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

Tese de Doutorado

Construção de aplicações seguras orientadas a eventos considerando aspectos de confidencialidade e integridade

Leandro José Ventura Silva

Campina Grande – Paraíba – Brasil
Setembro de 2019

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

Construção de aplicações seguras orientadas a eventos considerando aspectos de confidencialidade e integridade

Leandro José Ventura Silva

Tese 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 Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Arquiteturas de Software

Andrey Elísio Monteiro Brito
(Orientador)

Campina Grande, Paraíba, Brasil

S586c Silva, Leandro José Ventura.
Construção de aplicações seguras orientadas a eventos considerando aspectos de confiabilidade e integridade. / Leandro José Ventura Silva. – Campina Grande, 2019.
130 f.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2019.
"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito".
Referências.

1. Arquitetura de software. 2. Nuvem - arquitetura segura. 3. Conceitos de segurança. 4. Decisões arquiteturais. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU 004.273(043)

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO JESIEL FERREIRA GOMES CRB-15/256

"CONSTRUÇÃO DE APLICAÇÕES SEGURAS ORIENTADAS A EVENTOS CONSIDERANDO ASPECTOS DE CONFIDENCIALIDADE E INTEGRIDADE"

LEANDRO JOSÉ VENTURA SILVA

TESE APROVADA EM 02/09/2019

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

FRANCISCO VILAR BRASILEIRO, Ph.D, UFCG
Examinador(a)

JOÃO ARTHUR BRUNET MONTEIRO, Dr., UFCG
Examinador(a)

MARCO VIEIRA, Dr., UC
Examinador(a)

KEIKO VERÔNICA ONO FONSECA, Dra., UTFPR
Examinador(a)

CAMPINA GRANDE - PB

Resumo

O mundo encontra-se atualmente na era do *Big Data*. O número previsto de dispositivos conectados na Internet na próxima década é de centenas de bilhões. Esses dispositivos vão de aviões, carros, computadores, celulares a até mesmo pequenos sensores de batimentos cardíacos.

Processar esses volumes de dados exige grandes poderes computacionais. Ademais, o processamento seguro de dados deixou de ser um requisito desejável e passou a ser necessário, aumentando o custo computacional. Com isso, é proposto um modelo de arquitetura de software com foco no aumento da segurança dos dados processados, em termos de confiabilidade e integridade. Esse modelo é uma especialização do padrão *publish-subscribe* com entidades adicionais para controle de acesso, checagem da integridade de aplicações e dados e para gerenciamento seguro de chaves de criptografia. O modelo proposto foi utilizado para o desenvolvimento de duas aplicações. A primeira aplicação é na área de redes elétricas inteligentes (*smart grids*), especificamente na coleta de informações de consumo de residências com o uso de medidores inteligentes. A segunda aplicação consiste na captura de dados de dispositivos móveis para uso em publicidade segmentada. Em ambos os casos, houve coleta de informações que permitem modelar perfis de comportamento dos usuários.

As principais contribuições do modelo são as garantias de segurança dos dados, ao mesmo tempo em que é escalável, facilmente replicável em diversos provedores, sejam eles de nuvem ou não, e possui baixo acoplamento.

Abstract

The world is currently in the era of Big Data. The predicted number of connected devices to the Internet in the next decade is in the hundreds of billions. These devices range from planes, cars, computers, cell phones to even small heart rate sensors.

Processing these data volumes requires great computational powers. In addition, secure data processing is no longer a desirable requirement, increasing the computational cost. Thus, a software architecture model is proposed focusing on increasing the security of the processed data, in terms of reliability and integrity. This model is a specialization of the *publish-subscribe* standard with additional entities for access control, application and data integrity checking, and secure encryption key management. The proposed model was used for the development of two applications. The first application is in the area of smart grids, specifically in collecting household consumption information using smart meters. The second application is mobile data capture for use in targeted advertising. Both cases contain sensitive information that allowed to discover user behavior profiles.

The main contributions of the model are data security guarantees, while scalable and compatibility across multiple clouds and non-cloud providers.

Agradecimentos

Agradeço primeiramente a Deus, por me dar força de vontade, garra, inteligência e sabedoria para trilhar os meus caminhos.

Agradeço a meu pai (in memoriam), Leonidas, um homem à frente do seu tempo, cujo legado não será esquecido.

Agradeço também a minha mãe, Eyres e ao meu irmão, Rômulo, que sempre me apoiaram e me deram forças para não desistir dessa jornada.

Também sou muito grato a minha amada esposa e aos meus queridos filhos do coração, Abraão e Alice, que vieram para iluminar minha vida; esses três também sempre me fizeram continuar firme nessa conquista.

A Andrey, um professor excelente que soube como me suportar e também quando cobrar.

A Eloi, por sempre me ajudar na Smartiks e me aliviar um pouco no trabalho para que eu pudesse dar continuidade às atividades.

Aos companheiros do LSD, da Smartiks e do STI, por acreditarem no meu trabalho e sempre quebrar galhos para mim quando necessário.

A todos os meus amigos mais próximos; obrigado a todos pela torcida.

Por fim, ao pessoal da COPIN, que sempre me auxiliaram quando necessário.

Conteúdo

Lista de Símbolos

AES-GCM - *Advanced Encryption Standard - Galois/Counter Mode*

APB - *Advanced Peripheral Bus*

ASLR - *Address Space Layout Randomization*

AXI - *Advanced eXtensible Interface*

ECDH - *Elliptic-curve Diffie–Hellman*

IBM - *International Business Machines Corporation*

IaaS - *Infrastructure as a Service*

IoT - *Internet of Things*

KVM - *Kernel-based Virtual Machine*

NFC - *Near Field Communication*

PaaS - *Platform as a Service*

PKI - *Public Key Infrastructure*

RFID - *Radio-Frequency Identification*

SaaS - *Software as a Service*

SCONE - *Secure CONtainer Environment*

SEV - *Secure Encrypted Virtualization*

SGX - *Secure Guard eXtensions*

SLA - *Service Level Aggrement*

SME - *Secure Memory Encryption*

SMM - *System Management Model*

TEE - *Trusted Execution Environment*

TZASC - *TrustZone Address Space Controller*

UEFI - *Uniform Extensible Firmware Interface*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e contextualização

Vivemos atualmente na realidade conhecida como a era do *Big Data*. Um estudo feito pela IBM em 2017 descreve que 90% de todos os dados até esse ano foram criados nos dois anos anteriores [1]. É previsto que o número de dispositivos interconectados cresça de 2 bilhões em 2006 para 200 bilhões em 2020 [2]. Para as organizações, a responsabilidade sobre os dados aumenta à medida que mais informações sobre os seus usuários são armazenadas.

Os casos de vazamento de dados são recorrentes, e, dentre eles, é possível destacar um ocorrido na Índia em 2018, onde informações de nomes, números de identidade, dados bancários, entre outros dados sensíveis de mais de um bilhão de pessoas foram expostas [3] e na Marriott International em 2014, empresa dona de um conglomerado de hotéis de luxo, que sofreu um ataque que resultou no roubo de nomes, endereços, telefones, datas de nascimento, endereços de e-mail e cartões de créditos de meio bilhão de pessoas [4].

Além da exposição dos dados de bilhões de pessoas, as empresas sofrem prejuízos expressivos. A Target, rede de lojas de varejo nos Estados Unidos, divulgou que os custos relacionados à violação dos dados de 70 milhões de usuários atingiram US$ 290 milhões [5].

Historicamente, as empresas priorizam funcionalidades em detrimento de segurança, porém, com o aumento das violações de segurança e os prejuízos associados, a preocupação com requisitos de segurança cresceu [6; 7]. Todavia, desenvolver aplicações seguras não é trivial, visto que os erros de implementação são a principal fonte de vulnerabilidades, repre-

sentando dois terços do total [8].

Ao projetar novos sistemas, é necessário priorizar requisitos de segurança desde sua concepção, a fim de evitar ou minimizar incidentes de segurança. Sendo assim, a proposta deste trabalho é um modelo de arquitetura de software com foco em requisitos de segurança e escalabilidade, com o objetivo de diminuir erros de implementação a partir da redução das dificuldades envolvidas no desenvolvimento de aplicações seguras cujo processamento de grandes volumes de dados é obrigatório.

Tradicionalmente, requisitos de segurança são compostos pela tríade de **confidencialidade** – limitação de acessos não autorizados aos dados), **integridade** – manutenção e garantia da precisão e consistência dos dados durante todo o ciclo de vida da informação [9] – e **disponibilidade** – garantia de que a informação está acessível quando necessária.

O modelo proposto não lida diretamente com requisitos de privacidade, todavia, preocupações com a confidencialidade evitam o acesso indevido aos dados, o que gera algumas garantias de privacidade. Além disso, o foco do modelo é evitar vazamentos de dados e não são tratadas questões de disponibilidade, mas é permitido ao desenvolvedor adicionar técnicas próprias, caso necessário.

Para avaliar aplicações específicas e ter uma melhor contribuição, o modelo é centrado em aplicações orientadas a eventos. Essa escolha se deve à popularidade da *Internet das Coisas*, bem como outros sistemas que geram dados continuamente.

O modelo arquitetural foi utilizado em aplicações reais no ambiente de empresas. Com isso, ele provê mecanismos de proteção de dados mesmo sem saber como os mesmos serão usados no futuro. No caso de anonimização, por exemplo, é preciso ter conhecimento de como eles serão utilizados.

Esse modelo pode ser utilizado com: (i) tecnologias de hardware para garantir a autenticidade do código executado, assegurando sigilo no processamento de dados críticos e (ii) técnicas formais para processamento em dados criptografados. Visto que cada aplicação tem suas peculiaridades, diferentes arquiteturas podem ser desenvolvidas de forma específica seguindo o modelo proposto, bem como diferentes técnicas podem ser aplicadas.

O modelo é implantável tanto em provedores públicos que oferecem serviços de computação confidencial, como IBM SoftLayer, Microsoft Azure, Cloud&Heat, CloudSigma, Alibaba Cloud e Baidu, quanto em infraestruturas privadas.

1.2 Metodologia

Inicialmente, o objetivo da pesquisa foi identificar os desafios na construção de plataformas escaláveis para processamento de dados. Durante as pesquisas realizadas, priorizou-se o estudo da identificação de formas para atender requisitos de segurança na nuvem em diferentes casos de uso e, quais os ganhos e perdas envolvidos nessas abordagens.

As atividades de pesquisa realizadas foram:

1. Levantamento de requisitos, facilidades e dificuldades na construção de aplicações escaláveis para o processamento de grandes quantidades de dados;
2. Elaboração de uma arquitetura de software preliminar para agregação de dados na nuvem;
3. Uso da arquitetura de software para desenvolvimento de duas aplicações utilizando técnicas de criptografia homomórfica e soluções de hardware – Intel *Software Guard eXtensions* (SGX);
4. Comparação em termos de desempenho entre as técnicas escolhidas;
5. Desenvolvimento de um modelo de arquitetura de software com foco em segurança e privacidade;
6. Avaliação do modelo de arquitetura com terceiros.

1.3 Contribuições

Desenvolver sistemas que aplicam técnicas de segurança corretamente é uma tarefa complexa, pois exige conhecimentos específicos e, o ideal é que isso seja tratado desde a concepção da arquitetura de software. Em aplicações orientadas a eventos, isso se torna mais crítico pois há coleta constante de dados, que podem conter informações sensíveis sobre os usuários.

O modelo foi utilizado em aplicações de medição do consumo de energia, publicidade segmentada e contagem de pessoas. Com isso, foram coletadas evidências de que o modelo pode ser usado em casos que há coleta contínua de dados sigilosos e reduz a complexidade

de novas aplicações. Por fim, com base nos experimentos realizados, foi possível identificar quanto houve de perda de desempenho, em benefício da confidencialidade e integridade dos dados.

A contribuição desse trabalho é reforçada com as publicações:

1. Agregação de Dados na Nuvem com Garantias de Segurança e Privacidade no **XVI SBSEG – Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais** [10];

2. Security and Privacy Preserving Data Aggregation in Cloud Computing no **32nd ACM SIGAPP Symposium On Applied Computing – The 16th Edition of the Computer Security Track** [11];

3. Security and privacy aware data aggregation on cloud computing na **Journal of Internet Services and Applications** [12].

1.4 Organização

O restante do documento está organizado da seguinte forma: o Capítulo 2 contém o embasamento teórico necessário para o entendimento desse trabalho. O Capítulo 3 possui o modelo proposto. Nos Capítulos 4 e 5 são apresentados exemplos de uso do modelo para o cálculo do consumo médio de energia elétrica e para aplicações de localização *indoor*. O Capítulo 6 detalha a avaliação realizada do modelo arquitetural. Por fim, no Capítulo 7 retoma-se a discussão sobre as contribuições desse trabalho, perguntas de pesquisa e as atividades realizadas durante este trabalho.

Capítulo 2

Fundamentação Teórica

2.1 Arquiteturas de Software

O crescimento da computação em nuvem promoveu a integração de diversos serviços, que geralmente estão alocados em diferentes áreas geográficas. Com o aumento do uso das tecnologias da informação em diversos campos, tomar decisões para a construção de softwares torna-se cada vez mais delicado.

Por tais motivos, dentro da grande área de Ciência da Computação e especificamente na parte de Engenharia de Software, que por si só é uma área relativamente nova, surgiu o estudo acerca das arquiteturas de software.

Apesar de não existir uma definição formal de Arquitetura de Software, esse conceito foi proposto inicialmente por PERRY et al. [13] na publicação **Foundations for the Study of Software Architecture** e **An introduction to software architecture** de GARLAN et al. [14].

O primeiro define por analogia com a arquitetura de edifícios, e propõe o seguinte modelo de arquitetura de software:

$$ArquiteturadeSoftware = \{Elementos, Forma, Fundamentação\} \tag{2.1}$$

Ou seja, uma arquitetura de software é um conjunto de elementos arquitetônicos (ou, se quiser, design) que possuem uma forma específica.

Já GARLAN et al. [15] relembram que, assim como os bons programadores reconheceram estruturas úteis de dados no final dos anos 60, os bons projetistas de sistemas de software

reconhecem agora organizações de sistemas úteis. Um deles é baseado na teoria dos tipos de dados abstratos. Mas esta não é a única maneira de organizar um sistema de software. Muitas outras organizações se desenvolveram informalmente ao longo do tempo e agora fazem parte do vocabulário de projetistas de sistemas de software. Essas ditas *organizações* seriam chamadas de **Arquitetura de Software**.

Outra definição pode ser encontrada na norma ISO/IEC/IEEE 42010:2011 [16], que define a arquitetura de software como propriedades e conceitos que fundamentam um sistema, isto inclui seus elementos, relacionamentos e os princípios seguidos para projeto e evolução deste sistema. A arquitetura deve ser pensada de modo que supra as necessidades do sistema objeto.

Uma arquitetura de software é caracterizada pelo conjunto de componentes de um sistema, como eles se comportam, como se relacionam e pelas diretrizes que nortearam o desenvolvimento dos mesmos.

Com relação a sua utilidade, podemos citar os seguintes [17; 18].

- **Comunicação:** Desenvolvedores, gerentes de desenvolvimento e arquitetos, gerentes de projeto, testadores, documentadores e até mesmo clientes precisam entender o sistema, sendo assim, quanto mais alto nível, melhor a comunicação;

- **Contexto do sistema:** Os desenvolvedores e futuros mantenedores precisam entender o sistema como um todo. Em grandes sistemas, é difícil para os desenvolvedores compreenderem todos os seus os detalhes de forma eficiente. A falta do entendimento de estruturas relacionadas pode reduzir a qualidade do sistema em termos de desempenho, segurança e disponibilidade;

- **Alocação de trabalho:** É possível particionar o trabalho dos desenvolvedores de forma eficiente a partir da decomposição do sistema em estruturas relativamente independentes;

- **Redução dos custos de manutenção e evolução:** Arquiteturas podem minimizar esses custos com a antecipação de possíveis mudanças no sistema, garantindo que o projeto do mesmo facilite tais mudanças e documentando como elas podem afetar os elementos arquiteturais envolvidos;

- **Reúso e integração com sistemas legados e softwares de terceiros:** Uma arquitetura pode ser projetada para permitir e facilitar o reúso de componentes, arcabouços, bibliotecas, sistemas legados e softwares de terceiros.

No contexto desse trabalho, uma arquitetura de software é composta por partes interessadas, requisitos, decisões arquiteturais, pontos de vista e visões arquiteturais. Cada um desses elementos será discutido a seguir.

2.1.1 Partes Interessadas

Uma parte interessada (ou *stakeholder)* é qualquer entidade – seja ela um indivíduo, time ou organização – que possui interesse em um sistema. Seguem abaixo alguns tipos de partes interessadas.

1. **Usuários finais:** aqueles que irão, de fato, utilizar o sistema;
2. **Clientes:** responsáveis pela contratação do sistema. Em muitos projetos, são também os usuários finais;
3. **Desenvolvedores:** são responsáveis pela implementação do sistema de acordo com as restrições impostas pela arquitetura e outras partes;
4. **Testadores:** partes que irão verificar se o sistema se comporta de maneira esperada em diversos cenários;
5. **Implantadores:** partes encarregadas de instalar o sistema e mantê-lo operacional;
6. **Gerentes de desenvolvimento:** coordenam a equipe de desenvolvimento, participam da escolha de tecnologias e orientam os programadores;
7. **Arquitetos:** indivíduos responsáveis pela definição da arquitetura e, mais formalmente, pela produção da descrição arquitetural [19].

2.1.2 Requisitos

Um requisito pode ser definido como [20; 18]:

1. Uma condição ou capacidade necessária para uma parte interessada resolver um problema ou atingir um objetivo;

2. Uma condição ou capacidade que deve ser alcançada ou possuída por uma solução, ou componente de solução, para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos;

3. Uma representação documentada de uma condição ou capacidade como em (1) ou (2).

Os requisitos podem ser: (i) **do negócio** – objetivos, metas e necessidades da empresa; (ii) **de partes interessadas**; (iii) **da solução** – **funcionais** ou **não-funcionais**, os primeiros descrevem o comportamento e a informação que a solução irá gerenciar, os últimos descrevem qualidades que a solução deve possuir; (iv) **de transição**, descrevem capacidades que a solução deve possuir com o objetivo de facilitar a transição do estado atual da organização para um estado futuro desejado [18].

Os **requisitos de solução** citados anteriormente podem ser **requisitos arquiteturais**, isso acontece quando os mesmos capturam funcionalidades essenciais para o sistema; atingem vários elementos arquiteturais; possuem exigências rigorosas à arquitetura ou envolvem comunicação e sincronização com sistemas externos [21].

2.1.3 Decisões, Pontos de Vista e Visões Arquiteturais

Uma **decisão** pode ser considerada **arquitetural** quando a mesma se propõe a alcançar um ou mais atributos de qualidade do sistema [22; 23].

Um **ponto de vista** é um arcabouço conceitual que define elementos, conexões e técnicas que compõem uma visão arquitetural, além de especificar seu propósito de acordo com seus interessados [23]. Já as **visões** expressam a arquitetura de um sistema ou de parte dele a partir da perspectiva de um conjunto de interesses relacionados [16]. Visto isso, um **ponto de vista** é utilizado para elaborar diversas **visões arquiteturais**.

Existem diversas formas de se observar um sistema, um exemplo é o modelo **4+1**, composto por visão **lógica** – definição das unidades do sistema, como as classes, no caso de um modelo orientado a objetos, de **processos** – contém aspectos de concorrência e sincronização, **física** – mapeamento entre partes de software e de hardware, de **desenvolvi-**

mento – descreve como o software é dividido em termos de pacotes ou módulos em seu ambiente de desenvolvimento. As **decisões arquiteturais** são espalhadas nessas quatro visões e os **cenários**, que contemplam a quinta visão, descrevem como as outras visões interagem entre si [24].

O uso dessas visões facilita o projeto, a documentação e o entendimento da **arquitetura**, já que é possível abstrair detalhes dependendo de cada **visão**.

2.2 Conceitos de Segurança

Esta seção apresenta os principais conceitos de segurança a serem abordados no trabalho. A partir desses conceitos, serão propostas soluções seguras para o processamento de dados por uso de tecnologias de hardware e de software.

2.2.1 Confidencialidade

De acordo com o *Computer Security Resource Center* do NIST (National Institute of Standards and Technology)[1], **confidencialidade** significa preservar as restrições autorizadas no acesso e divulgação de informações, incluindo meios para proteger a privacidade pessoal e informações proprietárias. Confidencialidade refere-se a proteger a informação de ser acessada por partes não autorizadas.

O NIST SP 800-33 [25] define confidencialidade como o objetivo de segurança que gera o requisito de proteção contra tentativas intencionais ou acidentais de executar leituras de dados não autorizadas. A confidencialidade cobre os dados no armazenamento, durante o processamento e durante o trânsito.

Em outras palavras, apenas as pessoas autorizadas podem obter acesso a dados confidenciais. Uma falha em manter a confidencialidade significa que alguém que não deveria ter acesso conseguiu obtê-lo, por comportamento intencional ou por acidente. Tal falha de confidencialidade, comumente conhecida como violação, normalmente não pode ser remediada.

[1]Mais informações em `https://csrc.nist.gov`

2.2.2 Integridade

A **integridade** refere-se a garantir a autenticidade da informação – essa informação não é alterada e a fonte da informação é genuína. De acordo com o *Computer Security Resource Center*, integridade significa evitar a modificação ou a destruição indevida de informações e inclui assegurar a autenticidade das informações [26].

Também pode ser definida como uma propriedade em que os dados não foram alterados de maneira não autorizada desde que foram criados, transmitidos ou armazenados. Na recomendação NIST SP 800-56B Rev. 2 [27], a afirmação de que um algoritmo criptográfico "fornece integridade de dados" significa que o algoritmo é usado para detectar alterações não autorizadas.

Além disso, o NIST SP 800-33 [25] define integridade como o objetivo de segurança que gera o requisito de proteção contra tentativas intencionais ou acidentais de violar a integridade dos dados (a propriedade de que os dados não foram alterados de maneira não autorizada) ou integridade do sistema (a qualidade que um sistema tem quando executa a função pretendida de uma maneira não prejudicada, livre de manipulação não autorizada).

2.2.3 Privacidade

De acordo com o NIST SP 800-32 [28], privacidade significa restringir o acesso a informações do assinante ou parte confiante de acordo com a lei federal e a política da agência.

Apesar de se tratar de um termo comum em outras áreas do conhecimento, neste trabalho será considerada a definição do NIST. OLDEHOEFT [29] define privacidade como o direito de uma parte de manter o controle e a confidencialidade das informações sobre si.

2.3 Soluções para processamento seguro de dados

Esta seção possui uma breve introdução a abordagens para processar dados de forma segura. Dentre elas, **criptografia homomórfica**, que pode ser implementada completamente por *software* e outras soluções, que exigem *hardware* específicos.

2.3.1 Criptografia Homomórfica

RIVEST et al. [30] criaram o conceito de criptografia homomórfica em 1978 – métodos criptográficos que permitem certas computações em dados cifrados sem a necessidade de acesso às chaves privadas. Modelos completamente homomórficos se caracterizam por permitir operações de adição e multiplicação em blocos criptografados, de modo que o valor retornado seja uma encriptação do resultado das operações aplicadas sobre os dados originais. Embora o conceito não seja recente, estes modelos foram considerados puramente teóricos até que um sistema válido foi proposto por GENTRY [31]. Entretanto, questões relacionadas à eficiência ainda constituem uma barreira. Atualmente, todos os tipos de esquemas de criptografia completamente homomórfica propostos ainda possuem um longo caminho de evolução antes que sejam utilizados na prática [32].

2.3.2 Sistema de Criptografia *Unpadded RSA*

O algoritmo de criptografia RSA funciona da seguinte forma:

- *Inicialização*: dois números primos suficientemente grandes q and p são escolhidos. Em seguida, calcula-se $N = p \cdot q$.

- *Geração de chaves*: calcula-se $\phi(N) = (p-1) \cdot (q-1)$. Em seguida, seleciona-se um número primo como expoente público e tal que $e \in [3, \phi(N))$ e $mdc(e, \phi(N)) = 1$, onde mdc representa o maior divisor comum. Além disso, calcula-se o expoente privado d tal que $e \cdot d \equiv 1$. A chave pública é composta por (e, N) e a chave privada correspondente, (d, N).

- *Criptografia*: uma mensagem $m < N$ está criptografada sob a chave pública ao computar o cifrotexto $E(m) = c = m^e \ mod \ N$.

- *Descriptografia*: um cifrotexto $E(m)$ é descriptografado utilizando a chave privada, calculando $m = c^d \ mod \ N$.

