa uma variável, preenche uma estrutura de variáveis;
- realizar operações de **desalocação** de regiões da memória que possam ter sido alocados dinamicamente no processo de leitura;
- realizar o processo de **escrita**, onde, dada uma estrutura de variáveis, escreve a sua representação em uma cadeia de bytes;
- **medir** a quantidade de bytes necessários para representar uma estrutura de variáveis em uma cadeia de bytes.

Sendo $P(t, v)$ o predicado "t representa v", e V' o conjunto de variáveis, cujas suas regras sejam uma cadeia de variáveis e terminais maior que um:

$$\begin{gathered} V' \subset V \\ (\forall v \in V')(\exists t \in T) P(t, v) \\ |V'| = |T| \end{gathered} \tag{D.2}$$

Para exemplificar o uso do Doc2C, considere a seguinte gramática:

```
MSG1 => pub_key epid_group_id
pub_key => gx gy /* App. enclave elliptic curve public key */
gx => BYTES /* X coordinate (little-endian) */
gy => BYTES /* Y coordinate (little-endian) */
epid_group_id => BYTES /* EPID Groupd ID (little-endian) */
```

Código Fonte D.1: Exemplo de gramática BNF com comentários

Para a gramática apresentada em D.1, de acordo com a definição de gramática livre-de-contexto:

$$
\begin{gathered}
V = \{MSG1,\ pub\_key,\ epid\_group\_id,\ gx,\ gy\} \\
\Sigma = \{BYTES\} \\
R = \{MSG1 \rightarrow pub\_key\ epid\_group\_id, \\
pub\_key \rightarrow gx\ gy,\ gx \rightarrow BYTES,\ gy \rightarrow BYTES, \\
epid\_group\_id \rightarrow BYTES\} \\
S = MSG1
\end{gathered}
\tag{D.3}
$$

O código C++ gerado pela ferramenta, contendo apenas as estruturas de variáveis e as declarações de funções, de acordo com o exposto em D.2, sendo $V' = \{MSG1,\ pub\_key\}$:

```
/**
* @struct pub_key
* @brief App. enclave elliptic curve public key
*/
typedef struct pub_key
{
    kp_bytes gx; /**< X coordinate (little-endian) */
    kp_bytes gy; /**< Y coordinate (little-endian) */
} pub_key;

/**
* @struct msg1
* @brief MSG1 API
*/
```

```
15 typedef struct msg1
16 {
17     pub_key pub_key; /**< App. enclave elliptic curve public key */
18     kp_bytes epid_group_id; /**< EPID Groupd ID (little-endian) */
19 } msg1;
20
21 /**
22  * @fn size_t read_msg1(void *msg1_p, char *buffer, size_t *cursor)
23  * @brief Parse msg1 struct from a given @p buffer at @p cursor position
24  *
25  * This function parses the msg1 struct from a given @p buffer at @p
        cursor position.
26  * At the end of the execution, the @p cursor will be updated to the
        position after
27  * the amount of bytes consumed by the function.
28  * @param msg1_p msg1 pointer to empty struct to be filled
29  * @param buffer a buffer to be parsed by the function
30  * @param cursor the position on the buffer containing the msg1 data
31  * @return the number of read bytes
32  */
33 size_t read_msg1(void *msg1_p, char *buffer, size_t *cursor);
34
35 /**
36  * @fn void free_msg1(void *msg1_p)
37  * @brief Free msg1 struct
38  *
39  * This function free any memory pointer allocated by read_msg1()
        function.
40  * @param msg1_p msg1 pointer to the struct to be free'ed
41  */
42 void free_msg1(void *msg1_p);
43
44 /**
45  * @fn size_t serialize_msg1(void *_msg1, char *buffer)
46  * @brief Serialize @p _msg1 struct into @p buffer
47  *
48  * This function serializes msg1 struct into @p buffer.
```

```
49  * @param _msg1 pointer to the struct to be serialized
50  * @param buffer pointer to the target buffer
51  * @return num. of bytes written to the buffer
52  */
53 size_t serialize_msg1(void *_msg1, char *buffer);
54
55 /**
56  * @fn size_t measure_msg1(void *msg1_p)
57  * @brief Measure the size (in bytes) of msg1 struct in serialized format
58  *
59  * This function measures/predict the size (in bytes) of msg1 struct in
        serialized
60  * (buffer) format. It's useful for allocating buffers to store
        serialized structs.
61  * @param msg1_p msg1 pointer to the struct to be measured
62  * @return measured size
63  */
64 size_t measure_msg1(void *msg1_p);
65
66 /**
67  * @fn size_t read_pub_key(void *pub_key_p, char *buffer, size_t *cursor)
68  * @brief Parse pub_key struct from a given @p buffer at @p cursor
        position
69  *
70  * This function parses the pub_key struct from a given @p buffer at @p
        cursor position.
71  * At the end of the execution, the @p cursor will be updated to the
        position after
72  * the amount of bytes consumed by the function.
73  * @param pub_key_p pub_key pointer to empty struct to be filled
74  * @param buffer a buffer to be parsed by the function
75  * @param cursor the position on the buffer containing the pub_key data
76  * @return the number of read bytes
77  */
78 size_t read_pub_key(void *pub_key_p, char *buffer, size_t *cursor);
79
80 /**
```

```
81   * @fn void free_pub_key(void *pub_key_p)
82   * @brief Free pub_key struct
83   *
84   * This function free any memory pointer allocated by read_pub_key()
         function.
85   * @param pub_key_p pub_key pointer to the struct to be free'ed
86   */
87  void free_pub_key(void *pub_key_p);
88
89  /**
90   * @fn size_t serialize_pub_key(void *_pub_key, char *buffer)
91   * @brief Serialize @p _pub_key struct into @p buffer
92   *
93   * This function serializes pub_key struct into @p buffer.
94   * @param _pub_key pointer to the struct to be serialized
95   * @param buffer pointer to the target buffer
96   * @return num. of bytes written to the buffer
97   */
98  size_t serialize_pub_key(void *_pub_key, char *buffer);
99
100 /**
101  * @fn size_t measure_pub_key(void *pub_key_p)
102  * @brief Measure the size (in bytes) of pub_key struct in serialized
         format
103  *
104  * This function measures/predict the size (in bytes) of pub_key struct
         in serialized
105  * (buffer) format. It's useful for allocating buffers to store
         serialized structs.
106  * @param pub_key_p pub_key pointer to the struct to be measured
107  * @return measured size
108  */
109 size_t measure_pub_key(void *pub_key_p);
```