Dadas as mensagens m_1 e m_2, é possível obter uma criptografia de $m_1 \cdot m_2$ ao calcular:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= m_1^e \cdot m_2^e \; mod \; N \\ &= (m_1 \cdot m_2)^e \; mod \; N \\ &= E(m_1 \cdot m_2) \; . \end{aligned}$$

Logo, *unpadded RSA* é um sistema de criptografia homomórfica multiplicativa [33].

2.3.3 Sistema de Criptografia Paillier

Paillier [34] é outro algoritmo de criptografia assimétrica. Seu funcionamento é explicado abaixo:

- *Inicialização*: dois números primos suficientemente grandes p e q são escolhidos, $N = p \cdot q$, e $\lambda = mmc(p-1, q-1)$, onde mmc é o mínimo múltiplo comum. Um número aleatório $g \in_R \mathbb{Z}^*_{N^2}$ é escolhido de forma que $mdc(b, N) = 1$, onde $b = L(g^\lambda \; mod \; N^2)$ e $L(u) = \frac{(u-1)}{N}$.

- *Geração de chaves*: seja μ o inverso multiplicativo modular de b modulo N, ou seja, $\mu = b^{-1} \; mod \; N$. Então, a chave pública $P_k = (N, g)$ e a chave privada é $S_k = (N, \lambda, \mu)$.

- *Criptografia*: uma mensagem m é criptografada sob a chave P_k ao escolher um número aleatório $r \in_R \mathbb{Z}^*_{N-1}$ e computar $E_{P_k}(m) = g^m \cdot r^N \; mod \; N^2$.

- *Descriptografia*: um cifrotexto $c = E_{P_k}(m)$ é descriptografado ao utilizar a chave privada S_k para calcular $m = L(c^\lambda \; mod \; N^2) \cdot \mu \; mod \; N$.

Dadas as mensagens m_1 e m_2, pode-se obter uma criptografia de $m_1 + m_2$ ao computar:

$$\begin{aligned} E_{P_k}(m_1) \cdot E_{P_k}(m_2) &= g^{m_1} \cdot r_1^N \cdot g^{m_2} \cdot r_2^N \; mod \; N^2 \\ &= g^{m_1+m_2} \cdot (r_1 \cdot r_2)^N \; mod \; N^2 \\ &= E_{P_k}(m_1 + m_2) \; . \end{aligned}$$

Visto isso, Paillier é um sistema de criptografia homomórfica aditiva [33].

2.3.4 Sistema de Criptografia de ElGamal

Esse sistema se baseia no problema do logaritmo discreto e é descrito a seguir:

- *Inicialização*: um número primo grande q é escolhido. Em seguida, é escolhido um gerador g do grupo cíclico $\mathbb{Z}_q^*$.
- *Geração de chaves*: uma chave privada x é igual a um número aleatório de forma que $x \in_R \mathbb{Z}_q^*$. A chave pública correspondente é $y = g^x$;
- *Criptografia*: uma mensagem $m \in G$ é criptografada sob a chave y ao escolher um número aleatório $r \in_R \mathbb{Z}_{N-1}^*$ e computar $c = g^r$ e $d = m \cdot y^r$, logo, $E_y(m) = (c, d)$.
- *Descriptografia*: um cifrotexto $E_y(m)$ é descriptografado ao utilizar a chave privada x para calcular $m = d \cdot c^{-x}$.

Dadas as mensagens m_1 e m_2, pode-se obter uma criptografia de $m_1 \cdot m_2$ ao computar:

$$\begin{aligned} E_y(m_1) \cdot E_y(m_2) &= (c1 \cdot c2, d1 \cdot d2) \\ &= (g^{m_1+m_2}, m1 \cdot m2 \cdot y^{r_1+r+2}) \\ &= E_y(m_1 \cdot m_2)\ . \end{aligned}$$

Logo, ElGamal é um sistema de criptografia homomórfica multiplicativa [33].

2.3.5 Tecnologias de Hardware para Segurança

Esta seção apresenta tecnologias de hardware para promover segurança ao processar e/ou armazenar dados. Para o desenvolvimento desta tese, apenas processadores Intel foram considerados. No entanto, para fins de contextualização, outras tecnologias semelhantes são discutidas a seguir.

AMD *Secure Memory Encryption* (SME)

AMD *Secure Memory Encryption* é uma tecnologia para criptografia dos dados antes do seu armazenamento na memória e descriptografia dos mesmos ao recuperá-los. Isso é possível com o uso de um microprocessador dedicado – AMD Secure Processor – para realizar tais

operações. O gerenciamento de chaves é feito exclusivamente pelo mesmo e uma nova chave é criada a cada vez que o sistema é reiniciado.

Existe um baixo custo adicional pela segurança imposta nos dados, mas esse custo pode ser reduzido caso a memória seja parcialmente criptografada – por exemplo, separando páginas específicas para os dados críticos [35].

AMD *Secure Encrypted Virtualization* (SEV)

Secure Encrypted Virtualization[2] integra recursos de criptografia de memória principal (com o uso da tecnologia AMD SME) com a arquitetura de virtualização AMD-V para suportar máquinas virtuais criptografadas. Isso impede que o hipervisor tenha acesso ao conteúdo da memória das máquinas virtuais provisionadas, ou seja, não é necessário confiar no hipervisor e nos responsáveis pelo *host*. Por fim, essa tecnologia, assim como AMD SME, não exige adaptações ou mudanças nas aplicações que serão executadas nessa plataforma.

Cada máquina virtual, bem como o hipervisor, está associada a uma *tag* e, consequentemente, uma chave de criptografia. Com isso, os dados são restritos apenas à máquina virtual usando essa *tag*. Se esses dados forem acessados por qualquer outra entidade, incluindo o hipervisor, eles só poderão ver os dados em sua forma criptografada, proporcionando um isolamento dos dados [35].

Para garantir que a tecnologia esteja sendo realmente utilizada, o *firmware* auxilia no cumprimento de três propriedades: (i) **autenticidade da plataforma** – uma chave é assinada pela AMD e pelos donos do *host* para mostrar quem são os administradores da plataforma; (ii) **confidencialidade dos dados da máquina virtual** – somente o firmware conhece a chave de criptografia, e, (iii) **atestação de uma máquina virtual** – uma assinatura de vários componentes é fornecida pelo firmware – inclusive estado inicial da memória – para validação por parte do responsável pela máquina virtual.

ARM TrustZone

TrustZone é um conjunto de extensões de segurança disponível em processadores ARM11, Cortex A8, entre outros além de microcontroladores. Nessa tecnologia, há separação entre os

[2] Até o momento da escrita desse documento, somente os processadores AMD EPYC – exclusivos para servidores – possuem suporte a AMD SME e SEV.

ambientes não-seguro e seguro. Quando o processador executa instruções, ele entra em um estado não-seguro, em que só pode acessar recursos desse ambiente. Quando as instruções são executadas em um estado seguro, o processador pode acessar recursos dos dois ambientes [36]. É possível acessar o ambiente seguro a partir de chamadas a instruções de chamada segura (SMC) – o estado do ambiente atual é salvo para uso posterior.

Para viabilizar um ambiente seguro, é necessário que um software confiável (*Trusted OS*) seja desenvolvido para fazer uso dos recursos protegidos. Um ambiente de execução confiável (*Trusted Execution Environment* – TEE) é composto por hardware com suporte a TrustZone, *Trusted OS* e boot confiável (*Trusted Boot*). O TEE oferece as propriedades de segurança de confidencialidade e integridade para vários aplicativos confiáveis[3].

Diversos componentes foram projetados para impor restrições de segurança, preservando o baixo consumo de energia. A conexão AMBA3 – *Advanced Microcontroller Bus Architecture* versão 3 – entre AXI e APB permite a comunicação segura entre o processador e periféricos. AXI – *Advanced eXtensible Interface*, que é o barramento principal do sistema, contém um bit que indica se uma operação de leitura / gravação é direcionada para a memória segura ou não-segura. O barramento APB – *Advanced Peripheral Bus*, contém uma baixa largura de banda para reduzir o consumo de energia. O componente *TrustZone Address Space Controller* (TZASC) permite a classificação dinâmica de dispositivos mapeados na memória como segura ou não-segura e suporta a criação de um número arbitrário de partições [37].

A inicialização segura de ambas as partes deste sistema é uma preocupação fundamental. Sem a devida verificação de ambas as imagens, o dispositivo pode inadvertidamente inicializar uma versão maliciosa, dando aos invasores uma rota de entrada. Portanto, a tecnologia *TrustZone* suporta sequências de inicialização seguras [37].

Intel Software Guard eXtensions (SGX)

Intel SGX é uma tecnologia que tem como objetivo implementar um ambiente de execução confiável (TEE). Ela busca garantir proteção contra modificação e divulgação de dados e código de aplicações. Tal proteção é implementada em nível de hardware, onde o processador é responsável pelo controle de acesso a áreas de memória protegidas criptograficamente. Tentativas de acesso ou modificação não autorizadas são tratadas como falha ou acesso a

[3] `https://www.arm.com/products/security-on-arm/trustzone`

uma memória inexistente.

No jargão de Intel SGX, **enclaves** são as áreas de memória protegidas criptograficamente contra acesso e modificação. Enclaves SGX devem ser utilizados para processar dados sensíveis de forma segura. Durante o processo de instanciação de um enclave, todo o código e dados que o compõem são utilizados para gerar uma identidade criptográfica, **MRENCLAVE**, através do uso de uma função de *hashing*. Essa identidade pode ser depois utilizada para verificar a autenticidade do enclave.

Atestação Remota é o processo de provar que um determinado programa foi devidamente instanciado em uma plataforma. No caso do Intel SGX, o processo de atestação garante que um programa está sendo executado dentro de um enclave SGX, e que não foram feitas modificações no código do mesmo. O processo de atestação pode ser feito local ou remotamente. No caso da arquitetura em questão, interessa-nos particularmente o processo de atestação remota.

O processo de atestação remota é baseado no processo de troca de chave simétrica com o protocolo *Diffie-Hellman*, com a adição de uma assinatura digital, **QUOTE**, na última mensagem do protocolo, produzida com uma chave, **EPID**, acessível apenas a partir de enclaves SGX. O objetivo do **QUOTE** é provar que a última mensagem do processo de troca de chaves foi gerada dentro de um enclave SGX. Essa prova pode ser verificada através de um serviço de atestação disponibilizado pela Intel, o **IAS** – *Intel Attestation Service*. Parte do conteúdo do QUOTE é o MRENCLAVE do enclave que o gerou. Uma entidade pode obter segurança de que está se comunicando com uma aplicação segura ao verificar o **QUOTE** recebido de tal aplicação com o **IAS**, e, tendo uma verificação positiva, checando se o **MRENCLAVE** corresponde ao valor esperado.

2.4 Arquiteturas de nuvem seguras

Nos últimos anos, a computação na nuvem desenvolveu-se em diversos aspectos [38]. Os aspectos de segurança estão entre as principais contribuições e ainda está em constante desenvolvimento [39]. Um crescente número de dispositivos estão conectados à nuvem e com o surgimento da Internet das Coisas, esses números cresceram ainda mais [40].

Soluções que garantem a privacidade estão entre as mais demandadas. TIAN et al. [41],

por exemplo, propõem uma arquitetura que promove a privacidade dos dados de localização do usuário num sistema de *cloud-of-things*, ou **nuvem das coisas**, numa tradução livre.

Da perspectiva dos usuários da nuvem, a segurança é a grande preocupação que dificulta a adoção da nuvem como modelo de computação [42] porque:

- As empresas terceirizam o gerenciamento de segurança (e portanto não possuem controle);
- Coexistência de vários usuários na mesma localização e usando a mesma instância do serviço sem qualquer conhecimento sobre o nível de segurança adotado;
- A falta de garantias de segurança nos contratos de serviço (SLAs) entre os usuários da nuvem e os provedores da nuvem;
- Hospedar conjuntos de atributos valiosos em recursos publicamente disponíveis de infraestrutura aumenta a probabilidade de ataques.

Do ponto de vista dos provedores da nuvem, a segurança exige muitos gastos (licenças de soluções de segurança), recursos (a segurança é uma tarefa que consome recursos) e é uma tarefa difícil para dominar. Nesse sentido, arquiteturas de software são propostas para os mais diversos cenários: *smart grids* [43; 44], *home energy management* [45], *pagamentos* [46], monitoramento de atividade humana [47], dentre outros.

Em paralelo a isso, tecnologias de segurança como Intel SGX e ARM TrustZone, trazem um novo *benchmark* de segurança na nuvem. Nesse sentido, arquiteturas [48] para serviços de sistemas confiáveis são propostas. Soluções que se utilizam dessas tecnologias somadas às de software proporcionam um nível superior de segurança, pois além das considerações de privacidade e atestação, a integridade da informação também pode ser garantida.

SAMPAIO et al. [49] propõem um sistema cujo objetivo é habilitar um ecossistema de aplicativos que use plataformas confiáveis prontas para uso – neste caso, Intel SGX, para que os usuários possam permitir ou não que terceiros acessem o fluxo de dados, em tempo real, com um nível de sensibilidade específico. Além disso, essa abordagem não exige que os usuários gerenciem as chaves de criptografia diretamente.

2.5 Arquiteturas *publish-subcribe* seguras

Publish-subscribe é um padrão de troca de **mensagens** que permite o desacoplamento entre **produtores** e **consumidores** de mensagens. Esse padrão também permite alta **escalabilidade**, devido à natureza de distribuída dos produtores e consumidores e pelo processamento assíncrono. Por essas características, ele é útil para aplicações com vários participantes, alto volume e geração contínua de dados, sendo adequado para processamento de dados de dispositivos IoT e outras aplicações orientadas a eventos.

O *publish-subscribe* foca na filtragem de eventos, roteamento eficiente de eventos, semântica de entrega e composição de eventos [50]. Uma abordagem simplista de segurança seria executar todo o sistema com uma infraestrutura de rede segura (por exemplo, utilizando SSL – *Secure Socket Layer*). No entanto, isso não permite que a política de controle de acesso seja expressa em uma base de assinatura ou publicação e não há tratamento diferenciado de vários nós na rede, assumindo confiança total. Não seria prático para aplicativos em escala de Internet com nós de confiabilidade e confiança variadas. Além disso, essa abordagem não permite que a política que rege os privilégios dos nós evolua com base na experiência. Por isso, BELOKOSZTOLSZKI et al. [50] propõem um sistema baseado em funções para controle de acesso.

Já PIRES et al. [51] apresentam o SCBR (*Secure Content-Based Routing*): um sistema de publicação-assinatura baseado em conteúdo que explora os ambientes de execução confiáveis (TEEs), como o Intel SGX, para garantir a confidencialidade e a integridade dos dados, bem como o anonimato e a privacidade dos editores e assinantes. O sistema é baseado em Python e containers. De forma complementar, ARNAUTONOV et al. apresentam o PUBSUB-SGX [52], que em adição à inclusão dos estágios de filtragem e roteamento nos enclaves, inclui mecanismos de verificação para os produtores e consumidores.

Nesses casos, a arquitetura está condicionada ao processador e ao tipo de nuvem (pública, privada, híbrida) utilizada. Na literatura, é possível encontrar propostas de métricas quantificáveis para avaliação de arquiteturas de nuvem.

Por exemplo, SUN et al. [53] fornecem um sistema de avaliação de segurança quantificável para diferentes nuvens que podem ser acessadas por uma API consistente. O sistema de avaliação inclui mecanismos de verificação de segurança, mecanismos de recuperação de

segurança, modelo de avaliação quantificável de segurança, módulo de exibição visual, etc. O modelo de avaliação de segurança compõe um conjunto de elementos de avaliação correspondentes a campos diferentes, como computação, armazenamento, rede, manutenção e etc. Cada elemento é atribuído a uma tupla de três vulnerabilidades, pontuação e método de reparo.

LIANG et al. [54] propõem uma nova abordagem denominada SMK para fornecer serviços de sistema confiáveis para enclaves do SGX de dados na nuvem. O SMK aproveita os recursos de arquitetura Intel existentes, ou seja, o modo de gerenciamento do sistema (*System Management Model* – SMM) e a UEFI (*Uniform Extensible Firmware Interface*).

Embora o uso de TEEs esteja se popularizando, ainda existem poucos trabalhos que apoiam a migração de aplicações orientadas a eventos para ambientes seguros. Enquanto trabalhos como PUBSUBSGX e SCBR implementam sistemas *publish-subscribe* completos, neste trabalho, é proposto um modelo arquitetural que permite o reuso de soluções consolidadas e sem a dependência de TEEs específicos.

Capítulo 3

Modelo Arquitetural

O modelo arquitetural se baseia no padrão de projeto *publish-subscribe*. Nesse padrão, os componentes que enviam mensagens são chamados de *publishers* – eventualmente referenciados como **produtores** no decorrer desse documento. Por sua vez, os **consumidores** (*subscribers*) se registram para receber mensagens. Para reduzir o acoplamento e aumentar escalabilidade, um produtor não se conecta diretamente aos consumidores, já que as mensagens são trocadas por meio de **um ou mais barramentos**.

Esse modelo pode ser aplicado em situações em que dados confidenciais são coletados e devem ser agregados para anonimizar e produzir informações significativas. Por exemplo, nas eleições, dispositivos eletrônicos podem ser usados para enviar, periodicamente, resultados parciais para agregadores na nuvem, a fim de calcular o resultado final da votação. As identidades e resultados parciais dos eleitores não vazariam porque os agregadores seguros lidariam com essas informações em um ambiente isolado.

No modelo proposto, ilustrado na Figura 3.1, **IComponent** pode ser um produtor e/ou consumidor conectado ao barramento **IBus**. Quando há necessidade de conversão do formato de mensagens, um **IGateway** pode ser utilizado. Devido aos requisitos de confidencialidade e integridade, introduziu-se o conceito de um barramento seguro **ISecureBus**, que, além de gerenciar a troca de mensagens, possui mecanismos de autenticação, para garantir a identidade dos componentes registrados ao enviar e receber de mensagens. A autenticação deve ser implementada pelo componente **IPKI**, também responsável pela criação e revogação de credenciais de acesso.

As credenciais de acesso (**ICredentials**) são fornecidas às entidades (**IEntity**), que são

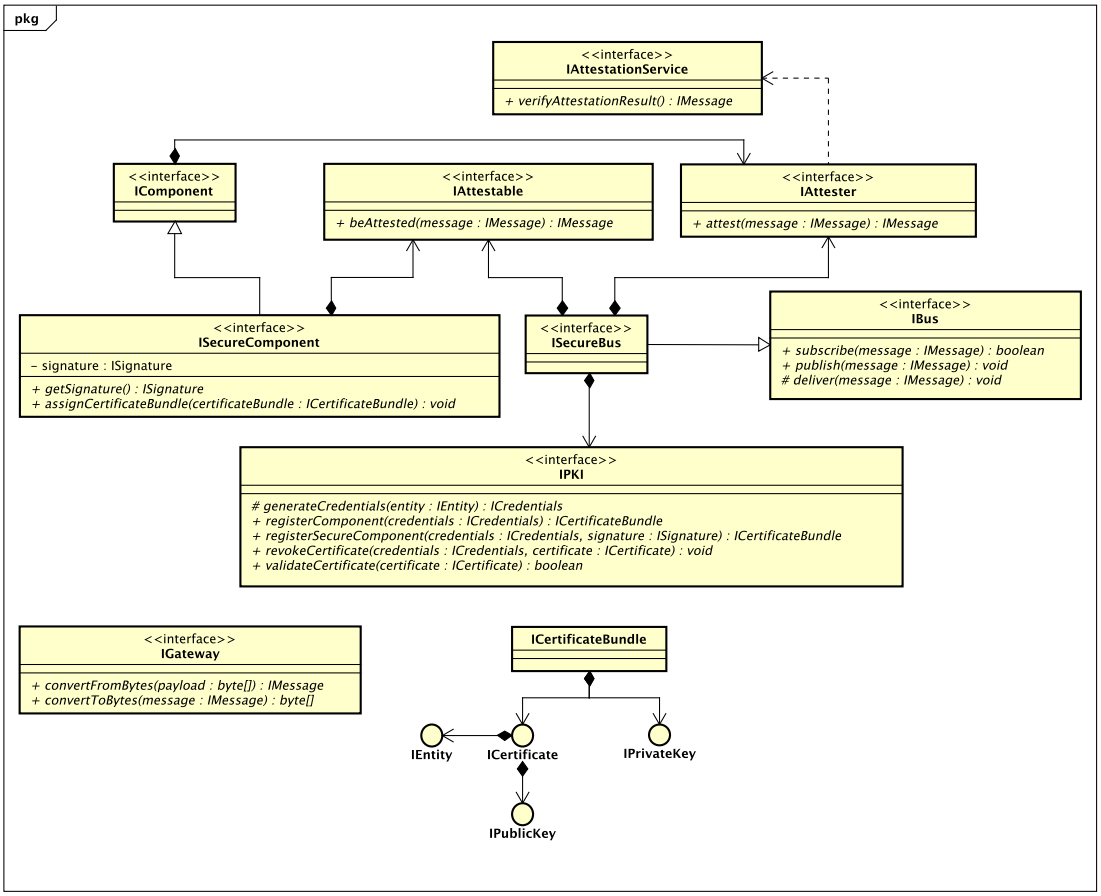

Figura 3.1: Diagrama UML do modelo arquitetural proposto

responsáveis pelo desenvolvimento de componentes a serem conectados no barramento seguro. Para o registro de componentes no barramento, é necessário informar as credenciais. Esse registro resulta em um **certificado** (**ICertificateBundle**), composto pelas chaves pública e privada, além de conter uma validade. Esse certificado será utilizado para **assinar** mensagens enviadas ao barramento seguro.

Componentes seguros devem, obrigatoriamente, ser atestáveis (**IAttestable**). Em outras palavras, devem existir maneiras de garantir que o componente possua a mesma implementação do momento da sua concepção. O barramento seguro é responsável pela atestação desses componentes no momento do seu registro e/ou periodicamente – com o uso do componente **IAttester**. Com o objetivo de validar a atestação, uma consulta é efetuada a um serviço de atestação – representado pelo componente **IAttestationService**. Nesse modelo, é necessário que o serviço de atestação seja **confiável**.

Uma versão simplificada da arquitetura descrita está ilustrada na Figura 3.2. Nela, é

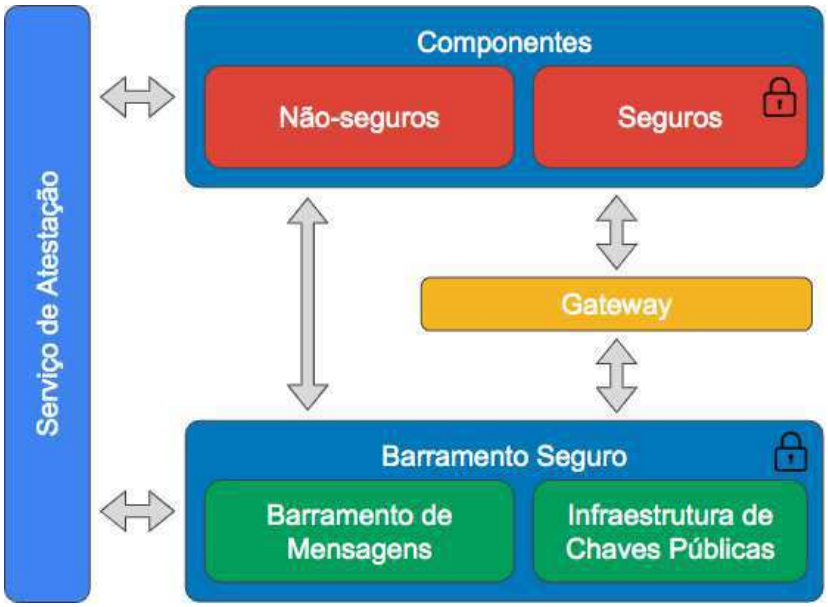

Figura 3.2: Diagrama simplificado do modelo arquitetural

possível ver as conexões entre os componentes. Em suma, é possível que os componentes se conectem diretamente ao barramento ou por meio de um ***Gateway***. Além disso, o serviço de atestação é consultado para validar as atestações.

O padrão *publish-subscribe* tem como foco alto desempenho e escalabilidade, mas não provê confidencialidade e integridade por padrão [55]. Visto isso, o modelo arquitetural proposto contém os seguintes componentes adicionais para contemplar requisitos de segurança: **ICredentials**, **IAttester**, **IAttestable**, **IAttestationService**, **IPKI**, **ICertificateBundle**, **IPrivateKey** e **IPublicKey**.

A seguir, os requisitos funcionais e não-funcionais serão explicitados, assim como as decisões, elementos arquiteturais e cenários relevantes. Por fim, instruções de uso para esse modelo serão apresentados.

3.1 Requisitos Funcionais

3.1.1 Gerar Credenciais

Descrição:	Para permitir o registro de novos componentes, seguros ou não, no barramento, é necessário que a entidade mantenedora – um desenvolvedor ou uma empresa, por exemplo – possua credenciais de acesso. Tais credenciais devem ser informadas à entidade de forma segura, para evitar vazamento de informações.
Justificativa:	A utilização de credenciais auxilia na confidencialidade dos dados, pois limita quem terá acesso a quais dados no barramento.
Vantagens:	Redução da chance de vazamento de informações por acessos indevidos.
Desvantagens:	Por si só, não impedem completamente o risco de vazamento de dados, pois uma entidade maliciosa pode fazer um uso indevido das informações coletadas. Para amenizar esse problema, o processo de atestação é recomendado.

3.1.2 Bloquear e Desbloquear Credenciais

Descrição:	Permite o bloqueio temporário das credenciais de uma entidade, impedindo que os componentes mantidos por ela acessem o barramento.
Justificativa:	Entidades com componentes suspeitos podem ter seu acesso bloqueado para averiguação por parte dos administradores de segurança.
Vantagens:	Como é um processo reversível, a comunicação dos componentes pode ser restabelecida de forma rápida.
Desvantagens:	Dependendo da implementação, o bloqueio pode ser efetivado somente em novas autenticações.

3.1.3 Remover Credenciais

Descrição:	É possível remover credenciais para desabilitar o acesso de todos os componentes de uma entidade, de forma permanente.
Justificativa:	Componentes que não participam mais do processo de coleta de dados devem ser removidos para evitar novos acessos indevidos.
Vantagens:	Reduz o acesso indevido aos dados, contribuindo com a confidencialidade.
Desvantagens:	Nenhuma identificada.

3.1.4 Registrar Componentes

Descrição:	Para o registro de componentes, deve-se informar as credenciais de acesso da entidade. Em seguida, o **barramento seguro** valida os dados informados e, caso positivo, retorna um **certificado** para ser associado ao componente.
Justificativa:	Uma entidade pode ter um ou mais componentes, de forma que cada um pode ter acessos a diferentes tipos de informações.
Vantagens:	Garante mais controle no nível de acesso aos dados por componente.
Desvantagens:	Não impede completamente o vazamento de dados, logo, componentes comuns não devem ter acesso a informações sensíveis.

3.1.5 Registrar Componentes Seguros

Descrição:	Para um componente seguro, deve-se informar, além das credenciais, a sua assinatura. Essa assinatura pode ser associada ao desenvolvedor ou à versão específica do código que é executado por esse componente.
Justificativa:	Os componentes seguros lidam com dados mais sensíveis, logo, é necessária uma proteção adicional para evitar o uso indevido dos dados.
Vantagens:	Possibilita o acesso de dados sensíveis somente a componentes com técnicas de segurança adequadas.
Desvantagens:	Assim como as credenciais, não impede o uso indevido dos dados sensíveis, logo, deve ser utilizado em conjunto com processos de atestação.

3.1.6 Revogar Certificados

Descrição:	Além de gerar, há casos em que um certificado seja revogado, por motivos de desativação de componentes, por exemplo. Desse modo, deve-se ignorar mensagens enviadas ao barramento que são assinadas por um certificado revogado.
Justificativa:	É necessário ter um mecanismo para evitar novas mensagens de componentes desatualizados ou descontinuados.
Vantagens:	Reduz o acesso indevido aos dados, contribuindo com a confidencialidade.
Desvantagens:	Nenhuma identificada.

3.1.7 Validar Certificados

Descrição:	Para identificar se uma mensagem do barramento deve ser aceita ou não, deve-se validar o certificado que a assinou. Caso ela não seja válida, o barramento irá recusar a mensagem. Deve-se verificar se o certificado se encontra no banco de dados, se a sua data está no período de validade e se o mesmo não fora revogado.
Justificativa:	É necessário prover uma maneira para impedir que componentes com acesso revogado ou vencido acessem o barramento.
Vantagens:	Reduz o acesso indevido aos dados, contribuindo com a confidencialidade.
Desvantagens:	Aumento do custo computacional e da latência, pois a validação deve ser feita a cada mensagem produzida.

3.1.8 Recuperar Assinatura

Descrição:	Componentes seguros devem possuir assinaturas, que serão informadas ao barramento seguro, juntamente com o certificado, para autenticar esse componente e permitir o seu registro.
Justificativa:	É necessário prover uma forma de validação da autenticidade de cada componente seguro.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da complexidade na construção do componente.

3.1.9 Associar Certificado

Descrição:	Como certificados têm validade e podem ser trocados dinamicamente, os componentes seguros devem permitir sua troca, mediante atestação do código.
Justificativa:	O tempo de indisponibilidade do componente deve ser o menor possível.
Vantagens:	Reduz a complexidade de manutenção do componente.
Desvantagens:	Nenhuma identificada.

3.1.10 Realizar Inscrição

Descrição:	Componentes podem se inscrever em **tópicos** do **barramento** para receber mensagens. Visto isso, a mensagem de inscrição enviada deve ser assinada com o seu certificado. Para componentes seguros, sua assinatura também deve compor a mensagem. No ato dessa inscrição, a assinatura e o certificado devem ser validados.
Justificativa:	Somente os componentes com as credenciais, certificados e assinaturas (para componentes seguros) podem ter acesso ao barramento.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da complexidade na construção do componente.

3.1.11 Publicar Mensagem

Descrição:	O barramento de mensagens deve permitir que mensagens sejam publicadas para que sejam entregues posteriormente. Somente mensagens geradas por componentes já registrados terão sua entrega realizada.
Justificativa:	Assim como na inscrição, a publicação não deve permitir acessos indevidos.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da complexidade na construção do componente.

3.1.12 Entregar Mensagem

Descrição:	Antes da entrega de uma mensagem, verifica-se se ela possui um certificado válido. Se possuir, deve-se entregar a mensagem para cada componente válido que está inscrito no tópico no qual ela é endereçada. Caso contrário, ela é descartada.
Justificativa:	As mensagens não podem ser entregues a participantes não autorizados.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da complexidade na construção do componente.

3.1.13 Iniciar Atestação

Descrição:	Antes de se inscrever em um tópico, componentes podem atestar o barramento, para se certificar de que o código em execução é o esperado. Quando os componentes são seguros, o barramento também pode fazer a atestação deles, formando uma **atestação de duas vias**.
Justificativa:	A atestação permite validar se os componentes realmente são quem dizem ser.
Vantagens:	Provê garantias de confidencialidade e integridade.
Desvantagens:	Aumento do tempo de inscrição de componentes no barramento.

3.1.14 Validar Atestação

Descrição:	Deve existir uma entidade confiável na qual o resultado da atestação, como por exemplo, um **token**, será enviado e validado.
Justificativa:	É necessário garantir que a atestação realizada não foi adulterada.
Vantagens:	Aumento da confidencialidade e integridade.
Desvantagens:	Aumento do tempo de inscrição de componentes no barramento.

3.2 Requisitos Não-Funcionais

3.2.1 Várias Instâncias em Execução

Atributos:	Tolerância a falhas e escalabilidade.
Descrição:	Todos os componentes na arquitetura devem permitir uma ou mais instâncias em paralelo, a fim de garantir maior capacidade de processamento e redução de pontos únicos de falha.
Justificativa:	Em aplicações orientadas a eventos, o número de dispositivos e/ou a quantidade de dados gerados pode ser alta, logo, é necessário que a arquitetura seja capaz de permitir esses cenários.
Vantagens:	Reduz as chances de indisponibilidade do sistema e permite que as taxas de entrada cresçam sem afetar negativamente o tempo de resposta.
Desvantagens:	Maior complexidade para implementação e manutenção do sistema.

3.2.2 Acesso Restrito

Atributos:	Autenticidade e confidencialidade.
Descrição:	Todas as operações do sistema devem ser acessíveis somente após autenticação.
Justificativa:	O sistema não deve ser acessado diretamente sem que haja uma autenticação prévia.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da complexidade na construção dos componentes.

3.2.3 Bloqueio de acessos inválidos

Atributos:	Confidencialidade e disponibilidade.
Descrição:	Caso existam várias tentativas inválidas de acesso dentro de um intervalo estabelecido, o usuário e IP de origem devem ser bloqueados.
Justificativa:	O sistema não deve ser suscetível a ataques de força bruta.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Nenhuma identificada.

3.2.4 Comunicação Segura

Atributos:	Confidencialidade e integridade.
Descrição:	Todos os dados trafegados pela rede devem ser encriptados.
Justificativa:	Mesmo que haja interceptação das mensagens trocadas entre componentes, elas não devem estar em texto plano.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da latência e da quantidade de dados trafegados.

3.2.5 Processamento Seguro

Atributos:	Confidencialidade e integridade.
Descrição:	Todos os dados críticos – que possuem dados privados dos seus possuidores – devem ser processados de forma segura, via hardware ou software. Algumas das técnicas de hardware e software aplicáveis podem ser vistas na seção 2.3.
Justificativa:	O processamento seguro deve permitir que o componente seja executável mesmo em ambientes não confiáveis.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da carga de processamento dos dados.

3.2.6 Armazenamento Seguro

Atributos:	Confidencialidade e integridade.
Descrição:	Dados críticos devem ser armazenados, em disco, de forma segura. Pode-se criptografar de forma que apenas o uso combinado de várias chaves possa revelá-los.
Justificativa:	Mesmo que haja acesso aos dados armazenados pelos componentes, eles não devem estar em texto plano.
Vantagens:	Reduz a possibilidade de vazamento dos dados.
Desvantagens:	Aumento da carga de processamento ao recuperar os dados.

3.3 Decisões Arquiteturais

3.3.1 Chaves de acesso devem possuir atributos

As chaves de acesso podem conter atributos como validade, nível de acesso e outros. O nível define os dados que o possuidor da chave poderá acessar. Isso permite maior flexibilidade mas torna a implementação mais complexa.

3.3.2 Realizar atestação antes de troca e processamento de dados

Para que o envio e processamento de dados coletados seja seguro, é necessário que exista uma atestação remota para garantir que a comunicação não seja interceptada até que chegue no local seguro. Nesse local, os dados serão desencriptados, processados e, possivelmente, encriptados novamente para novas comunicações.

3.3.3 Criptografar dados com chave negociada no processo de atestação

Como resultado da atestação remota, uma chave derivada pode ser obtida para que um canal seguro seja estabelecido. Além do uso de SSL, os dados podem ser criptografados para garantir uma maior segurança ao canal.

3.3.4 Utilizar tecnologias de hardware e software para processamento e armazenamento de dados

Para permitir o processamento e armazenamento seguro de dados, processadores com suporte a Intel SGX, ARM TrustZone, AMD SME ou AMD SEV podem ser utilizados. Mais detalhes sobre essas tecnologias podem ser encontrados no Capítulo 2.

3.4 Elementos Arquiteturais

A arquitetura é composta por: **IMessage**, **IComponent**, **ISecureComponent**, **IAttestable**, **IAttester**, **IAttestationService**, **IBus**, **ISecureBus**, **IPKI**, **IGateway** e **ICertificateBundle**. Eles serão detalhados nas próximas seções, juntamente com suas interfaces.

3.4.1 Mensagens

Para que os componentes se comuniquem, tudo é feito a partir de troca de mensagens – **IMessage**, sejam elas diretas ou indiretas. Uma mensagem possui um conjunto de atributos e ela é, na maioria dos casos, encriptada com alguma chave, seja ela derivada do processo de atestação ou do certificado de quem a assina.

3.4.2 Componentes Seguros & Não-seguros

Um componente dessa arquitetura pode ser **seguro** ou **não-seguro**, e pode ser um **produtor**, **consumidor** ou um **serviço**. Quando o mesmo é seguro, além de um certificado, ele possui uma assinatura, que vai depender da tecnologia aplicada[1]. Com relação a sua natureza, quando ele é um **produtor**, mensagens devem ser enviadas a um tópico do barramento; quando é um **consumidor**, mensagens devem ser consumidas de um tópico e quando é um **serviço**, ele fornece operações para outros componentes da arquitetura. Não há restrições para componentes que possuam múltiplas naturezas, ao contrário da sua segurança, que é excludente.

[1]Por exemplo, no caso de Intel SGX, a assinatura pode ser com relação ao código do software – **MRENCLAVE** ou com relação ao desenvolvedor – **MRSIGNER**

Operações

As operações abaixo são obrigatórias, porém, cada componente pode ter operações a mais para, por exemplo, fornecer serviços.

- **ISecureComponent.getSignature**: ver Seção 3.1.8;
- **ISecureComponent.assignCertificateBundle**: ver Seção 3.1.9.

3.4.3 Componentes para Atestação

Um componente do tipo **IAttester** pode realizar atestação de outros componentes do tipo **IAttestable**. O processo de atestação é validado com o uso de um serviço de atestação, representado pelo **IAttestationService**. No caso do Intel SGX, esse componente é implementado pelo **Intel Attestation Service**. Ademais, um componente **IAttester** também pode ser utilizado quando o dispositivo que necessita realizar essa operação não possua capacidade, seja ela de hardware ou software.

Operações

- **IAttestable.beAttested**: o componente **IAttester** monta mensagens para que esse componente as consuma e envie uma resposta. Esse processo pode ter mais de uma etapa, que é identificada por atributos da mensagem;
- **IAttester.attest**: envio de desafio para o componente **IAttestable**. Esse desafio pode conter parâmetros adicionais, que serão enviados por quem está iniciando a atestação.
- **IAttestationService.verifyAttestationResult**: valida o processo de atestação. Essa entidade deve ser confiável por quem inicia o processo de atestação.

3.4.4 Barramentos

Um **barramento – IBus** – é responsável por aceitar inscrições de consumidores, aceitar envio de mensagens por produtores e por encaminhá-las para os interessados. Nessa arquitetura, o diferencial se encontra no **barramento seguro – ISecureBus**, que além das operações

citadas, também contém o componente **IPKI**, responsável por criar e revogar certificados e credenciais, autenticar, atestar consumidores e validar mensagens.

Operações

- **IBus.subscribe**: ver seção 3.1.10;
- **IBus.publish**: ver seção 3.1.11.
- **IBus.deliver**: ver seção 3.1.12.

3.4.5 Infraestrutura de Chaves Públicas

O componente **IPKI** é responsável pela Infraestrutura de Chaves Públicas (*Public Key Infrastructure* – PKI). Ele é responsável pela geração, bloqueio e desbloqueio de credenciais; registro de componentes seguros e não-seguros; revogação e validação de certificados.

Um **ICertificateBundle** contém uma chave privada (**IPrivateKey**) e um certificado (**ICertificate**), que possui uma validade e uma chave pública (**IPublicKey**). Esse componente é associado a uma entidade, que pode ser um desenvolvedor, ou uma empresa, por exemplo.

Operações

- **IPKI.generateCredentials**: ver Seção 3.1.1;
- **IPKI.lockCredentials**: ver Seção 3.1.2;
- **IPKI.unlockCredentials**: ver Seção 3.1.2;
- **IPKI.registerComponent**: ver Seção 3.1.4;
- **IPKI.registerSecureComponent**: ver Seção 3.1.5;
- **IPKI.revokeCertificate**: ver Seção 3.1.6;
- **IPKI.validateCertificate**: ver Seção 3.1.7;

3.4.6 Gateways

Quando for necessária a conversão dos formatos de dados, é possível utilizar um componente do tipo **IGateway** para efetuar essa operação. Ele também pode ser utilizado para agrupar e enviar conjuntos de mensagens de vários produtores.

Operações

Essas operações são exemplos de como um **gateway** pode ser implementado.

- **convertFromBytes**: a partir de um conjunto de bytes, uma **mensagem** é criada;
- **convertToBytes**: a partir de uma mensagem, um *array* de bytes é criado.

3.5 Cenários Relevantes

A Figura 3.3 ilustra os cenários relevantes do modelo arquitetural: (i) **Registro de Entidades**; (ii) **Registro de Componentes no Barramento**; (iii) **Atestação Remota** e (iv) **Produção e Consumo de Mensagens**. Nas subseções a seguir, cada um desses cenários é detalhado.

3.5.1 Cenários de Registro

Os dois primeiros cenários ocorrem sempre que uma nova **entidade** deseja acesso ao barramento ou quando um novo **componente** é criado.

Registro de Entidades

Esse cenário inicia com uma entidade (**IEntity**) que deseja produzir ou consumir dados do barramento. Para isso, a entidade deve possuir os dados para registro na Infraestrutura de Chaves Públicas (**IPKI**). A forma de passagem dos dados de registro e das credenciais (**ICredentials**) criadas vai depender do problema em que esse modelo arquitetural é aplicado, podendo ser presencialmente ou remotamente, por exemplo.

Como dito anteriormente, as credenciais criadas são necessárias para que componentes seguros e não-seguros sejam registrados no barramento seguro (**ISecureBus**) e devem ser

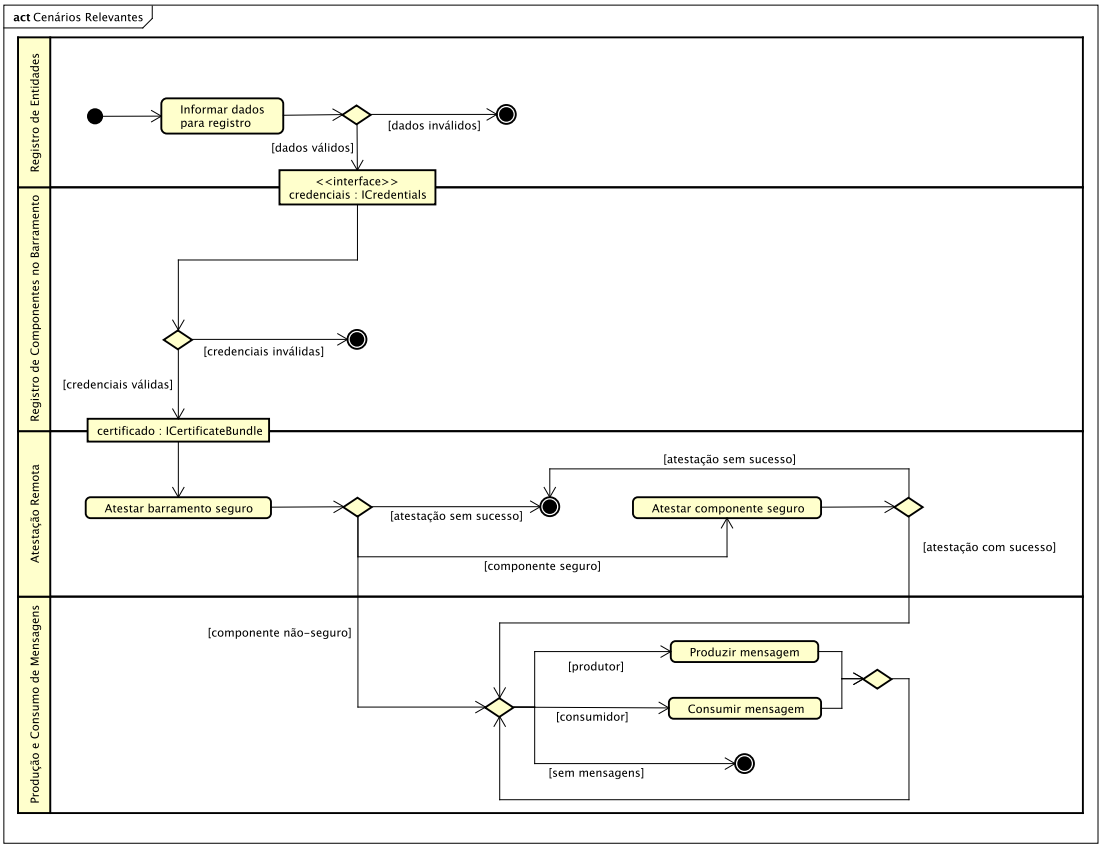

Figura 3.3: Diagrama com o fluxo de atividades dos componentes do modelo arquitetural

armazenadas pela entidade em um local seguro. Já a Infraestrutura de Chaves Públicas é responsável por armazenar os dados necessários para conseguir validá-las, de forma que as credenciais puras não serão armazenadas.

Registro de Componentes no Barramento

Uma vez que a entidade possui credenciais de acesso, ela pode registrar componentes seguros e não-seguros para produzir e consumir do barramento. Assim como na subseção anterior, esse registro pode ser feito presencial ou remotamente, o que depende dos requisitos do problema a ser resolvido. Feito esse registro e de posse do certificado (**ICertificateBundle**), a entidade deve armazená-lo de forma segura e associá-lo ao componente registrado para que o mesmo possa gerar mensagens reconhecíveis pelo barramento.

A Infraestrutura de Chaves Públicas é responsável por armazenar as **chaves públicas** e **assinaturas** – no caso de componentes seguros – que participaram desse processo. As

assinaturas são armazenadas para permitir atestações futuras e as chaves privadas ficam somente em posse das entidades.

3.5.2 Cenários de Processamento de Dados

Os dois cenários a seguir ocorrem sempre que uma nova instância de um componente é criada – seja ele seguro ou não-seguro. Eles ocorrem com uma frequência maior que os cenários de registro.

Atestação Remota

Esse cenário começa com o envio de uma mensagem de atestação para o **barramento seguro** por parte do **componente**. Essa mensagem é assinada com a chave privada que fora obtida no registro desse componente. Ao receber a mensagem, o **barramento seguro** valida se ela realmente foi assinada por um componente registrado anteriormente. Após a validação, o **barramento seguro** retorna uma mensagem contendo as informações de atestação para o **componente**, que por sua vez, valida essas informações com o serviço de atestação (**IAttestationService**). O **barramento seguro** também realiza uma atestação quando o componente a se inscrever é **seguro**.

Com o canal estabelecido pelo processo de atestação, chaves adicionais – para encriptar ou desencriptar mensagens – podem ser repassadas de forma segura. Dependendo de como o modelo arquitetural é instanciado pode existir, por exemplo, uma chave para cada componente no barramento, uma chave para cada grupo de componentes ou até mesmo uma chave por tipo de mensagem.

Produção e Consumo de Mensagens

Assim que o processo de atestação é finalizado, o componente se inscreve em **tópicos**, a fim de produzir e consumir mensagens. É válido lembrar que o componente conseguirá acesso apenas às mensagens das quais ele possui a chave, uma vez que os dados armazenados nesses tópicos são criptografados.

A Figura 3.4 possui uma visão detalhada dos cenários citados nas subseções anteriores, ilustrando as operações e componentes envolvidos. Os passos 1, 2 e 3 estão relacionados

com o cenário de **Registro de Componentes no Barramento**, já os passos 4, 4.1, 5, 5.1, 6, 6.1 e 6.2, com o cenário de **Atestação Remota** e os passos 7, 7.1 e 7.2, com o cenário de **Produção e Consumo de Mensagens**.

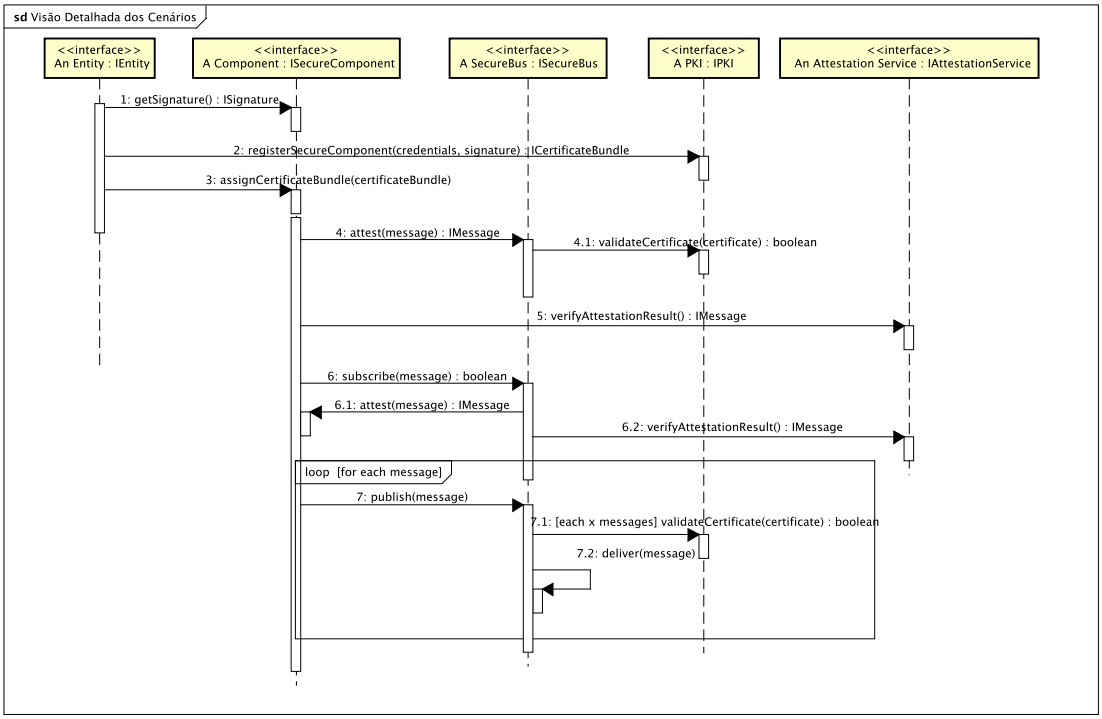

Figura 3.4: Diagrama de sequência dos cenários relevantes

3.6 Instruções de Uso

Após a compreensão do modelo arquitetural – incluindo seus **requisitos funcionais** e **não-funcionais**, **componentes** e **decisões arquiteturais** – é possível **instanciar uma arquitetura de software** para embasar o desenvolvimento de um sistema para processamento seguro de dados. Para tal, é necessário realizar as seguintes atividades:

1. Levantar requisitos;
2. Mapear os componentes do modelo para o problema real;
3. Listar decisões arquiteturais;
4. Realizar implementação com base na arquitetura instanciada.