Código Fonte D.2: Exemplo de código C++ gerado pelo Doc2C

Através do uso desta abordagem de compilador de protocolos, foi possível detectar automaticamente inconsistências na especificação do protocolo do Apache Kafka. Sugestões

para correções destas inconsistências foram submetidas e aceitas pela comunidade de desenvolvedores[1][2].

Esta ferramenta foi utilizada como apoio no desenvolvimento do componente Kafka Proxy, na geração do código que viabiliza o suporte ao protocolo de comunicação do Apache Kafka [4], e também foi utilizada para gerar o código de suporte ao protocolo de autenticação do sistema gerenciador de chaves.

[1] https://issues.apache.org/jira/browse/KAFKA-9255

[2] https://issues.apache.org/jira/browse/KAFKA-8698

Apêndice E

Comandos utilizados na avaliação de performance

```
$ ./bin/kafka-topics.sh \
    --bootstrap-server <ENDERECO DO BROKER/PROXY> \
    --create \
    --topic proxy-test \
    --replication-factor 3 \
    --partitions 9
```

Código Fonte E.1: Comando para criação do tópico

```
$ ./bin/kafka-producer-perf-test.sh \
    --producer-props \
      bootstrap.servers=<ENDERECO DO BROKER/PROXY> \
    --topic proxy-test \
    --num-records 2500000 \
    --record-size <TAMANHO DA MENSAGEM> \
    --throughput -1
```

Código Fonte E.2: Comando para teste de performance de produção

```
$ ./bin/kafka-consumer-perf-test.sh \
    --broker-list <ENDERECO DO BROKER/PROXY> \
    --topic proxy-test \
    --threads 1 \
    --timeout 100000 \
```

```
6     --messages 2500000
```

Código Fonte E.3: Comando para teste de performance de consumo

```
1  $ ./bin/kafka-run-class.sh kafka.tools.EndToEndLatency \
2      <ENDERECO DO BROKER/PROXY> \
3      proxy-test \
4      200 \
5      1 \
6      <TAMANHO DA MENSAGEM> \
7      1
```

Código Fonte E.4: Comando para teste de latência ponto-a-ponto[1]

```
1  $ pidstat -IurhH -p <PID do Kafka Proxy> 1
```

Código Fonte E.5: Comando para coleta de informações sobre consumo de recursos

[1] A aplicação utilizada para teste de latência ponto-a-ponto foi minimamente modificada para apresentar a latência para todas as mensagens isoladamente. O código fonte da versão modificada se encontra em https://github.com/ffosilva/kafka/commit/645378122d6848f4a.