Recomenda-se que essas atividades sejam realizadas de forma iterativa e incremental. De modo que na necessidade de refinamentos, deve-se retornar a passos anteriores antes de seguir em frente. Além disso, é importante documentar a arquitetura a ser implementada – um modelo para documentação de arquiteturas de software é fornecido por SILVA [18].

Nas seções seguintes, cada uma das atividades será detalhada.

3.6.1 Levantamento de Requisitos

O primeiro passo a ser realizado é o levantamento de requisitos do sistema, para que seja entendido **o que** (requisitos funcionais) o sistema deve fazer e **como** deve fazer (requisitos não-funcionais). O modelo já possui um conjunto de **requisitos**, mas nem todos precisam ser aplicados na **arquitetura instanciada**, então, isso também deve ser endereçado nessa etapa.

O levantamento de requisitos é importante para entender o escopo do problema e deve ser mitigado no começo, pois mudanças posteriores nos requisitos podem ser custosas e até mesmo inviabilizar o projeto.

3.6.2 Mapeamento de Componentes com o Modelo

Uma vez que os requisitos foram definidos, mesmo que parcialmente, deve-se mapear os componentes propostos no modelo para resolver o problema real. Para tal, os componentes instanciados na arquitetura serão baseados nos componentes propostos no modelo.

Assim como os requisitos, nem todos os componentes arquiteturais propostos serão utilizados. Todavia, por ser um modelo baseado em *publish-subscribe*, há componentes que são obrigatórios na arquitetura instanciada. Com isso, **IComponent** e **IBus** estarão nessa arquitetura. Além disso, componentes responsáveis pela segurança no processamento de dados também são necessários: **IAttestable** e **IAttester**.

3.6.3 Listagem de Decisões Arquiteturais

Nessa etapa, deve-se listar decisões arquiteturais – aquelas que impactam em vários pontos do sistema. Nela, as tecnologias de hardware e software devem ser definidas para que os requisitos sejam atendidos. Essas decisões impõem limitações de origem externa (e inerentes ao cenário da aplicação) à solução desejada.

3.6.4 Realizar implementação

Após a definição **do que** deve ser feito, **como** deve ser feito e quais **tecnologias** serão aplicadas, é possível iniciar a **implementação**.

Nas etapas anteriores, é crucial que as tecnologias escolhidas tenham sido validadas e realmente atendem aos requisitos definidos. Uma troca de tecnologia durante a implementação do sistema é bastante custosa. Fora isso, é importante que inspeções periódicas sejam realizadas a fim de garantir que a arquitetura de software ainda é seguida da forma que foi concebida – caso contrário, deve-se corrigir o código ou atualizar a arquitetura.

Capítulo 4

Aplicação I: Consumo de Energia

O crescimento da demanda de recursos de energia elétrica motivou tanto o governo quanto a indústria a buscarem formas alternativas de disponibilizar essa energia e, principalmente, de aperfeiçoar o gerenciamento da rede elétrica. Entretanto, aumentar a eficiência e balancear a malha energética não é uma tarefa trivial [10]. Para isso, uma opção é utilizar medidores inteligentes (*smart meters*) que podem, periodicamente, medir e reportar o consumo energético [56] e, com base nos dados detalhados, recomendações podem ser geradas. Quanto maior o nível de detalhe dos dados, melhor a recomendação e, consequentemente, a economia [57].

No entanto, a medição periódica e detalhada do consumo energético causa preocupações com a privacidade dos consumidores, já que é possível inferir informações pessoais a partir do que é coletado, como tipos de equipamentos na residência ou a presença e o número de moradores [58]. Em casos mais extremos, é possível identificar o canal de televisão sendo assistido [59].

No escopo desse estudo, duas aplicações foram investigadas: (i) consumo em grupos de residências ou regiões – visando o balanceamento energético; (ii) cálculo de consumo energético mensal por consumidor – visando a redução de custos e de erros de leitura. Com o uso de medidores inteligentes torna-se possível a redução ou até mesmo dispensa de visitas físicas de agentes da concessionária para aferição dos medidores nas residências.

Nas subseções a seguir, a visão geral da arquitetura – que é uma instância do modelo proposto no capítulo 3 – e seus requisitos serão apresentados, seguidos pela descrição de cada componente arquitetural, detalhes sobre implementação e, por fim, experimentos realizados

com as abordagens disponíveis.

4.1 Visão Geral

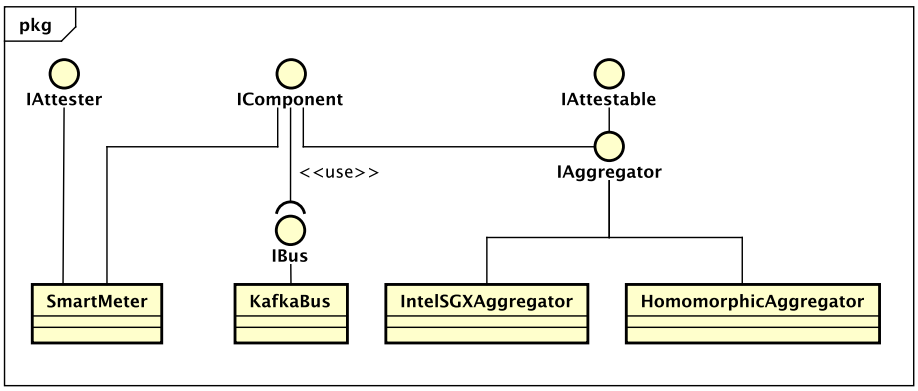

Figura 4.1: Diagrama UML da aplicação para cálculo de consumo de energia

A arquitetura de software, ilustrada pela Figura 4.1, possui os seguintes componentes: barramentos de mensagens (**KafkaBus**), medidores inteligentes (**SmartMeter**), agregadores (**IntelSGXAggregator** e **HomomorphicAggregator**) e consumidores (**IComponent**). Os barramentos de mensagens são responsáveis pela comunicação entre os medidores inteligentes, os agregadores e os consumidores. Após serem produzidos, os dados serão publicados no barramento de mensagens e, em algum momento, serão tratados por agregadores. Estes podem realizar operações arbitrárias, mas devem ser capazes de executá-las de forma segura. Posteriormente, os dados agregados serão consumidos por aplicações.

4.2 Elementos Arquiteturais

A Figura 4.2 contém uma versão simplificada da arquitetura proposta. Os ***Smart Meters*** representam os produtores, o **Apache Kafka** representa o barramento de mensagens e a agregação dos dados pode ser utilizando **Intel SGX** ou **criptografia homomórfica**. Os detalhes serão listados a seguir.

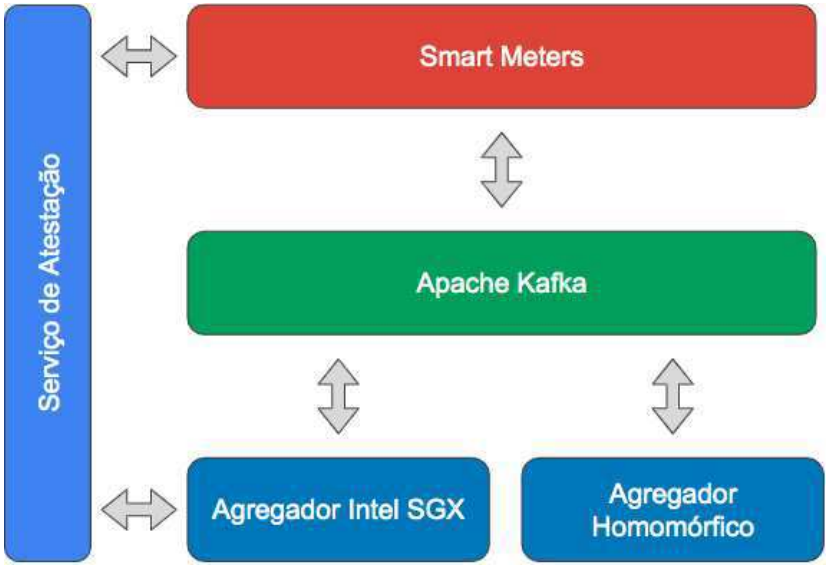

Figura 4.2: Esquema simplificado da aplicação para cálculo de consumo de energia

4.2.1 Barramentos de mensagens

Um barramento de mensagem é responsável pela troca de informações entre produtores, agregadores e consumidores de forma transparente. Um produtor pode criar mensagens sem saber maiores detalhes sobre os agregadores (como localização física ou endereço IP), por exemplo. Com isso, é possível reduzir o acoplamento e garantir escalabilidade.

Cada componente da arquitetura deve se inscrever em um tópico de mensagens. Todos os agregadores ou consumidores de um tópico receberão as mensagens enviadas pelos produtores. Ademais, é permitido o consumo exclusivo de novas mensagens ou de todas as mensagens retidas de um certo tópico. Por fim, a quantidade de mensagens armazenadas para um tópico pode ser configurável.

Por ser crítico na comunicação entre todos os envolvidos, esse componente permite a troca segura de mensagens. Cada envolvido pode possuir um certificado emitido por uma autoridade certificadora que permitirá sua autenticação e evitará que intrusos enviem ou recebam mensagens. Além disso, mensagens com conteúdo sensível devem estar criptografadas. É papel do agregador transformar informação sensível e criptografada – como o consumo energético instantâneo de uma residência – em informação agregada – como o consumo em um intervalo maior de tempo ou o consumo de uma região. .

4.2.2 Agregadores

Após o consumo dos dados de um tópico comum, é necessário realizar operações nos mesmos. Tais operações, como soma, multiplicação ou agrupamento, são realizadas pelos agregadores. Dois agregadores são exemplificados na nossa implementação da arquitetura: um agregador **homomórfico** e um agregador baseado em **Intel SGX**. Ambos são projetados para realizar **operações de soma** e serão detalhados nas seções a seguir.

Agregador Homomórfico

Esse agregador faz uso da criptografia homomórfica para realizar operações de forma segura e privada nos dados. A abordagem utilizada por esse componente é baseada no esquema proposto por BUSOM et al. [60], que faz uso do sistema de criptografia de ElGamal [61]. Tal sistema possui a propriedade homomórfica multiplicativa mas com possibilidade de adição [62].

Para que esse tipo de agregação funcione é necessário que cada produtor $p \in [1, n]$ de um tópico comum possua os seguintes itens:

- Um número primo grande q (de pelo menos 2048 bits) e um gerador g de ordem q do grupo multiplicativo G de $\mathbb{Z}_q^*$;
- Uma chave privada x_p;
- Uma chave pública $y_p = g^{x_p}$;
- Um certificado $cert_p$ a ser validado por uma autoridade certificadora;

Sempre que novos produtores se inscreverem em um tópico de mensagens, uma fase de configuração é obrigatória a fim de permitir o envio das informações. O procedimento da configuração é descrito a seguir:

1. O agregador envia uma mensagem de requerimento de configuração;
2. Cada produtor envia y_p e $cert_p$ ao tópico;
3. O agregador verifica a validade de cada $cert_p$ e insere uma mensagem com $\{y_1, ..., y_n\}$ e $\{cert_1, ..., cert_n\}$ no tópico;

4. Cada produtor verifica a validade de cada $cert_p$ e calcula uma chave pública global $y = \prod_{p=1}^{n} y_p$.

Os passos acima são realizados para garantir que (i) existam somente medidores válidos em uma região – evitando que um atacante forge um medidor e (ii) a chave pública global seja utilizada na computação de valores intermediários, permitindo que o valor agregado seja conhecido no final de cada coleta.

Os seguintes passos serão realizados a cada instante de tempo que for necessária uma coleta de dados para agregação:

1. O agregador envia uma mensagem de requerimento de dados ou, alternativamente, os produtores podem iniciar uma transmissão periodicamente;
2. Cada produtor p gera um número aleatório $z_p \in \mathbb{Z}_q^*$ e calcula $C_p = E_y(g^{v_p+z_p}) = (c_p, d_p)$, em que v_p representa o valor coletado por p e a função E_y é a função de criptografia de ElGamal – utiliza-se a chave pública global y para encriptar valores;
3. Todos os valores C_p são publicados no tópico do barramento de mensagens associado àquele produtor (por exemplo, a região em que o medidor está instalado);
4. O agregador realiza sua computação: $C = \left(\prod_{p=1}^{n} c_p, \prod_{p=1}^{n} d_p\right)$ e insere C no tópico;
5. Cada produtor calcula $T_p = c^{x_p}.g^{z_p}$ e publica-o no tópico;
6. O agregador pode, então, obter $D = d.\left(\prod_{p=1}^{n} T_p\right)^{-1}$, em que $d = \prod_{p=1}^{n} d_p$;
7. Por fim, é possível obter $V = \sum_{p=1}^{n} v_p$ ao calcular $\log_g D$;
8. O agregador publica o resultado obtido no mesmo ou em outro tópico do barramento de mensagens, de forma que ele fique disponível para possíveis consumidores.

Agregador Intel SGX

O agregador Intel SGX faz uso da tecnologia de mesmo nome para prover segurança e privacidade dos dados sensíveis sendo agregados, através do uso de áreas protegidas de

memória (**enclaves**), inacessíveis até mesmo por usuários com mais privilégios. Em nossa implementação usamos o algoritmo de criptografia simétrica *AES Galois/Counter Mode* (AES-GCM) descrito por DWORKIN [63], com chave de tamanho de 128 bits, para a troca de mensagens confidenciais entre os produtores e o agregador. Este algoritmo permite verificar a autenticidade dos dados confidenciais recebidos por meio da detecção de modificações não intencionais ou modificações intencionais não autorizadas feitas aos dados. Além disso, torna a comunicação imune a ataques do tipo *side-channel* baseados em *software*.

As seguintes condições precisam ser satisfeitas para o correto funcionamento deste agregador:

- Cada produtor $p \in [1, n]$ de um tópico deve possuir uma chave simétrica k_p;
- O agregador terá acesso a uma chave k_p a partir do processo de atestação remota.

Para utilizar chaves de 128 bits na comunicação segura, cada produtor $p \in [1, n]$ de um tópico atesta o agregador, ou seja, verifica se o agregador está sendo executado em um enclave SGX, por meio do processo de **Atestação Remota**, descrito por ANATI et al. [64]. Em poucas palavras, o processo de atestação remota aproveita os recursos do *hardware* para produzir uma assinatura criptográfica do conteúdo dos enclaves SGX com uma chave somente acessível pelo processador da plataforma. O serviço de atestação fornecido pela Intel (**IAS**) pode verificar se as informações produzidas realmente vieram de um enclave SGX com o código correto. Esse processo também possui um esquema de troca de chave subjacentes baseado no protocolo de troca de chave Diffie-Hellman (ECDH) da curva elíptica. Como resultado, cada produtor irá (i) verificar o agregador com relação a sua integridade e (ii) derivar a chave de 128 bits k_p para comunicações seguras – também derivada pelo agregador.

Ao receber dados confidenciais, o agregador pode decifrar as mensagens usando o mesmo algoritmo AES-GCM e, em seguida, executar a agregação nos dados confidenciais decifrados. A confidencialidade e a integridade dos dados são alcançadas por meio das garantias fornecidas pelos enclaves SGX [65].

Para agregar uma coleta de dados feita em um instante de tempo, os seguintes passos serão realizados:

1. O agregador envia uma mensagem de requerimento de dados ou, alternativamente, os produtores podem iniciar uma transmissão periodicamente;

2. Cada produtor p cria um valor aleatório (***nonce***) n_p, e calcula $C_p, M_p = G_e(v_p, n_p, k_p)$, onde C_p é o valor coletado por p após a encriptação AES-GCM, M_p é o código de autenticação de mensagem de v_p, onde v_p é o valor medido por p, e a função G_e é a função de criptografia AES-GCM no modo de encriptação.

3. Cada produtor publica C_p, M_p, e n_p ao tópico;

4. O agregador obtém $V = \sum_{p=1}^{n} v_p$ ao calcular $\sum_{p=1}^{n} G_d(C_p, n_p, k_p, M_p)$, onde a função G_d é a função AES-GCM no modo decriptação, ao mesmo tempo em que verifica a integridade de cada uma das mensagens recebidas no tópico;

5. O agregador publica o resultado obtido no mesmo ou em outro tópico do barramento de mensagens, de forma que ele fique disponível para possíveis consumidores.

Os dados confidenciais processados por este agregador estão livres de vazamento de informações porque a memória dentro de um enclave SGX é criptografada. Além disso, todos os dados são trocados usando uma conexão segura estabelecida pelo processo de atestação remota. Esse processo também garante que o código em execução dentro de um enclave SGX não tenha sido modificado.

4.3 Requisitos Funcionais

Essa arquitetura deve atender aos requisitos citados nas seções 3.1.10, 3.1.11, 3.1.12, assim como os requisitos funcionais a seguir.

4.3.1 Cadastrar Regiões

Descrição:	A aplicação deve permitir que novas regiões sejam criadas.
Justificativa:	Os dados devem ser agregados por região, logo, é necessário que o sistema permita que novas regiões sejam criadas.
Vantagens:	Essa separação é útil para facilitar a divisão da agregação dos dados em diferentes nós.
Desvantagens:	Nenhuma identificada.

4.3.2 Cadastrar Medidores Inteligentes

Descrição:	A aplicação deve permitir que novos medidores inteligentes sejam criados. Cada medidor inteligente possui um identificador único.
Justificativa:	O sistema deve possuir um cadastro de todos os medidores que podem acessar o sistema.
Vantagens:	Evita acessos indevidos ao sistema.
Desvantagens:	Nenhuma identificada.

4.3.3 Associar Medidor Inteligente a Região

Descrição:	A aplicação deve permitir a associação de um medidor inteligente a somente uma região.
Justificativa:	A associação deve ser necessária para que o sistema tenha o registro de quais medidores pertecem a qual região.
Vantagens:	Possibilita a agregação dos dados dos medidores por região.
Desvantagens:	Nenhuma identificada.

4.3.4 Calcular consumo de energia elétrica por região

Descrição:	A aplicação deve permitir o cálculo do consumo de energia elétrica por regiões ou grupos de medidores.
Justificativa:	Esses cálculos permitem a otimização da distribuição de carga no sistema elétrico.
Vantagens:	Redução de possíveis faltas de energia por má distribuição de carga.
Desvantagens:	Nenhuma identificada.

4.3.5 Calcular a conta mensal por consumidor

Descrição:	A aplicação deve permitir o cálculo da conta mensal por consumidor para reduzir os custos de verificação manual.
Justificativa:	Esses cálculos permitem a otimização da distribuição de carga no sistema elétrico.
Vantagens:	Redução de possíveis faltas de energia por má distribuição de carga.
Desvantagens:	Nenhuma identificada.

4.4 Requisitos Não-Funcionais

A arquitetura deve atender a todos os requisitos citados na seção 3.2.

4.5 Decisões Arquiteturais

Além de todas as decisões arquiteturais já citadas na seção 3.3, as decisões adicionais abaixo são necessárias para atender aos requisitos dessa aplicação.

4.5.1 Uso do Apache Kafka como barramento de mensagens

A solução Apache Kafka[1] foi escolhida para a comunicação entre produtores, agregadores e consumidores por prover todos os serviços exigidos para o barramento de mensagens (inclusive a troca segura de mensagens) permitindo a autenticação dos envolvidos e o uso de canais seguros de comunicação.

4.5.2 Utilizar Intel SGX como tecnologia de hardware para agregar dados

O sistema deve ter um agregador que utiliza Intel SGX para processar os dados. As chaves utilizadas pelo agregador devem ser negociadas através do processo de atestação remota.

[1] `http://kafka.apache.org`

4.5.3 Utilizar criptografia homomórfica como tecnologia de software para agregar dados

O sistema deve possuir um agregador que utiliza criptografia homomórfica para processamento dos dados. Esse agregador deve ser utilizado em provedores cuja tecnologia Intel SGX não esteja disponível.

4.6 Modelo de Adversário

No contexto dessa aplicação, considera-se que o **adversário** seja a própria concessionária, pois há possibilidade de que a mesma possa expor os dados para terceiros de alguma forma – por vazamento de dados, por exemplo. Consideramos que esse adversário seja honesto mas curioso (*honest, but curious*), de forma que ele é um participante legítimo no protocolo de comunicação que não se desviará do protocolo definido, mas tentará aprender todas as informações possíveis a partir de mensagens legitimamente recebidas [66].

Com isso, os medidores inteligentes devem utilizar técnicas de (i) **atestação** para verificar que o código em execução na concessionária não tenha sido modificado ou de (ii) criptografia homomórfica, para garantir que operações sejam realizadas nos dados mesmo em cifrotexto.

Para agregadores Intel SGX, confia-se na tecnologia aplicada e no serviço de atestação da Intel, de forma que não existam falhas nos processadores ou *bugs* na SDK e que o atacante não consiga burlar o ambiente de execução seguro que a tecnologia provê e/ou falsificar as requisições e respostas necessárias para efetivar uma **atestação remota**.

No caso de agregadores homomórficos, não há como realizar uma atestação para garantir a integridade do software, mas, confia-se na técnica aplicada, uma vez que os dados são computados de forma criptografada, o que garante confidencialidade por padrão. Por fim, tentar decifrá-los utilizando força bruta é bastante custoso computacionalmente.

É necessário também confiar que a aplicação, incluindo os *smart meters*, agregadores e consumidores, não contenha *bugs* e/ou falhas. Por fim, confia-se que os produtores não irão trabalhar juntos para expor alguém – por exemplo, produzindo sempre zero, de forma que a diferença seja referente aos outros produtores.

4.7 Avaliação da Arquitetura Proposta

No experimento realizado, cada medidor inteligente é simulado por uma *thread* em Java que gera valores de consumo e publica no barramento de mensagens. Todos os medidores de uma região r estão inscritos no mesmo tópico do barramento de mensagens. O agregador da região r é um processo em execução que também está inscrito no tópico em questão e é responsável por agrupar as medições para cada instante de tempo t e, quando os dados de consumo do conjunto de medidores da região são coletados para um instante t, é possível calcular a soma desses valores.

Para o cálculo da conta mensal com o agregador homomórfico, também desenvolvido em Java, cada medidor faz uso de k pares de chaves públicas e privadas, sendo um par para cada medição com um reaproveitamento de forma circular, de acordo com o módulo do identificador da medição. Por fim, a cada fim de ciclo, o processo de agregação é executado para aquele intervalo de tempo. Nesse caso, utiliza-se um conjunto de chaves para que a concessionária obtenha o consumo acumulado após um intervalo de tempo definido, e não o consumo exato em um determinado instante.

No caso do agregador Intel SGX, apenas uma chave simétrica fixa, previamente compartilhada com o agregador, é usada por cada cada medidor, além de um *nonce* gerado aleatoriamente a cada medição. Para efeito de simplificação, a troca de chaves de forma segura entre medidores e agregador foi abstraída, e pode ser considerada um problema resolvido na fase de atestação [67].

Finalmente, as duas abordagens são robustas contra situações onde o agregador publica resultados agregados mas que permitem a inferência de valores instantâneos. Na abordagem homomórfica, o produtor participa do processo de agregação e pode identificar situações onde as agregações usam poucos dados ou possuem intervalos com sobreposição. Na abordagem Intel SGX, o produtor validou o código do atestador inicialmente e este não pode ser atualizado sem uma nova atestação.

4.7.1 Experimentos e Resultados

Depois que as provas de conceito foram desenvolvidas, os experimentos foram executados para avaliar a viabilidade de cada agregador. A métrica escolhida para comparar os agrega-

dores foi **tempo de resposta**, porque muitos provedores de nuvem cobram pelo tempo que seus recursos foram usados. Foram feitos testes para medir os tempos de resposta dos agregadores **homomórficos**, **Intel SGX** e também dos agregadores sem garantia de segurança ou privacidade – importante para medir a sobrecarga imposta pelos agregadores seguros.

Os testes foram executados usando uma máquina com processador Intel *Kaby Lake* (7ª geração) i7 7700HQ e $16\ GB$ de RAM. O sistema operacional escolhido foi o Ubuntu 16.04 e, dependendo do agregador, os experimentos foram executados usando o $host$, usando contêineres *Docker* e também usando máquinas virtuais. Para o agregador Intel SGX, o KVM SGX[2] e o QEMU SGX[3] foram instalados para permitir que as máquinas virtuais usem as instruções do SGX. Dez replicações foram realizadas para cada experimento e foram consideradas as médias com intervalo de confiança de 95%.

A Tabela 4.1 contém a descrição, os parâmetros de entrada e o ambiente para cada teste realizado. Para os agregadores que não possuem segurança alguma e os que utilizam Intel SGX, o número de medidores variou de 50 a 1.500 para mostrar como eles funcionam em regiões pequenas, médias e grandes. Tipicamente, o número de residências atendidas varia de 50 a 150 [68], porém, um dos objetivos desse experimento foi estressar os agregadores e identificar o custo adicional nos cenários mais complexos.

Para agregadores homomórficos, decidiu-se usar o intervalo de duas medições para mostrar que, para períodos menores, esse agregador pode não ser uma escolha recomendada. É melhor usar essa abordagem quando os dados são coletados em períodos mais longos.

Em experimentos preliminares, verificou-se que esses valores cobriam um intervalo suficiente para determinar se o tempo de resposta cresce linear ou exponencialmente, dependendo do número de medidores. Para o agregador homomórfico, o limite superior foi de 250, pois valores maiores do que isso levariam a testes de duração muito longa.

A finalidade do primeiro experimento, mostrado na Figura 4.4, foi avaliar o tempo necessário para o uso do Intel SGX em diferentes ambientes. A sobrecarga de tempo de resposta imposta pela tecnologia Intel SGX é notável, especialmente quando usada em máquinas virtuais, como mostra a Figura 4.3. Por outro lado, o custo imposto por contêineres *Docker* é quase imperceptível.

[2] `https://github.com/01org/kvm-sgx/wiki`

[3] `https://github.com/01org/qemu-sgx/wiki`

Tabela 4.1: Caso de teste, ambiente, agregador e parâmetros de entrada usados para cada experimento

Agregador	Teste	Ambiente	Núm. de residências	Tempo total simulado	Intervalo entre medições
Não-seguro	Consumo por região	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	1 dia	60 segundos
Não-seguro	Conta mensal	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	30 dias	60 segundos
Intel SGX	Consumo por região	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	1 dia	60 segundos
Intel SGX	Conta mensal	Host, VM, Container	50, 100, 200, 250, 500, 750, 1000, 1250, 1500	30 dias	60 segundos
Homomórfico	Consumo por região	Host	10, 50, 100, 200, 250	1 dia	60, 900 segundos
Homomórfico	Conta mensal	Host	10, 50, 100, 200, 250	30 dias	60, 900 segundos

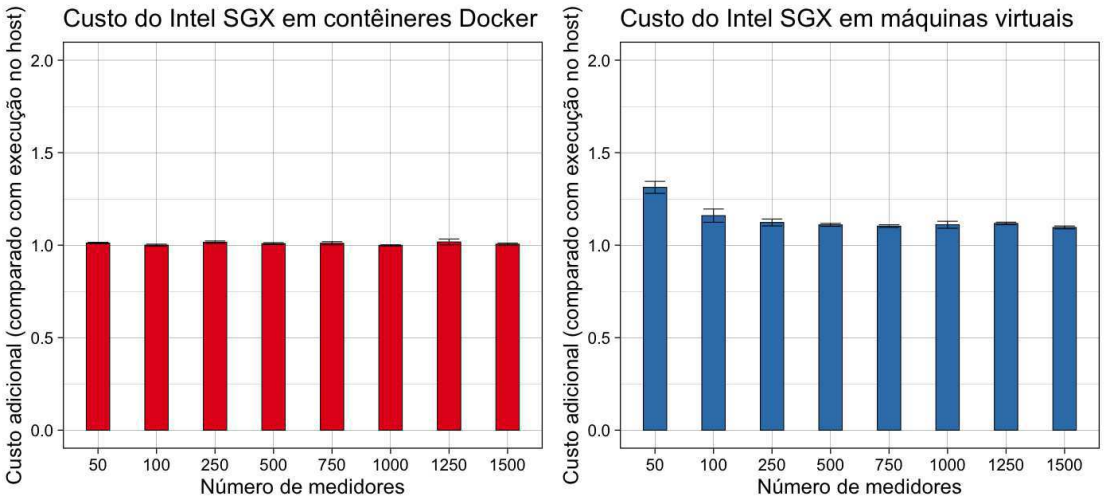

Figura 4.3: Custo do uso do Intel SGX em contêineres *Docker* e máquinas virtuais

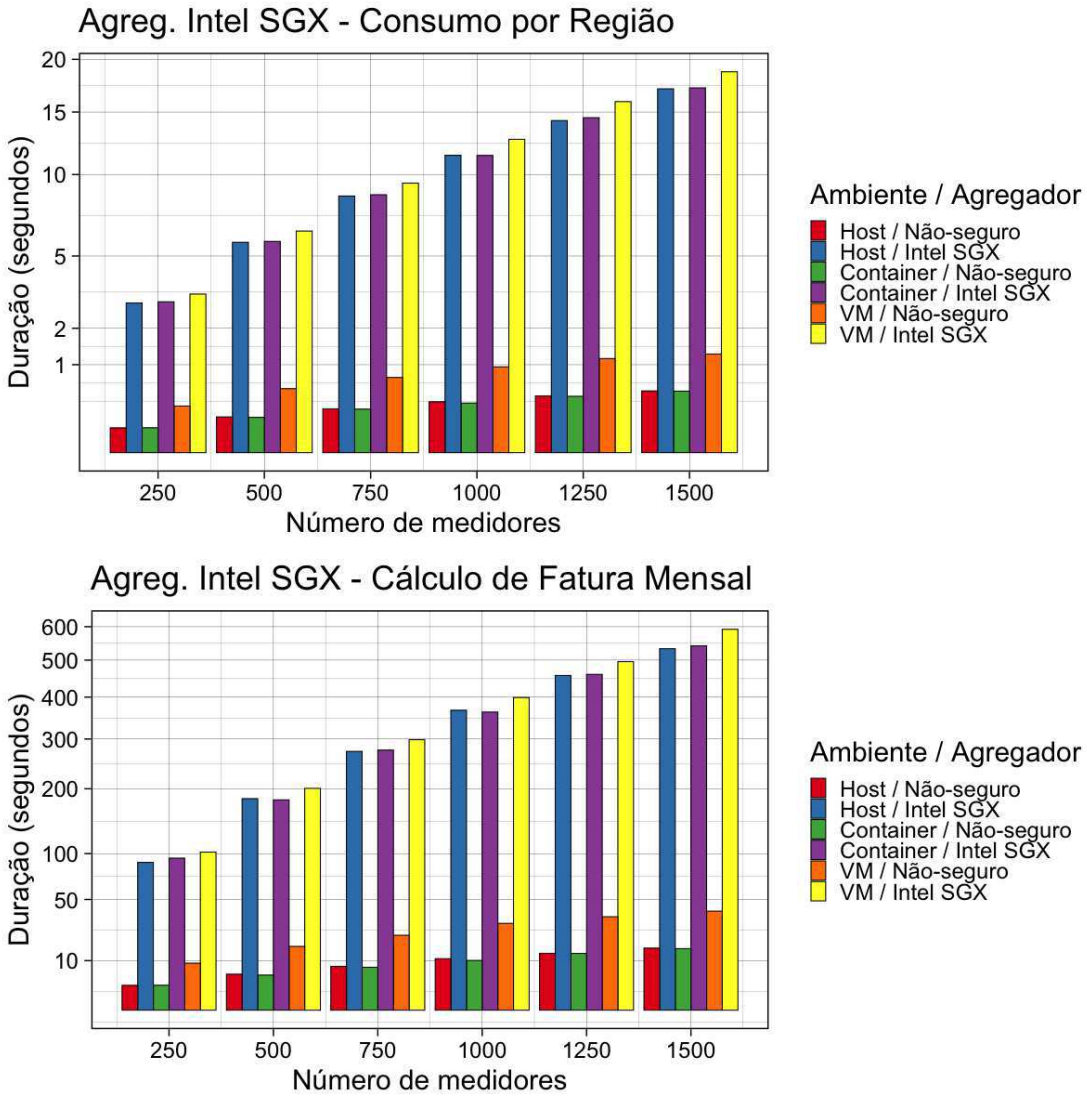

Figura 4.4: Comparação entre agregadores não-seguros e Intel SGX para o consumo regional e o cálculo mensal de faturas usando host, máquinas virtuais e contêineres *Docker*

Os resultados do agregador homomórfico podem ser vistos na Figura 4.5.

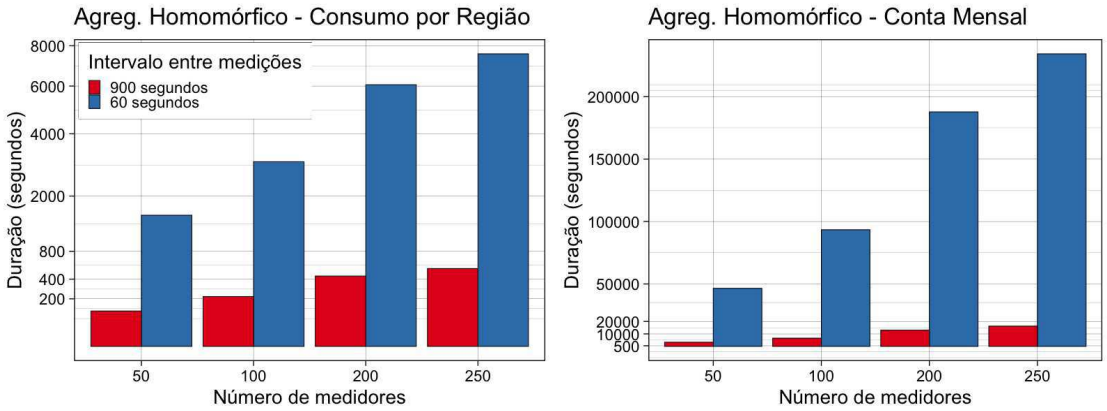

Figura 4.5: Agregação homomórfica para o consumo regional e faturamento mensal usando o ambiente do host

Em suma, encontra-se na Tabela 4.2 uma comparação entre os dois agregadores seguros desenvolvidos. Apesar da alta sobrecarga do agregador homomórfico, ele pode ser uma opção viável para cálculos mensais por ser paralelizável e aplicável a qualquer provedor de nuvem, ao contrário da tecnologia Intel SGX, que não está disponível em todos os provedores.

Tabela 4.2: Comparação entre os agregadores Intel SGX e homomórfico. Os tempos de resposta estão em segundos

Teste	Núm. de residências	Intel SGX		Homomórfico		
		Tempo total	Medição única	Tempo total	Medição única	Custo adicional
Consumo por região	50	0.584 ± 0.008	< 0.001	1523 ± 12.601	1.058	2608
Consumo por região	100	1.186 ± 0.009	< 0.001	3018.6 ± 13.675	2.096	2545
Consumo por região	200	2.428 ± 0.062	0.002	6060 ± 45.299	4.208	2496
Conta mensal	50	17.6 ± 1.110	< 0.001	46510.6 ± 319.974	1.077	2643
Conta mensal	100	36.2 ± 1.360	< 0.001	93412.8 ± 940.763	2.162	2580
Conta mensal	200	75.2 ± 2.964	0.002	187803.8 ± 564.790	4.347	2497

A abordagem de agregação de dados usando o Intel SGX produz tempos de resposta muito menores do que a abordagem de criptografia homomórfica. No entanto, cada abordagem tem vantagens e desvantagens que serão discutidas a seguir.

4.7.2 Discussão dos Experimentos

Apesar de ser mais lenta, a criptografia homomórfica tem vantagens que a tornam viável em certas aplicações. Nenhum hardware específico é exigido para sua execução, tornando-a

facilmente implantável em vários ambientes. Além disso, pode ser implementada em diferentes linguagens de programação, uma vez que é puramente matemática. Dependendo da quantidade de dados a ser computada, seu custo em termos de tempo pode ser insignificante em comparação aos benefícios de segurança habilitados por esta abordagem. Dado que toda a computação é feita sobre o texto cifrado, então todo o conteúdo é desconhecido para o provedor de serviços e/ou quaisquer invasores em potencial.

Outra vantagem clara da agregação homomórfica é que o consumidor de dados é incapaz de inferir as medições individuais de um determinado consumidor, porque seria necessário que todos os outros consumidores fossem corruptos.

Por outro lado, a abordagem homomórfica tem uma limitação relacionada às operações que podem ser feitas sobre o texto cifrado, dado que a maioria das criptografias são parcialmente homomórficas. Algoritmos que permitem cálculos arbitrários sobre o texto cifrado, propostos por GENTRY [31], ainda exigem tempos de processamento muito altos.

Outro fator negativo observado na agregação homomórfica é que uma troca de mensagens adicional é necessária para calcular o consumo de energia de uma região. Isso não é trivial porque o envio de mensagens do provedor de serviços públicos para os medidores requer que o medidor tenha um endereço acessível ou que ele utilize uma estratégia de *polling* para verificar periodicamente se há novas mensagens, como é necessário na segunda fase do algoritmo.

A abordagem com Intel SGX, por sua vez, tem alto desempenho e baixo custo adicional, se comparado ao uso de criptografia homomórfica. Isso deve-se ao fato de que não é necessário executar nenhuma computação sobre o texto cifrado além de fornecer garantias de segurança e privacidade para dados dos usuários. Além disso, essa abordagem não exige grandes esforços para implementar aplicativos que usam a tecnologia, dado que existem bibliotecas que suportam a criação, uso e atestação de enclaves. Dessa forma, sua adoção é interessante quando o hardware requerido está disponível e há uma necessidade de velocidade de execução, além da confidencialidade dos dados envolvidos.

No entanto, ainda existem algumas questões em aberto sobre a integração entre a solução Intel SGX e o ambiente de computação em nuvem. Enquanto as soluções de computação em nuvem são baseadas na dissociação completa entre hardware físico e aplicativos, por exemplo, através de virtualização de hardware, o Intel SGX vincula diretamente a segurança

de um aplicativo ao processo de atestação, que por sua vez exige a participação da Intel com o IAS.

Outro ponto negativo relativo ao uso da Intel SGX é sua API limitada que impossibilita, por motivos de segurança, a execução de chamadas do sistema a partir do código executado dentro de enclaves. Essa limitação por sua vez, pode dificultar a portabilidade de aplicações a serem usadas com a tecnologia. Por fim, outro ponto que deve ser levado em consideração por qualquer aplicativo que use o Intel SGX é que existe um limite de apenas 128 MB de memória disponível para criar e executar enclaves por servidor físico. Técnicas de paginação podem ser usadas para criptografar a memória antes de exportá-la para fora do processador, mas isso impõe cargas adicionais. [69]

4.8 Análise de Ameaças

Nesta seção, discutimos possíveis fragilidades em ambos os componentes: agregadores Intel SGX e homomórficos.

4.8.1 Vulnerabilidades no Intel SGX

No agregador Intel SGX, para obter valores individuais, o adversário precisa extrair esses dados do enclave. As possibilidades para isso são descritas a seguir.

- **Ataques de *side-channel***: ataques desse tipo ameaçam a confidencialidade dos dados, no entanto, não há riscos à integridade deles. O SGX protege contra muitos tipos de ataques, até mesmo de usuários e softwares privilegiados. No entanto, um adversário que faz uso de *side-channel* é capaz de coletar estatísticas da CPU em relação à execução e pode usá-las para deduzir as características do software que está sendo executado e os dados que estão sendo processados. Exemplos de análises são estatísticas de energia, estatísticas de desempenho, incluindo *misses* de cache de plataforma, estatísticas de ramificação via sincronização e informações sobre páginas acessadas por meio de tabelas de páginas. Está bem documentado que o SGX não se defende dos adversários de *side-channel* [70; 71], porém, há estratégias viáveis para se proteger desses ataques [72].

- **Vulnerabilidades de software**: dependendo da vulnerabilidade explorada, há ameaças à confidencialidade e/ou à integridade dos dados. Os componentes do SGX são complexos e dificilmente livres de bugs, como qualquer outro software. Existem *drivers*, bibliotecas, dependências e instruções complexas disponíveis para desenvolvedores. Além disso, os desenvolvedores de enclaves podem cometer erros e até mesmo as chamadas a áreas protegidas podem conter vulnerabilidades comuns, como sobrecargas de *buffer* baseadas em pilha, ponteiros pendentes e *strings* de formato sem validação. Esse problema é impulsionado pela parte limitada da memória do enclave porque afeta a eficácia da ASLR (*Address Space Layout Randomization*).

  ASLR é uma técnica de segurança envolvida na proteção contra muitos tipos de ataques. Por exemplo, para evitar que um atacante salte de forma confiável para uma função explorada específica na memória, essa ténica organiza aleatoriamente as posições de espaço de endereço das principais áreas de dados de um processo – incluindo a base do executável e as posições da pilha, *heap* e bibliotecas.

  Com o SGX, como o espaço de memória de um enclave é muito pequeno, um mecanismo simples de força bruta pode identificar facilmente o endereço correto. Realizou-se experimentos e observou-se que, para execuções diferentes, um elemento tem seus endereços alterados por apenas dois *bytes*, o que significa que a aleatorização é de aproximadamente apenas 65536 possibilidades. Isso é muito pequeno considerando que um invasor pode, por exemplo, aumentar a probabilidade de sucesso do ataque através da injeção de instruções NOP antes de um código malicioso [73].

  Para se defender de ataques assim, recomenda-se o uso de softwares de auditoria de código, adicionar códigos para detectar estouros de memória e encerrar a execução do programa, desativar a execução de código na *stack* e na *heap*, bem como utilizar ambientes com tratamentos para isso, como o SCONE[4].

- **Decifragem**: para obter dados individuais e confidenciais, os invasores podem tentar quebrar o canal de comunicação seguro estabelecido por meio do processo de atestação remoto. A troca de uma chave simétrica efêmera durante o processo de atestação remota é realizada através da execução de um *handshake* ECDH. Portanto, as vulnera-

[4]Mais detalhes podem ser vistos em `https://sconedocs.github.io/SCONE_ENV/#safety`

bilidades dependem da possibilidade de resolver o problema do logaritmo discreto da curva elíptica. Os ataques *man-in-the-middle* são mitigados através do uso de certificados autenticados pela Intel durante o processo de atestação remota.

A criptografia simétrica é usada na comunicação segura (AES-GCM ou CTR) e também pelo mecanismo de criptografia de memória para criptografar o enclave usando o AES-CTR. Os atacantes podem tentar decifrá-los por força bruta e obter dados individuais e confidenciais. No entanto, no momento, tal ataque não é computacionalmente viável e não há nenhum ataque prático conhecido que permita que alguém sem conhecimento da chave leia dados cifrados quando tais algoritmos são implementados corretamente. AES é uma cifra confiável, e os únicos ataques práticos bem-sucedidos foram ataques de *side-channel* em pontos fracos encontrados na implementação ou no gerenciamento de chaves de produtos específicos de criptografia baseados nessa cifra.

4.8.2 Vulnerabilidades na Criptografia Homomórfica

Com o agregador homomórfico, o adversário precisa decifrar os dados. As possibilidades para isso são descritas abaixo.

- **Resolvendo um problema computacionalmente intratável**: Algoritmos de criptografia assimétrica cujas propriedades homomórficas se baseiam na dificuldade em resolver um problema em tempo polinomial. Por exemplo, *unpadded* RSA e ElGamal fornecem propriedades homomórficas multiplicativas. O RSA é baseado na dificuldade em fatorar um inteiro [74] e o ElGamal é baseado na dificuldade em calcular um logaritmo discreto em tempo polinomial [61]. Paillier é outro algoritmo de criptografia assimétrica baseado na dificuldade em fatorar um inteiro, mas sua propriedade homomórfica é aditiva [34].

  Existem muitos algoritmos para calcular o logaritmo discreto, mas nenhum deles é executado em tempo polinomial. Um exemplo de algoritmo é o rho de Pollard para logaritmos [75]. Para fatoração de números inteiros, o mais eficiente conhecido é o *general number field sieve* (GNFS) [76].

- **Explorando falhas de implementação**: Desde suas publicações, os sistemas de criptografia assimétrica foram analisados em busca de vulnerabilidade por muitos pesquisadores. Eles ilustram principalmente os perigos do uso indevido dos algoritmos. De fato, implementá-los com segurança é uma tarefa não trivial. Dan Boneh [77] apresenta alguns desses ataques ao RSA e descreve as ferramentas matemáticas subjacentes que eles usam.

  Por exemplo, ao usar o RSA, Wiener [78] mostra que um pequeno expoente privado d resulta em uma quebra total do sistema criptográfico. Uma consequência do pequeno expoente privado d é que o expoente público e é grande (lembrando que $e \cdot d \; mod \; N = 1$). Por outro lado, quando o expoente público e é muito pequeno, a criptografia pode ser suscetível a ataques baseados no teorema de Coppersmith [79].

  Existem muitos outros ataques de RSA, como o ataque de *Broadcast*, de HASTAD [80], o ataque de mensagem relacionada, de FRANKLIN et al. [81] e o ataque de Exposição de Chave Parcial [82]. Outros sistemas criptográficos com propriedades homomórficas, como ElGamal e Paillier, quando implementados incorretamente, também sofrem de vulnerabilidades semelhantes.

Capítulo 5

Aplicação II: Publicidade Segmentada

A publicidade segmentada é uma forma de publicidade em que os anunciantes podem usar métodos sofisticados para segmentar os públicos mais receptivos com determinadas características, com base no produto ou na pessoa que o anunciante promove [83]. Esses traços podem ser demográficos, focados em raça, status econômico, sexo, idade, nível de educação, nível de renda e emprego, ou podem ser baseados nos valores, personalidade, atitudes, opiniões, estilos de vida e hábitos do consumidor [84].

Nesse estudo de caso, aplicações em dispositivos móveis de usuários podem conter ferramentas de coleta de dados para aprimorar suas sugestões de publicidade. Essas ferramentas podem coletar informações de localização – dados de GPS, acelerômetro e outros, redes Wi-Fi na redondeza, versão do aparelho utilizado, horários de uso do dispositivo e mais.

A arquitetura em questão visa a garantia de segurança e privacidade dos dados de sensores enviados por aplicações instaladas em dispositivos móveis de usuários. Há necessidade de cuidado com os dados de sensores em seu armazenamento, processamento e acesso. Para garantir isso, componentes críticos, como gerenciadores de chaves e serviços de armazenamento e agregação de dados, devem fazer uso de tecnologias de hardware. Nesse caso, escolheu-se Intel SGX, por ser uma tecnologia promissora em uso crescente pela comunidade e mantida por uma empresa de grande escala e reputação[1].

[1] A empresa que irá utilizar essa arquitetura e alguns detalhes de implementação não serão abordados por questões de confidencialidade.

5.1 Visão Geral

O diagrama exposto na Figura 5.1 contém os componentes da arquitetura proposta. Todos os dados são coletados por uma biblioteca (**SDK**) e enviados via um **Gateway** para o barramento de mensagens (**KafkaBus**). Os dados são consumidos pelo **Reencrypter** e pelo **LocationMatching** e são armazenados pelo sistema de gerenciamento de banco de dados (**MongoDatabaseManager**).

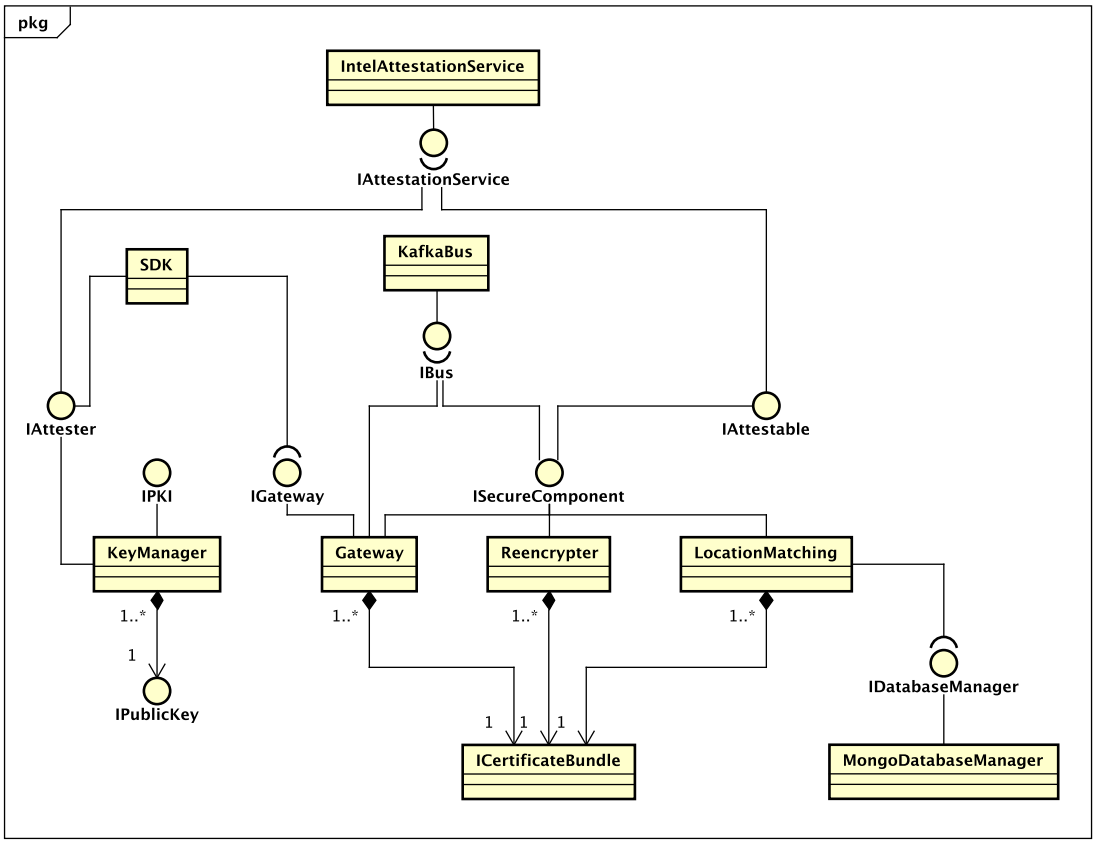

Figura 5.1: Modelo UML da arquitetura de software proposta

5.1.1 Elementos Arquiteturais

Uma versão simplificada da arquitetura é ilustrada na Figura 5.2. Os dispositivos móveis utilizam o ***Gateway*** para se comunicar com o barramento seguro, que é composto pelo **Apache Kafka** e pelo **Key Manager** – responsável pela gerência dos certificados. O **Reencrypter** é responsável pela criptografia dos dados em diversos níveis. Por fim, esses dados são processados pelo **Location Matching** e armazenados no **Apache Mongo**. Cada componente

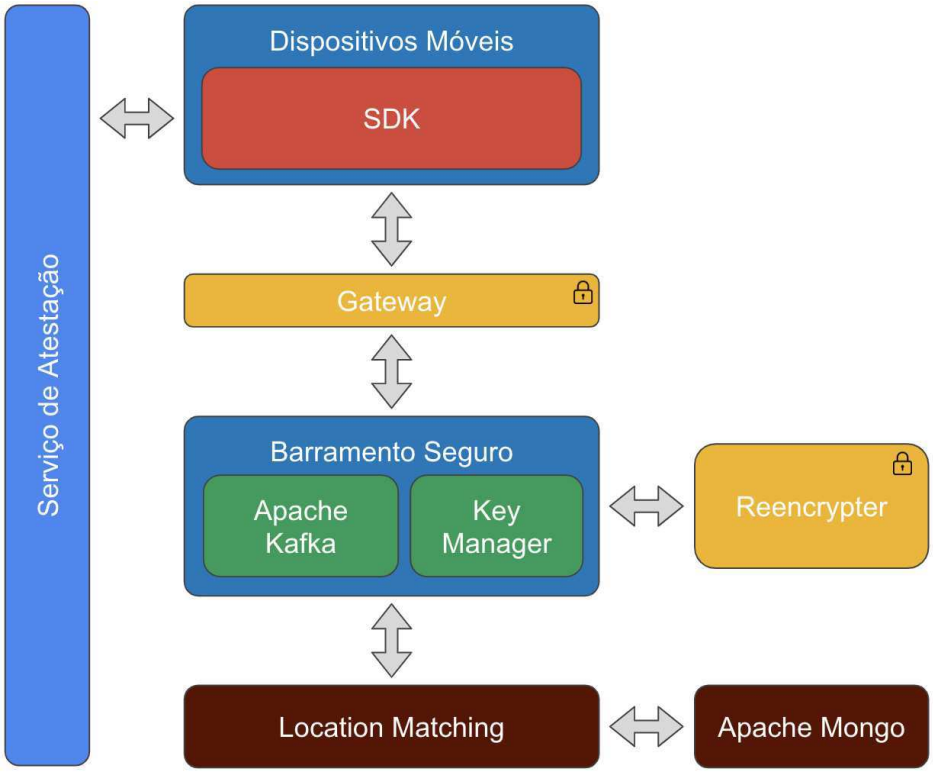

Figura 5.2: Diagrama simplificado da arquitetura de software proposta

citado é descrito em detalhes a seguir.

KafkaBus

O barramento de mensagens utilizado é o **Apache Kafka**, compartilha as funcionalidades citadas na Seção 4.2.1 e recebe mensagens do **SDK** a partir do **Gateway**. Todas as mensagens geradas pelo **Gateway**, **Reencrypter** e **LocationMatching** passam por esse barramento.

SDK

O **SDK** possui implementações para os sistemas operacionais **Android** e **iOS** e é importado por aplicações de terceiros que desejam exibir publicidades. Esse **SDK** coleta informações para determinar o perfil do usuário, assim como a sua localização de tempos em tempos.

O **SDK** possui a funcionalidade de atestação remota do **Gateway** e da validação dessa atestação com o **IntelAttestationService**.

KeyManager

O **KeyManager** contém um subconjunto de operações do componente citado na Seção 3.4.5. Ele armazena as chaves e quaisquer outras informações seguras que serão fornecidas ao **Gateway**, **Reencrypter** e **LocationMatching**, bem como realiza a atestação remota dos mesmos antes do fornecimento.

Os segredos do **KeyManager** são armazenados em um arquivo selado com a tecnologia Intel SGX, que impede que o mesmo seja lido por aplicações de terceiros, mesmo se tentarem dentro de um enclave, pois a decifragem desse arquivo depende de duas chaves: uma que está inserida no processador em que o sistema é executado e outra que é única para o desenvolvedor da aplicação[2].

Gateway

O **Gateway** é a porta de entrada dos dados enviados pelo **SDK** via HTTPS. Esse componente é implementado com suporte à tecnologia Intel SGX, de forma que os dados recebidos são desencriptados dentro de um enclave, cuja chave é negociada via atestação remota, por parte do **SDK**. Como dito anteriormente, todos os dados recebidos são encriptados com uma chave, que fora obtida do **KeyManager** e então inseridos no barramento de mensagens.

A aplicação pode ter um ou mais instâncias do **Gateway**, dependendo da taxa de entrada de mensagens. Deve existir uma associação entre um **SDK** e um **Gateway**, a fim de evitar múltiplas atestações remotas e/ou chaves compartilhadas para um mesmo usuário.

Reencrypter

Esse componente é responsável por reencriptar os dados de nível 1 com diferentes chaves. Além de reencriptar os dados para aplicações de nível 2 ou superior, o **Reencrypter** reduz a precisão dos mesmos – para dados de localização, quanto maior o nível, menor a precisão dos dados. Esse processo é realizado dentro de um enclave Intel SGX e todas as chaves são recuperadas do **KeyManager**, após atestação remota.

[2]Mais detalhes podem ser vistos em `https://software.intel.com/en-us/blogs/2016/05/04/introduction-to-intel-sgx-sealing`

LocationMatching

O **LocationMatching** é o componente responsável por (i) ofuscar informações de identificação do usuário; (ii) delegar o armazenamento dos dados de localização do usuário ao **MongoDatabaseManager**; (iii) analisar os dados como parte do processo de detecção de lugares relevantes.

MongoDatabaseManager

O **MongoDatabaseManager** é responsável por gerenciar – inserir, buscar, atualizar e remover – quaisquer dados da aplicação, dentre eles, as publicidades, os perfis dos usuários, as associações entre publicidades e perfis de usuários e os dados de localização em tempo real de cada usuário.

IntelAttestationService

Para validar cada execução do processo de atestação remota, é necessário contatar o serviço de atestação da Intel. Esse serviço é acessível através de um *web service* e não há cobrança por ele. É possível também criar um serviço de atestação próprio, porém foge do escopo dessa aplicação[3].

5.1.2 Cenários

A Figura 5.3 ilustra a inicialização dos componentes seguros da arquitetura:

- **KeyManager:** esse componente representa a Infraestrutura de Chaves Públicas, que é definido na Seção 3.4.5 e deve ser iniciado antes dos componentes restantes, pois contém os segredos a serem utilizados por eles;
- **Gateway**, **Reencrypter** e **LocationMatching:** esses componentes acessam o **KeyManager** e são atestados pelo mesmo para que possam receber os segredos necessários para tratar mensagens enviadas pelo **SDK**. Cada componente possui um con-

[3]Mais detalhes sobre isso podem ser vistos em `https://software.intel.com/en-us/blogs/2018/12/09/an-update-on-3rd-party-attestation`

junto de segredos específico. Além disso, o **KeyManager** também é atestado por cada um desses componentes, para reduzir a chance de vazamento de dados;

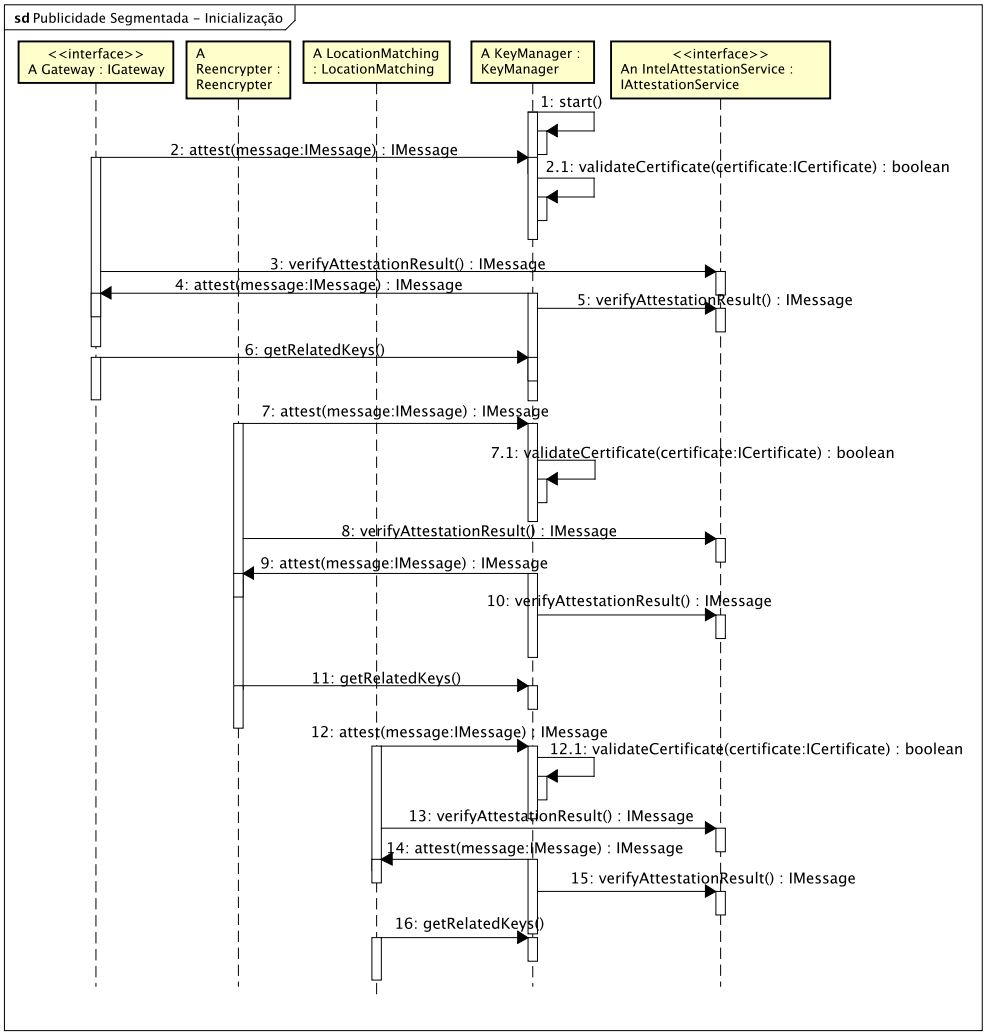

Figura 5.3: Diagrama de sequência para a inicialização dos componentes seguros

O fluxo de coleta e processamento de dados, que é detalhado na Figura 5.4, é descrito a seguir:

- **SDK:** acessa o **Gateway** e inicia o processo de atestação. Esse processo é feito por troca de mensagens HTTP utilizando protocolo SSL;
- **Gateway:** envia os dados necessários para que o **SDK** possa validar o processo de atestação no **IntelAttestationService**. Os dados coletados para atestação são gerados dentro do enclave Intel SGX;

- **SDK:** envia os dados de resposta do **Gateway** para o serviço de atestação (**IntelAttestationService**);
- **IntelAttestationService:** envia uma resposta positiva ou negativa com relação à validade do *QUOTE* enviado. Uma verificação positiva indica que a chave segura (simétrica) negociada durante o processo de atestação pode ser utilizada para comunicação segura entre as partes. Nesse momento, ambas as partes (**SDK** e **Gateway**) são confiáveis;
- **SDK:** periodicamente, coleta, criptografa com a chave segura e envia dados criptografados para o **Gateway** utilizando conexão segura (SSL);
- **Gateway:** Os dados criptografados com a chave segura são desencriptados dentro do enclave Intel SGX. Em seguida, são criptografados com uma chave de acesso a dados de nível 1;
- **Gateway:** Os dados criptografados com a chave de acesso de dados nível 1 são enviados para tópicos no Apache Kafka (**KafkaBus**) utilizando protocolo SSL;
- **KafkaBus:** encaminha os dados do **Gateway** para os consumidores dos respectivos tópicos;
- **Reencrypter:** esse componente – um exemplo de aplicação de nível 1 – lida com os dados sem nenhuma agregação que foram enviados pelo **SDK**. Quaisquer aplicações que lidam com os dados puros devem desencriptar e agregar os dados dentro de enclaves Intel SGX. Os dados agregados são encriptados com chaves de acesso de níveis 2 ou superior e inseridos no **Apache Kafka**;
- **LocationMatching e outras aplicações de nível 2 e superiores:** recebem e desencriptam os dados agregados utilizando a chave de acesso de relacionada.

5.2 Requisitos Funcionais

Essa arquitetura deve atender aos requisitos citados nas Seções 3.1.4, 3.1.5, 3.1.6, 3.1.7, 3.1.8, 3.1.9, 3.1.10, 3.1.11, 3.1.12, 3.1.13, 3.1.14, assim como os requisitos funcionais a

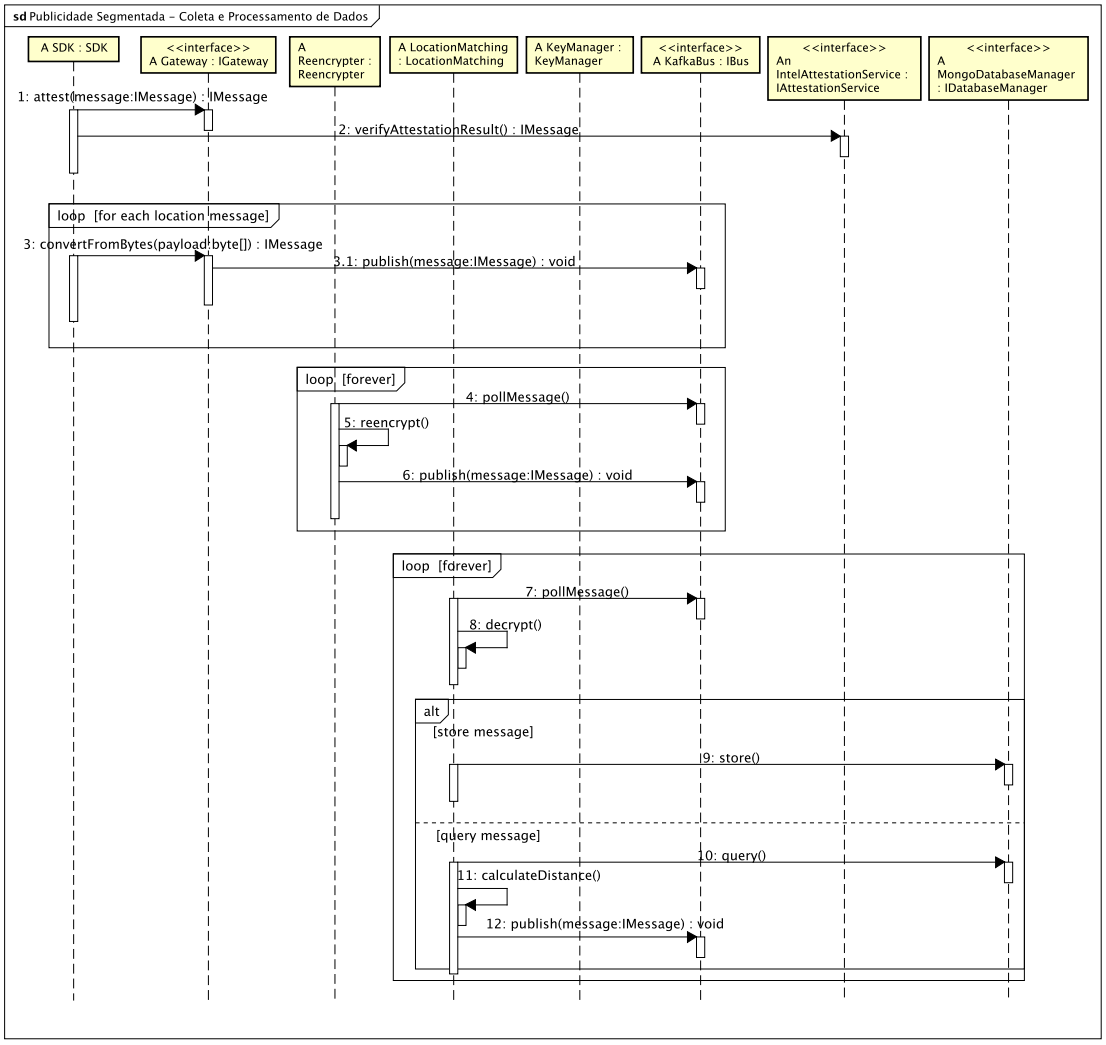

Figura 5.4: Diagrama de sequência para a coleta e processamento de dados

seguir.

5.2.1 Cadastrar Publicidades

Descrição:	Deve existir um cadastro de publicidades que podem ser sugeridas pela aplicação. Esse cadastro deve conter informações como tipo de mídia a ser exibida e links de acesso para o usuário.
Justificativa:	O cadastro de publicidades permite a geração das sugestões para os usuários.
Vantagens:	Maior flexibilidade ao sistema.
Desvantagens:	Nenhuma identificada.

5.2.2 Cadastrar Perfis de Usuário

Descrição:	A aplicação deve possuir um cadastro de perfis de usuário, em que o intervalo de idade, de nível de renda e o sexo é armazenado.
Justificativa:	Os perfis dos usuários possibilitam sugestões mais direcionadas aos usuários.
Vantagens:	Maior flexibilidade ao sistema.
Desvantagens:	Nenhuma identificada.

5.2.3 Associar Perfis de Usuário a Publicidades

Descrição:	Cada publicidade pode ser associada a um ou mais perfis de usuário.
Justificativa:	Para viabilizar o tipo de publicidade implementada, é necessário ter o cadastro de quais perfis de usuários podem visualizar certas publicidades.
Vantagens:	Sugestões melhor direcionadas.
Desvantagens:	Aumento na complexidade da implementação e no tempo de processamento das sugestões.

5.2.4 Criar sugestões de publicidade de acordo com os dados do usuário

Descrição:	A aplicação deve ser capaz de realizar sugestões de publicidade com base na localização em tempo real do usuário. O seu perfil, como idade, nível de renda e sexo também devem ser considerados.
Justificativa:	As sugestões de publicidade devem considerar também a localização do usuário, para que ele visualize publicidades de locais próximos.
Vantagens:	Sugestões melhor direcionadas.
Desvantagens:	Coleta de informações sensíveis de localização, porém, isso é amenizado pelas técnicas de confidencialidade e integridade aplicadas.

5.3 Requisitos Não-Funcionais

A arquitetura deve atender a todos os requisitos citados na Seção 3.2.

5.4 Decisões Arquiteturais

Além de todas as decisões arquiteturais já citadas na Seção 3.3 e na Seção 4.5.1, as decisões adicionais abaixo foram necessárias para atender aos requisitos dessa aplicação.

5.4.1 Os dados no barramento de mensagens devem ter diferentes níveis de criptografia

Cada tópico do barramento será composto por mensagens encriptadas por uma chave distinta. Caso uma aplicação de um certo nível seja comprometida, é possível trocar a chave utilizada para encriptar as mensagens, fazendo com que novos dados não sejam acessíveis.

5.4.2 O campo de identificação para usuários deve ser ofuscado

Além de encriptar as informações armazenadas do usuário, como idade, sexo e localização, o identificador único do mesmo deve ser ofuscado com o uso de funções *hash*. A função *hash* deve ser executada em ambientes seguros, para evitar que o identificador original seja obtido por algum atacante.

5.4.3 Utilizar MongoDB para armazenar os dados de localização dos usuários

Todos os dados de localização devem ser armazenados em um banco de dados MongoDB[4]. Como os dados trafegados são todos no formato JSON, eles podem ser armazenados sem conversão no MongoDB, permitindo também consultas de forma eficiente nos mesmos. Além disso, o MongoDB foi concebido com foco em desempenho, uso na nuvem, escalabilidade e permite replicação dos dados – aumentando a tolerância a falhas.

[4] `https://www.mongodb.com/`

5.4.4 Os componentes Gateway e Reencrypter devem ser desenvolvidos em C++

Tanto o **Gateway** quanto o **Reencrypter** devem ser implementados na linguagem C++ com o uso do SDK do Intel SGX para maximizar a vazão de dados, a fim de evitar que a porta de entrada do sistema seja um gargalo.

Com relação à complexidade de desenvolvimento causada por limitações no SDK Intel SGX, o impacto é mínimo pois ambos os componentes fazem uso somente de funções de criptografia, que são fornecidas nessa SDK. E, por outro lado, o consumo de memória é reduzido, já que somente as chaves precisam ficar na memória de um enclave Intel SGX, evitando que o limite de 128 MB seja atingido e permitindo que várias instâncias desses componentes sejam executadas sem comprometer o desempenho.

5.4.5 Utilizar TaLoS como biblioteca para a terminação de comunicação TLS dentro de um clave

O TaLoS [85] é uma biblioteca de código-aberto que aumenta a segurança de conexões SSL com o uso de Intel SGX. Ela permite adaptar aplicações que fazem uso de SSL com pouco esforço, pois expõe a mesma API do OpenSSL. Com essa biblioteca, as chaves privadas, as chaves de sessão – utilizadas para comunicar-se com os clientes – e as operações para encriptar e desencriptar os dados estão dentro de um enclave Intel SGX.

5.4.6 O componente LocationMatching deve ser desenvolvido com Python em um ambiente SCONE

Deve-se utilizar Python com SCONE para implementar o **LocationMatching** devido a sua maior complexidade, pois o componente deve realizar sugestões de publicidade dependendo da localização e do perfil do usuário, bem como gerenciar os dados de localização armazenados no MongoDB. Embora o desempenho de aplicações com SCONE e Python seja inferior em relação a aplicações em C++ com Intel SGX, esse componente possui uma taxa de entrada bem menor que o **Gateway** e o **Reencrypter**.

5.5 Modelo de Adversário

Similar ao modelo em 4.6, considera-se que o adversário seja a prestadora dos serviços de publicidade, porque há possibilidade de coleta dos dados coletados por terceiros – invasão, funcionários mau intencionados e outras possibilidades. Considera-se aqui também um modelo de adversário honesto mas curioso, que tentará aprender todas as informações possíveis a partir de mensagens recebidas de forma legítima. A chance de vazamento é amenizada com o uso de atestação remota e pela criptografia dos dados de localização armazenados.

Deve-se confiar na solução de hardware adotada (Intel SGX), no serviço de atestação da Intel e nas funções de criptografia utilizadas para comunicação e armazenamento dos dados. Não é necessário confiar na prestadora de serviços pois, ao confiar no processo de atestação remota provido pelas extensões de hardware, pode-se confirmar se o código em execução não contém funcionalidades comprometidas. Assim como disposto na Seção 4.6, é necessário confiar também que não existam *bugs* ou falhas na SDK, no processador com relação à tecnologia Intel SGX e nos módulos dessa aplicação.

5.6 Avaliação da Arquitetura Proposta

5.6.1 Experimentos e Resultados

Um conjunto de experimentos foi planejado e executado a fim de obter estimativas de custo em termos de desempenho, além de verificar a viabilidade da arquitetura proposta para essa aplicação. Esses experimentos foram realizados em dois ambientes diferentes, listados a seguir.

- **Ambiente OpenStack:** foram provisionadas duas máquinas virtuais idênticas com as seguintes características:
  - **Modelo do processador:** Intel(R) Xeon(R) CPU E3-1280 v6;
  - **Núcleos de processamento:** 4;
  - **Memória RAM:** 8 GB;
  - **Hipervisor:** KVM com suporte a Intel SGX;

  - **Sistema Operacional:** Ubuntu Server 16.04 LTS.

- **Ambiente SoftLayer da IBM[5]:** foram alocados dois servidores dedicados (*bare metal* idênticos com as seguintes características:

  - **Modelo do processador:** Intel(R) Xeon(R) CPU E3-1270 v6;
  - **Núcleos de processamento:** 4;
  - **Memória RAM:** 16 GB;
  - **Hipervisor:** KVM com suporte a Intel SGX;
  - **Sistema Operacional:** Ubuntu Server 16.04 LTS.

O ambiente da SoftLayer da IBM foi escolhido por ser um dos únicos provedores a permitir o uso de Intel SGX até o momento da escrita desse documento – dentre eles, Google Cloud Engine e Amazon Web Service não possuem suporte a Intel SGX. Há possibilidade também do uso do Intel SGX com Microsoft Azure, porém esse acesso não foi conseguido antes do término das análises.

O ambiente OpenStack foi escolhido por ser uma solução de nuvem de código-aberto adotada por grandes empresas, como IBM, Dell, HP, Cisco, Rackspace e outros. Além disso, é possível acoplar o hipervisor KVM com suporte a Intel nesse ambiente.

Todos os experimentos foram realizados em contêineres Docker, por terem baixa perda de desempenho e por facilitarem a preparação, execução e monitoramento desses experimentos nos ambientes escolhidos.

A Tabela 5.1 contém os experimentos, as taxas de entrada escolhidas, onde os mesmos foram realizados e seus objetivos. Para os dois ambientes, os experimentos foram replicados dez vezes para cada taxa de entrada e as mensagens utilizadas foram de 1024 bytes – escolheu-se esse tamanho pois os testes foram realizados em redes de 1 Gbps e, dependendo do experimento, mensagens com tamanhos maiores poderiam ter a rede como fator limitante. Todos os valores mostrados a seguir utilizam a média com intervalo de confiança de 95%.

Para os três experimentos citados, há somente uma instância de cada componente em execução. Os três componentes implementados – **Gateway**, **Reencrypter** e

[5]Mais informações em `http://www.softlayer.com/pt-br`

Tabela 5.1: Listagem dos experimentos realizados, ambientes, parâmetros de execução e objetivos

Componentes	Ambientes	Taxas de Entrada	Objetivos
Gateway, **Apache Kafka**	OpenStack	25, 50, 75, 100 e 125 requisições por segundo	Determinar o custo adicional ao utilizar terminações TLS dentro de um enclave.
Reencrypter, **Apache Kafka**	IBM SoftLayer, OpenStack	1000, 2500, 5000 e 10000 mensagens por segundo	Determinar o custo adicional ao utilizar o **Reencrypter** com até dois níveis diferentes de criptografia.
LocationMatching, **Apache Kafka**, **MongoDB**	IBM SoftLayer, OpenStack	200, 400, 600, 800 e 1000 mensagens por segundo	Determinar o custo adicional ao executar o **LocationMatching** com **SCONE**.

LocationMatching – utilizam somente um núcleo de processamento. Para que múltiplos núcleos sejam utilizados, mais instâncias podem ser executadas.

Para o primeiro experimento, duas versões do **Gateway** foram compiladas. A versão com a abordagem não-segura utiliza a biblioteca **LibreSSL**. Já a abordagem com Intel SGX utiliza o **TaLoS**, que é baseado na mesma versão da biblioteca **LibreSSL** usada na versão não-segura.

Os resultados do primeiro experimento – composto pelo **Gateway** e **Apache Kafka** – podem ser vistos nas Figuras 5.5 e 5.6. As requisições ao **Gateway** foram feitas com o **Apache JMeter**[6], e as mensagens foram consumidas do **Apache Kafka** por uma aplicação desenvolvida em C. A latência foi calculada a partir da diferença entre o momento de consumo final de cada mensagem – após o processamento do **Gateway** – e cada requisição HTTPS, feitas pelo JMeter. O uso de CPU – considera-se aqui o percentual de uso com relação a todos os núcleos de processamento, memória e rede foram capturados por ferramentas providas pelo **Docker**.

O segundo experimento foi dividido em duas rodadas. A primeira rodada foi composta por um **produtor** de mensagens, pelo **Apache Kafka** e por um **consumidor** – o mesmo utilizado no primeiro experimento. Na segunda rodada, o componente **Reencrypter** foi adicionado, a fim de avaliar os custos impostos por ele em termos de latência, uso de CPU, de memória e de rede. As Figuras 5.7 e 5.8 contêm os resultados desse experimento.

O terceiro experimento envolve um produtor – o mesmo do experimento anterior, o **Apa-**

[6]Aplicação indicada para testes de *stress* em sistemas web. Mais informações em `http://jmeter.apache.org`

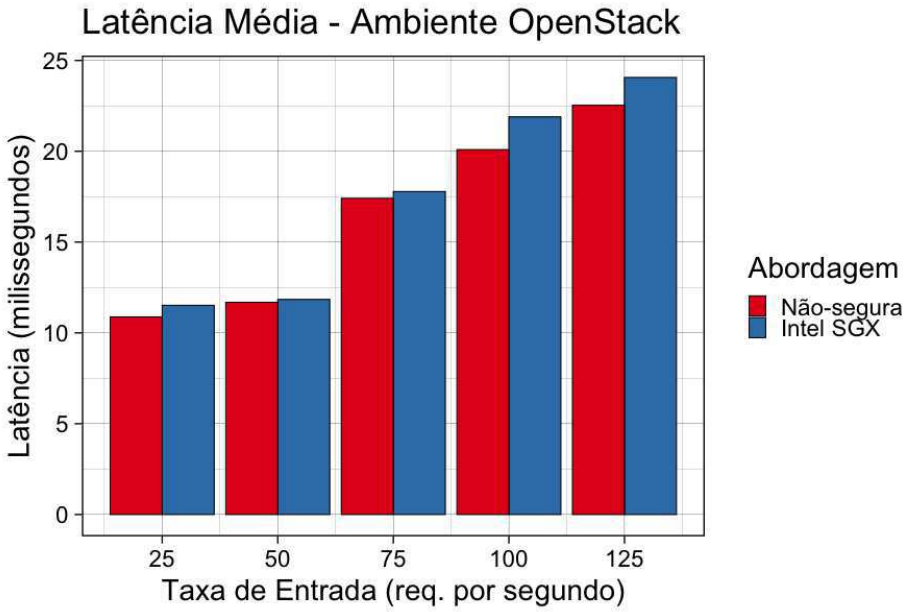

Figura 5.5: Média das latências para o experimento com o **Gateway**.

che Kafka, o **MongoDB** e o **LocationMatching**. Duas rodadas foram executas, de forma que a primeira foi composta pelo **LocationMatching** sem o **SCONE**. Na segunda rodada, o **LocationMatching** foi executado com o **SCONE**. Os resultados do experimento podem ser vistos nas Figuras 5.9 e 5.10.

5.6.2 Discussão dos Experimentos

No primeiro experimento, testou-se o **Gateway** e foi possível verificar que a latência do sistema se manteve próxima nas duas abordagens. O uso de CPU se manteve abaixo de 10% para as duas versões do **Gateway**. Para esse cenário, a solução com Intel SGX teve até 74% a mais de uso do CPU em relação à abordagem não-segura. A memória consumida pela versão segura ficou em torno de 21 MB, enquanto que a não-segura consumiu uma média de 1,5 MB. Em ambas abordagens, a memória não variou diante da taxa de entrada. Por fim, esse componente não teve impacto visível no **Apache Kafka**, pois a quantidade de mensagens continua a mesma, embora a versão segura trafega mensagens um pouco maiores, por serem criptografadas.

No segundo experimento, a introdução do **Reencrypter** tem impacto direto no **Apache Kafka**. Com dois níveis de criptografia adicional e com a taxa de entrada de 10000 men-

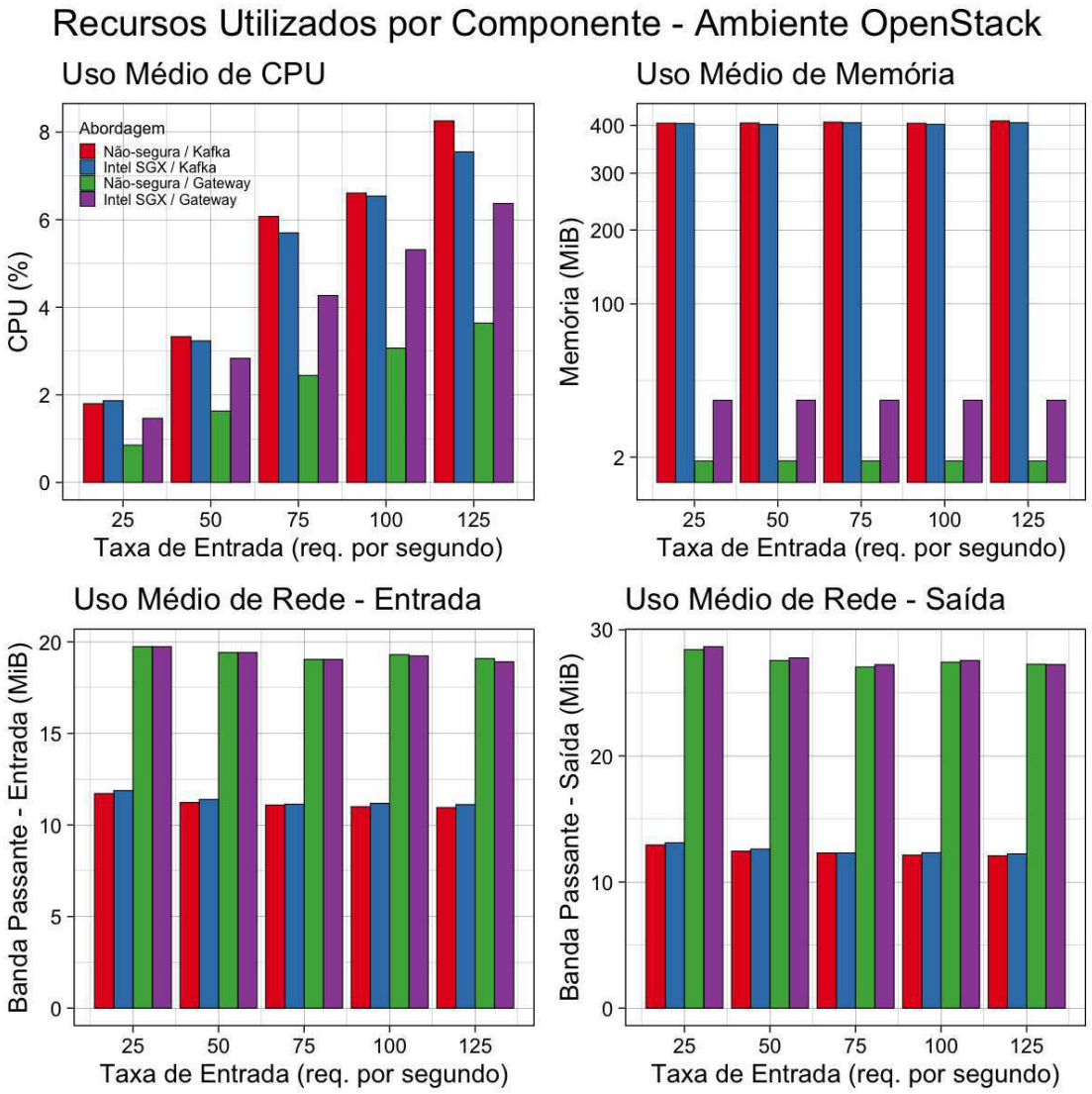

Figura 5.6: Recursos utilizados por componente para o experimento com o **Gateway**.

sagens por segundo, a latência passou de uma média de 14 ms para 170 ms, já o uso de processamento do **Apache Kafka** foi de 11% para 36%. Em média, a memória consumida subiu 750 MB para cada nível adicional. Para a rede, o tráfego aumenta 100% para cada nível de criptografia introduzido. Nesse cenário, os gargalos visíveis são o próprio **Apache Kafka** e a rede.

Para o caso do **Location Matching**, a abordagem segura tem uma latência menor que 10 segundos até uma taxa de entrada de 400 mensagens por segundo. Acima disso, a solução segura não atende bem, pois a latência cresce expressivamente, chegando a uma média de 40 segundos para uma taxa de entrada de 1000 mensagens por segundo.

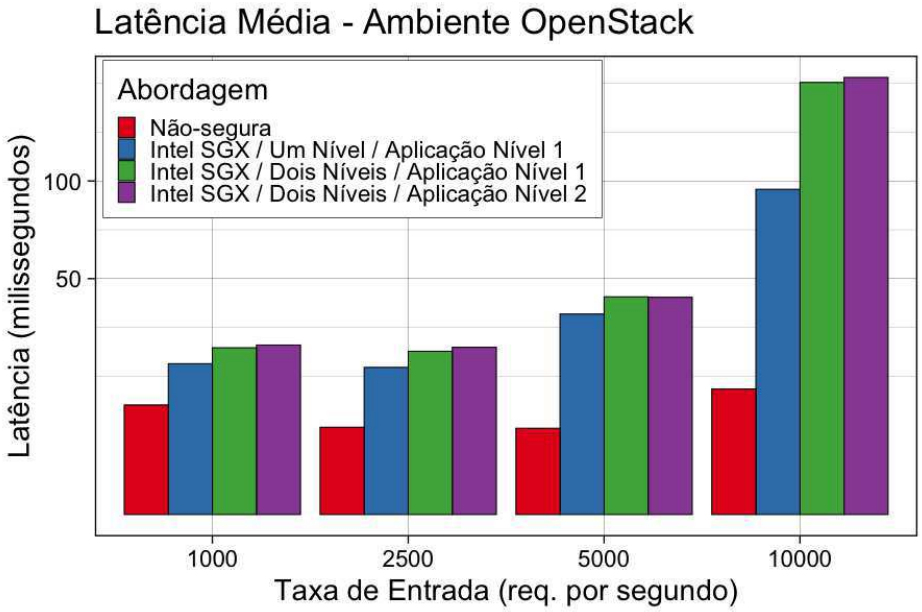

Figura 5.7: Média das latências para o experimento com o **Reencrypter**.

Devido à implementação do **SCONE**, todos os núcleos reservados ficam em utilização constante, independente do processamento necessário. Nesse experimento, reservou-se um núcleo para a instância do **LocationMatching**, mas é possível estimar que o uso do processamento chega a seu limite para a taxa de entrada de 400 mensagens por segundo, já que a latência tende a crescer indefinidamente acima disso. O uso de memória é também acrescido para que aplicações com o **SCONE** funcionem sem uma perda de desempenho, ficando na média de 1 GB, enquanto que a versão não-segura tem uma máxima de 114MB.

5.7 Análise de Ameaças

Ao utilizar Intel SGX, as ameaças identificadas na Seção 4.8.1 também são aplicáveis. Dentre elas, (i) **ataques de *side-channel***, que ameaçam a confidencialidade dos dados a partir da coleta de estatísticas do processador e pode usá-las para deduzir as informações sobreo software e os dados que estão sendo processados; (ii) vulnerabilidades **de software**, que ameaçam a confidencialidade e integridade dos dados a partir de falhas nos módulos em execução; (iii) **decifragem**, que ameaça a confidencialidade dos dados, a partir de ataques de força-bruta.

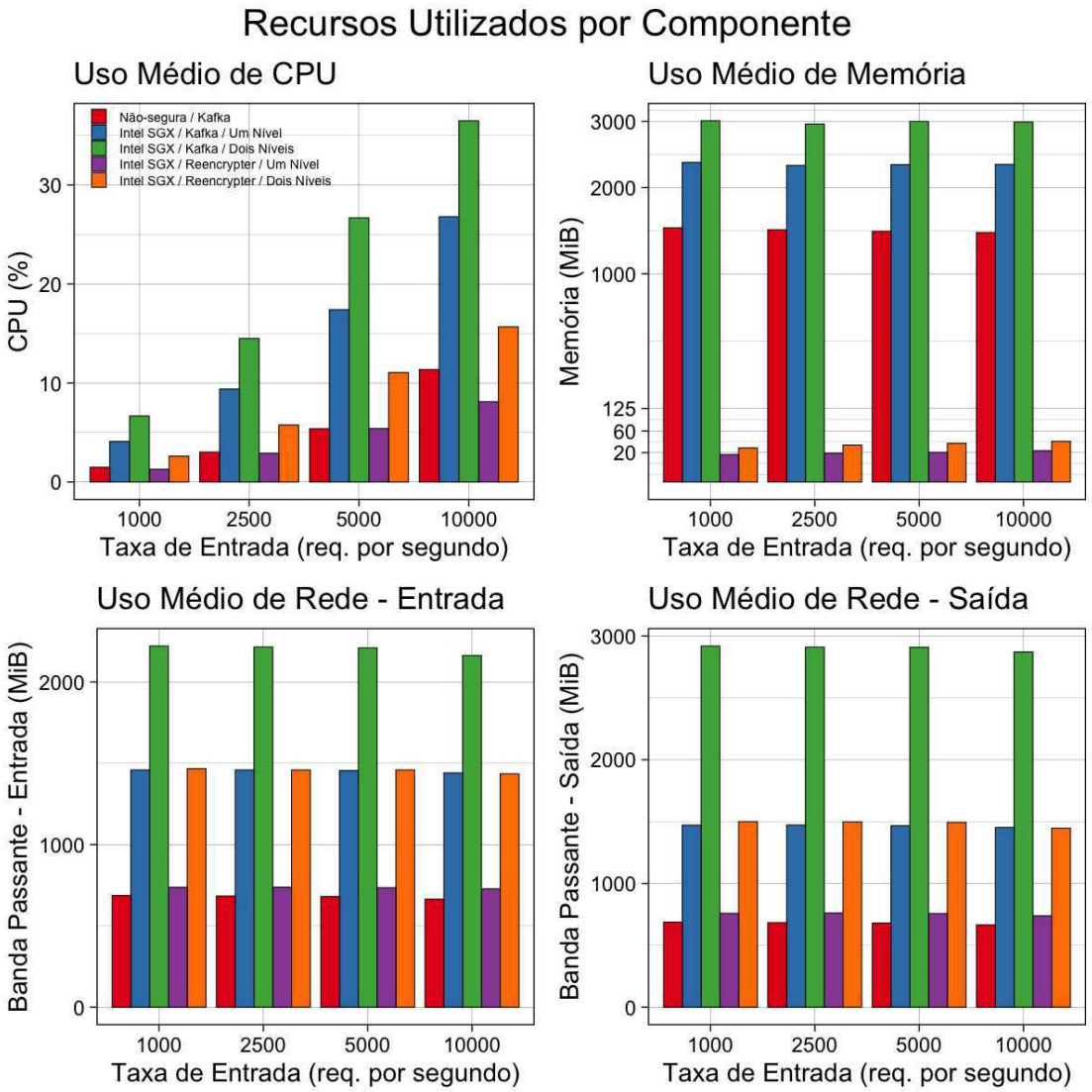

Figura 5.8: Recursos utilizados por componente para o experimento com o **Reencrypter**.

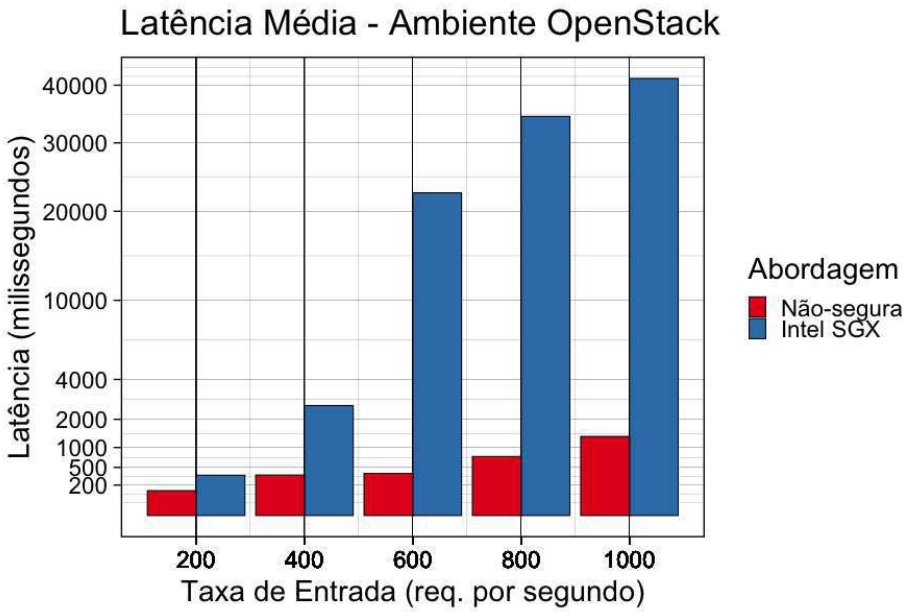

Figura 5.9: Média das latências para o experimento com o **LocationMatching**.

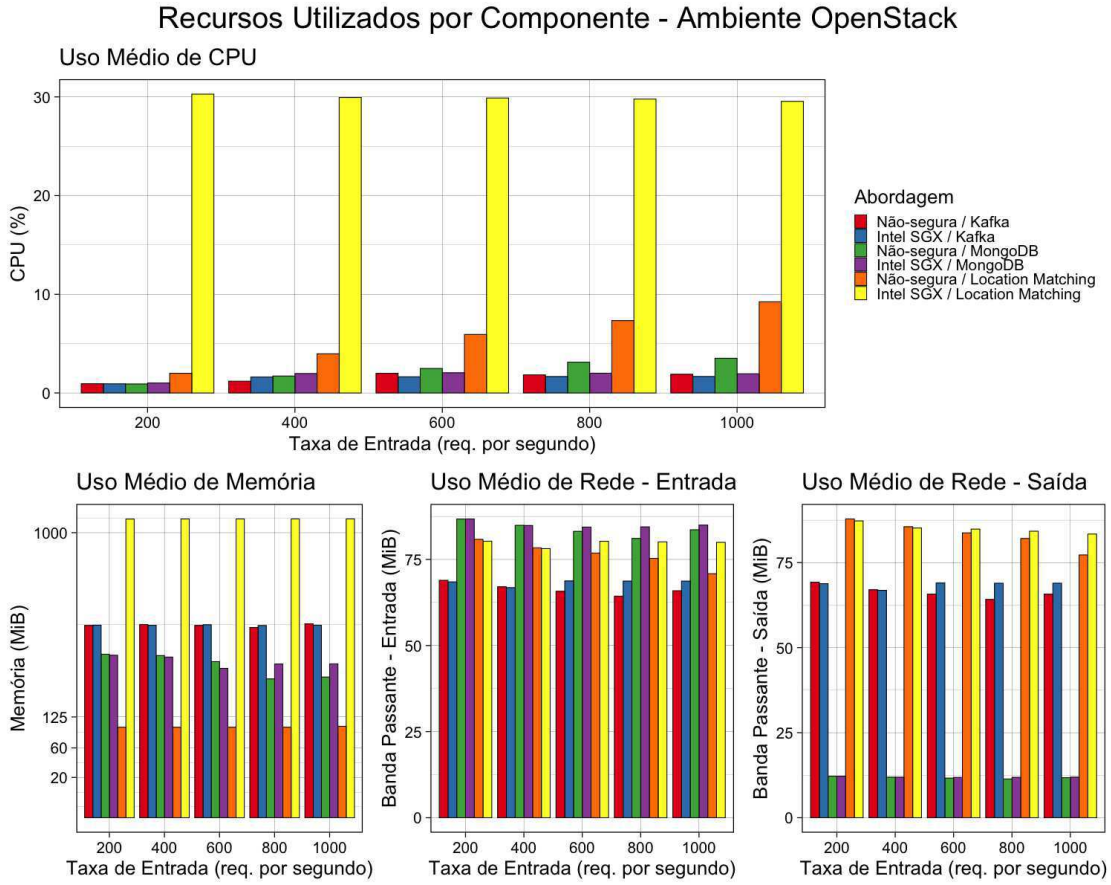

Figura 5.10: Recursos utilizados por componente para o experimento com o **LocationMatching**.

Capítulo 6

Avaliação

De acordo com o modelo arquitetural proposto nesta tese e os exemplos de aplicação aqui definidos, surgiu a necessidade de avaliação com desenvolvedores para elencar os pontos fortes e fracos deste modelo.

Esta avaliação teve como principal foco a usabilidade do modelo por times de desenvolvimento e uma revisão para escolha de métodos aplicáveis ao nosso favor.

6.1 Teste de Usabilidade

O principal objetivo de um teste de usabilidade é compreender como usuários vão vir a interagir com seu modelo, produto, arquitetura e outros. Sendo assim, espera-se que o retorno dado pelos seus avaliadores venha a corrigir falhas de implementação ou design do que foi proposto.

Em um teste de usabilidade padrão, o ambiente que as tarefas vão ser realizadas é controlado, seja por uma lista ordenada de tarefas ou sob condições definidas anteriormente.

Para avaliar a usabilidade de um sistema podemos aplicar uma metodologia chamada **Avaliação Heurística**: visibilidade do status do sistema, compatibilidade entre o sistema e o mundo real, controle e liberdade para o usuário, consistência e padrões, prevenção de erros, reconhecimento em lugar de lembrança, flexibilidade e eficiência de uso [86].

6.1.1 Protocolo *Think Aloud*

O tipo de avaliação realizado pelo protocolo *Think Aloud* é feito a partir de testes, ou seja, ele é aplicável nos mais diversos estágios do ciclo de vida de uma aplicação: design, desenvolvimento do código em si, testes e até a entrega do produto em si. Uma descrição simples dele é dada em seu nome, em uma tradução livre: Pensar alto.

Protocolos *Think Aloud* são usados para identificar o comportamento cognitivo do desempenho em tarefas enquanto estão usando uma tecnologia ou modelo e determinar como essa informação pode ser utilizada para facilitar a resolução de problemas.

Estes protocolos podem ser separados nas seguintes categorias:

1. **Concorrente:** os usuários são requisitados para pensar e falar abertamente ao mesmo tempo em que estão realizando as atividades determinadas.

2. **Retrospectivo:** usuários são convidados a lembrar o que eles estavam pensando ao realizar aquelas atividades em um momento anterior.

6.1.2 Entrevista

Diferente de métodos como o *Think Aloud* e o teste de usabilidade remoto, a entrevista não é um método de avaliação por teste mas sim por questionamento, a maior diferença é que, por haver o questionamento durante uma entrevista conseguimos obter boas informações sobre como o usuário se sentiu utilizando a solução, porém, isso torna os resultados da eficiência da solução meio obscuros, tendo em mente que uma entrevista é uma avaliação subjetiva.

6.1.3 Teste de Usabilidade Remoto

O teste de usabilidade remoto pode ser aplicado nos mais diversos campos, mas recentemente sua força é vista em testes de aplicações para usúarios finais. Suas aplicações, tal qual as do *Think Aloud*, podem ser vistas em diversos pontos do ciclo de desenvolvimento. Pede-se para que o usuário teste o software e relate o que ocorreu.

Nesse teste, o ambiente não é controlado para o tipo de execução. Logo, podem ocorrer resultados inesperados ou mal usos da ferramenta, além do fato de ser custoso lidar com

diversos membros que estão geograficamente distribuídos e da necessidade de criar um ambiente somente para gravação e acompanhamento destes.

Com essa revisão e tomando em consideração sugestões da banca avaliadora dessa tese, o método escolhido como mais adequado para o estudo foi o *Think Aloud*. Exemplos de artigos e teses que fazem uso deste método são abundantes e vão das mais diversas áreas de computação, a julgamento de ginástica artística e educação primária [87; 88; 89].

6.2 Metodologia

Na condução desse experimento, a questão de pesquisa a ser respondida foi:

- RQ_1: O modelo arquitetural proposto auxilia equipes de desenvolvimento a tratar problemas de escalabilidade e segurança, em termos de confidencialidade e integridade dos dados no desenvolvimento de aplicações?

Baseando-se na questão de pesquisa acima, a hipótese nula a ser refutada foi:

- H_0: Utilizar o modelo arquitetural não irá auxiliar no desenvolvimento de aplicações com requisitos de escalabilidade e segurança, especialmente em confidencialidade e integridade.

Devido à natureza dessa hipótese, conduziu-se um estudo utilizando o modelo retrospectivo do protocolo *Think Aloud* a fim de obter conhecimento sobre o que os desenvolvedores estavam pensando, quais eram seus objetivos e quais eram suas estratégias em todas as etapas.

Para avaliar os resultados obtidos, as métricas consideradas foram:

- M_1: Tratamento de requisitos de escalabilidade;
- M_2: Tratamento de requisitos de confidencialidade;
- M_3: Tratamento de requisitos de integridade;
- M_4: Prevenção contra ataques externos (de fora da infraestrutura de hospedagem);
- M_5: Prevenção contra ataques internos (de dentro da infraestrutura de hospedagem).

Com o objetivo de avaliar essas métricas, os participantes foram submetidos à modelagem de uma aplicação de contagem de pessoas em ambientes. Essa aplicação é detalhada nas Tabelas 6.1 e 6.2.

Tabela 6.1: Aplicação a ser modelada pelas equipes

Pacotes de broadcast são enviados por dispositivos para localizar redes sem fio próximas. Isso pode ser usado para estimar o número de pessoas em ambientes. Logo, considere que existem roteadores sem fio com o firmware modificável para enviar os MACs, *timestamps* e níveis de sinal de cada pacote recebido. Considere que podem existir diferentes ambientes com números de roteadores variados. Esse envio é feito a partir de requisições a um *web service*.

Tabela 6.2: Funcionalidades desejadas para a aplicações proposta

• Informar a aplicações de terceiros o número de pessoas em períodos de tempo arbitrários. Por exemplo, a cada 10 segundos, a cada minuto, a cada 10 minutos e a cada hora;
– 1.1. Deve ser possível revogar, até mesmo autenticado, o acesso de certas aplicações a períodos de tempo específicos;
• Informar se um dado MAC esteve em um certo ambiente há um tempo especificado pelo requisitante;
• Informar os horários que um MAC esteve em um certo ambiente entre um intervalo especificado pelo requisitante.

A amostra foi de duas equipes, **A** e **B**, de desenvolvimento. Cada equipe foi composta por uma média de dez desenvolvedores, porém, somente uma pessoa por equipe participou da avaliação. O critério de inclusão foi de equipes pequenas com baixo conhecimento acerca do desenvolvimento de softwares considerando aspectos de confidencialidade e proteção de dados.

A equipe **A** representou o grupo de controle, logo, não utilizou o modelo arquitetural para modelar a aplicação de contagem de pessoas. A equipe **B**, por sua vez, fez uso do modelo proposto.

A equipe **B** utilizou uma versão inicial do guia de uso do modelo para auxiliar a construção de arquiteturas seguras, que pode ser encontrado no apêndice A. Em seguida,

pediu-se um resumo do mesmo e que quaisquer comentários fossem armazenados em um documento. A equipe **B** teve dois dias para realizar essa etapa.

Em seguida, apresentou-se a aplicação a ser modelada dentro do prazo de uma semana. Para as duas equipes, relatou-se que a aplicação poderia ser hospedada em uma infraestrutura não confiável. Após a modelagem, as equipes realizaram a implementação com base nas suas respectivas arquiteturas dentro do prazo de um mês. Por fim, uma última reunião foi realizada com cada equipe para coletar os pensamentos após a implementação de fato.

6.3 Resultados

As arquiteturas modeladas pelas equipes **A** e **B** estão, respectivamente, nas Figuras 6.1 e 6.2. Além disso, as equipes produziram breves explicações sobre as arquiteturas, que se encontram nas Tabelas 6.4 e 6.5. Baseando-se na análise desses artefatos, verificou-se que a equipe **A** escolheu uma abordagem *serverless* [90] para receber as informações dos dispositivos conectados e uma abordagem cliente-servidor para agregar e informar as aplicações de terceiros. Já a equipe **B**, que fez uso do guia, utilizou o padrão **publish-subscribe**, de forma que um barramento é utilizado para manter as informações de todos os dispositivos conectados e o outro barramento, para manter as informações de forma agregada, sem identificar unicamente cada dispositivo – as aplicações de terceiros terão acesso a esse barramento.

De posse das duas arquiteturas, realizou-se a coleta das métricas M_1 a M_5 e os resultados podem ser vistos na Tabela 6.3. As justificativas para cada um dos resultados obtidos segue:

- M_1 **(tratamentos para escalabilidade)**: a arquitetura da equipe **A** utiliza contêineres Docker para se comunicar com aplicações de terceiros, o que facilita a escalabilidade e faz uso de **serverless** para a comunicação com os roteadores, ou seja, a cada requisição, uma função é executada a fim de salvar os dados dos dispositivos no banco. As funções podem ser executadas em um ou mais nós e isso é gerenciado pela plataforma a ser utilizada.

  A arquitetura da equipe **B** utiliza **publish-subscribe**, que auxilia na escalabilidade, pois permite que existam vários barramentos, produtores e consumidores. Nesse caso, a importância é que os agregadores e o próprio barramento sejam escaláveis, pois os roteadores já fazem parte de uma infraestrutura já existente;

- M_2 **(tratamentos para confidencialidade)**: a equipe **A** escolheu OAuth 2.0[1] para impedir acessos não autorizados. Esse padrão é bem estabelecido para permitir acesso de terceiros a aplicações e dados.

  A equipe **B** concentrou o tratamento da confidencialidade na aplicação intermediária. Essa aplicação provê chaves de acesso a produtores e consumidores após a atestação dos mesmos;

- M_3 **(tratamentos para integridade)**: como a equipe **A** fez uso exclusivamente de OAuth 2.0, não há indícios de que a integridade dos dados é garantida. Esse padrão provê garantias de confidencialidade, já que impede acessos indevidos, mas não trata da integridade dos dados que serão trafegados pelo sistema.

  A arquitetura da equipe **B** possui checagens de integridade com o uso da atestação. Além disso, todos os dados trafegados são criptografados, o que permite que as partes realizem checagem de integridade em todas as mensagens;

- M_4 **(prevenção de ataques externos)**: não foram encontradas evidências para prevenção de ataques externos na arquitetura da equipe **A**.

  Na arquitetura da equipe **B**, a aplicação intermediária provê os endereços de acesso somente às partes após atestação, o que reduz a chance de ataques externos. Com relação aos barramentos, embora eles sejam conhecidos somente pelos participantes atestados, é possível que seus endereços sejam descobertos por uma varredura na rede, mas, as mensagens que passam pelos barramentos são criptografadas com as chaves providas pela aplicação intermediária;

- M_5 **(prevenção de ataques internos)**: a arquitetura da equipe **A** não possui técnicas para prevenir ataques internos – ataques advindos do próprio ambiente de hospedagem.

  Para a arquitetura da equipe **B**, as técnicas utilizadas para ataques externos também são aplicáveis. Além disso, os dados desencriptados dos dispositivos são processados exclusivamente dentro de um enclave Intel SGX, reduzindo a superfície de ataques.

Com base nos resultados obtidos, há evidências de que a hipótese H_0 pode ser refutada, pois a equipe **B** conseguiu realizar a modelagem da arquitetura ao mesmo tempo em que

[1] `https://oauth.net/2/`

Tabela 6.3: Resultados obtidos a partir da modelagem realizada pelas duas equipes

Métrica	Equipe **A**	Equipe **B**
M_1 (tratamentos para escalabilidade)	Sim	Sim
M_2 (tratamentos para confidencialidade)	Sim	Sim
M_3 (tratamentos para integridade)	Não	Sim
M_4 (prevenção de ataques externos)	Não	Sim
M_5 (prevenção de ataques internos)	Não	Sim

endereçava corretamente os requisitos de escalabilidade, confidencialidade e integridade – a arquitetura proposta pela equipe de controle (**A**) não contemplou estratégias para garantir a integridade dos dados e/ou prevenir ataques externos e internos. Por outro lado, uma ameaça à validade do experimento concebido é que outras equipes deveriam ter participado, porém, isso custaria mais recursos humanos e financeiros, o que não foi possível dentro do escopo desse estudo.

O protocolo *think aloud* também foi útil para identificar algumas fraquezas do modelo, dentre elas a falta de exemplos e de maior detalhamento das definições empregadas. Isso resultou no aperfeiçoamento dos Capítulos 2 a 5, que são a base do guia. Os detalhes sobre as mudanças são listados a seguir:

- Cenários foram inseridos para detalhar operações recorrentes;
- Inclusão de novos requisitos funcionais, não-funcionais e decisões-chave;
- Detalhamento das aplicações apresentadas nos Capítulos 4 e 5.

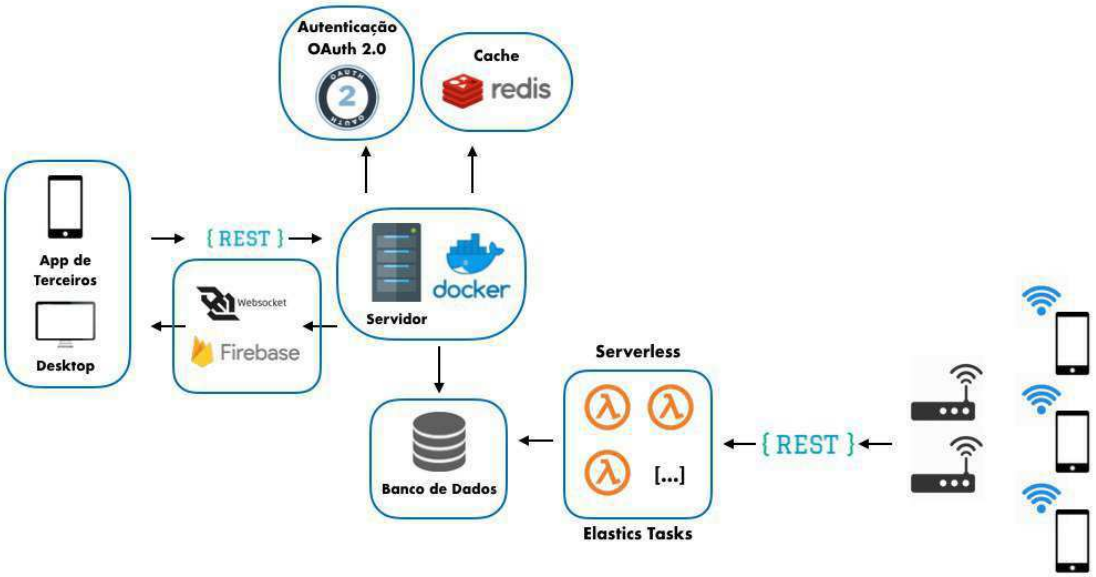

Figura 6.1: Diagrama da arquitetura de software – Equipe **A**

Tabela 6.4: Explicação da arquitetura – Equipe A

A arquitetura proposta visa atender principalmente os requisitos não-funcionais como escalabilidade, disponibilidade e segurança. Os dispositivos que estiverem enviando os sinais de localização de redes sem fio serão detectados pelos roteadores cujos firmwares foram adaptados para ao detectar tais sinais, enviar uma requisição para serviços serverless (e.g. o AWS Lambda) que irão registrar em um banco de dados a detecção destes dispositivos. A abordagem serverless foi escolhida para que seja escalável de acordo com a demanda de solicitação dos roteadores.

Também foi provido um servidor back-end para acesso a estes dados por terceiros. Deste modo, aplicativos podem se autenticar ao servidor e solicitar alguns dos dados ou se inscreverem para serem notificados de determinados eventos. Esta notificação pode ser realizada através de um Web Socket, que mantém um conexão ativa entre o servidor e o cliente, ou através do Firebase que irá realizar um ”push notification”para aplicativos móveis.

O servidor utilizando a abordagem de contêineres, no caso o Docker, para em caso de necessidade, subir novas instâncias de forma rápida e eficiente, ou seja, escalonamento horizontal. Cache com Redis é provido para melhora de performance para dados já processados/consultados. As instâncias evitam assim realizar muitos acessos simultâneos ao banco de dados.

A autenticação é controlada por um servidor OAuth 2.0 que gera tokens de autenticação com períodos de validade que são revogados automaticamente. Além disso, é possível solicitar a revogação de todos os tokens ativos.

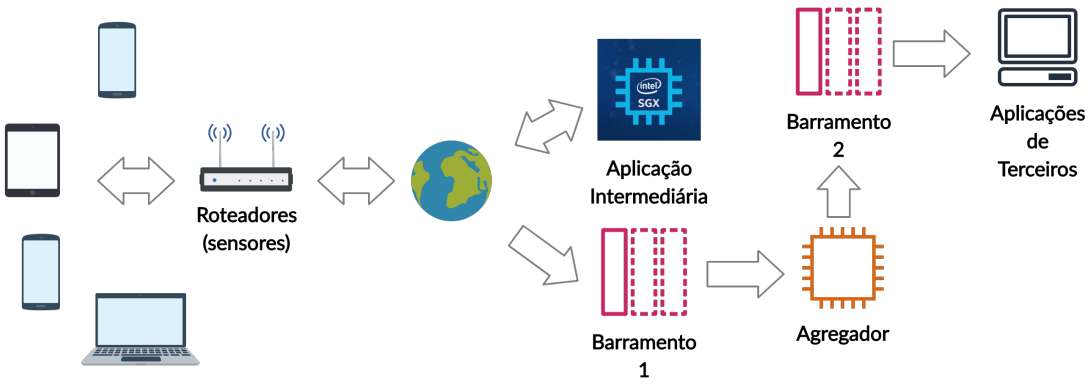

Figura 6.2: Diagrama da arquitetura de software – Equipe **B**

Tabela 6.5: Explicação da arquitetura – Equipe B

Vão existir dois barramentos: barramento 1 e barramento 2.
O barramento 1 é dividido por áreas. Cada área é um tópico. O barramento 1 é responsável receber as informações dos sensores (produtores) e enviar pros agregadores (consumidores).
O barramento 2 também é dividido por áreas. O barramento 2 é responsável por receber informações processadas dos agregadores (produtores) e mandar para as aplicações de terceiros (consumidores) que desejam receber as informações processadas.
Para uma aplicação ter acesso ao endereço de qualquer um dos barramentos é necessário passar pela aplicação segura intermediária. A aplicação intermediária é segura e é responsável por fazer a atestação das aplicações (produtores e consumidores). Se a atestação passar, a aplicação intermediária manda o endereço do barramento que a aplicação vai se comunicar, um certificado, uma chave para encriptar os dados (produtores) e/ou uma chave de desencriptar dados (consumidores) e o tópico que a aplicação vai produzir e/ou consumir.
Vão existir três tipos de aplicações se comunicando com os barramentos:

1. Agregadores que serão consumidores no barramento 1 para receber as informações dos roteadores e produtores no barramento 2 mandando as informações processadas. Os agregadores recebem uma chave para desencriptar os dados recebidos no barramento 1 e uma chave para encriptar os dados que irão enviar no barramento 2. Os agregadores são as únicas aplicações que funcionam como produtores e ouvintes ao mesmo tempo (produtor em um barramento e ouvinte em outro barramento);

2. Roteadores que são apenas produtores no barramento 1. Os roteadores recebem uma chave para encriptar os dados enviados ao barramento 1.

3. Aplicações de terceiros que querem receber as informações processadas dos agregadores. As aplicações de terceiros se comunicam ao barramento 2. As aplicações de terceiros recebem uma chave para desencriptar os dados enviados pelos agregadores.

Os barramentos só são acessados por aplicações que foram previamente atestadas pela aplicação intermediária. Isso garante que ninguém tem acesso sem antes ser atestado. Se a atestação falhar, a aplicação não vai saber o endereço do barramento e nem qual tópico escutar ou produzir.
Além da atestação, as aplicações produtoras e consumidoras precisam de certificados (gerados pela aplicação intermediária) com validade para poder escutar ou produzir informação.
Entende-se que as aplicações que vão se comunicar com os barramentos devem atestar a aplicação intermediária.

Capítulo 7

Considerações Finais

O crescimento do número de dispositivos interconectados nos leva a uma sociedade cada vez mais informada. A computação na nuvem surgiu como uma forma de facilitar o processamento desses dados. Porém, a sua falta de controle e possibilidades de ataques internos e externos aumentam as preocupações com a segurança dos dados.

Provedores de nuvem de empresas como Amazon, Google e outros não fornecem, de forma nativa, soluções para atestar os recursos fornecidos por eles. Vendo essa lacuna, um modelo de arquitetura de software foi proposto. Esse modelo foca em requisitos de escalabilidade, confidencialidade e integridade dos dados. Para desenvolver aplicações orientadas a eventos, com grandes volumes de dados, em alta velocidade e de produção contínua, o padrão de projeto *publish-subscribe* foi escolhido e, para atender requisitos de confidencialidade e de integridade, o modelo foi projetado para permitir tecnologias de hardware – como Intel SGX, AMD SEV e AMD SME – e/ou tecnologias de software – como criptografia homomórfica.

Para auxiliar o desenvolvimento desse modelo, duas aplicações em áreas distintas porém com características similares foram desenvolvidas. A primeira consistiu na coleta e processamento de dados de consumo de energia elétrica e, a segunda, consistiu na coleta e processamento de dados de localização dos usuários para permitir publicidade segmentada. Durante esse desenvolvimento, houve muitos aprendizados, permitindo então o refinamento do modelo. Por exemplo, a versão inicial não previa uma infraestrutura de chaves públicas e componentes para criptografar os dados em diferentes níveis, o que permite um controle maior para dados muitos sensíveis, enquanto mantém a flexibilidade para os dados menos

sensíveis.

Com o objetivo de identificar os ganhos e perdas ao tratar requisitos de segurança, foram realizados experimentos para ambas as aplicações. Com esses experimentos, foi possível identificar os percentuais e custos de recursos adicionais necessários para atender cargas de trabalho de forma segura. Inicialmente, um desafio encontrado foi de conseguir recursos de hardware para realizar os experimentos com Intel SGX, pois, no começo desse trabalho, a tecnologia estava em seus passos iniciais. Atualmente, o número de processadores que suporta essa tecnologia, bem como de provedores de serviço, vem crescendo a cada dia.

As arquiteturas preliminares produzidas para as aplicações de consumo de energia e de publicidade segmentada foram aprovadas por empresas do ramo, o que aumenta a confiança no modelo. Para aumentar ainda mais essa confiança, decidiu-se então que outros desenvolvedores utilizassem o modelo, o que levou à aplicação do protocolo *Think Aloud*. Como resultado, o modelo foi aperfeiçoado mais uma vez, com base no uso e *feedback* de duas equipes de desenvolvimento diferentes.

A principal contribuição desse trabalho para o estado da arte é a redução da complexidade para elaborar novas aplicações orientadas a eventos com preocupações de segurança, pois o modelo já contém um arcabouço de requisitos, decisões e elementos arquiteturais e foi validado por empresas a partir da construção de aplicações para resolver problemas reais.

Outra contribuição relevante é a avaliação detalhada das penalidades de desempenho impostas pelo uso de tecnologias de segurança, assim como as descobertas dos parâmetros a serem modificados a fim de reduzir perdas de desempenho. Ao escolher Intel SGX, a limitação de até 128 MB para a memória encriptada é um fator relevante para o desenvolvimento de aplicações. Além disso, para aplicações que executam em ambientes SCONE, o uso de máquinas virtuais permite a alocação de porções menores da memória encriptada, o que causa um desempenho superior em certos cenários, como aconteceu para a aplicação de publicidade segmentada. Na criptografia homomórfica, a complexidade de processamento depende da periodicidade de coleta dos dados, logo, períodos maiores podem se tornar viáveis, como foi estudado na aplicação do cálculo de consumo de energia.

Diante dos estudos realizados, algumas ramificações foram identificadas para exploração em trabalhos futuros. Essas linhas de investigação são descritas a seguir.

- **Estudar e aplicar mais requisitos relacionados à segurança:** o modelo contempla

requisitos de confidencialidade e integridade, porém, há mais requisitos de segurança a se considerar, como disponibilidade, em que os prejuízos financeiros causados por sistemas fora do ar por poucas horas pode chegar a milhões de dólares [91];

- **Aplicar o *Think Aloud* com uma amostra maior:** embora o *feedback* das duas equipes tenha sido bastante proveitoso, mais equipes poderiam dar uma maior solidez estatística à validação. No entanto, encontrar profissionais indicados e separar várias equipes de desenvolvimento para realizar experimentos de validação com o modelo não é tão simples, já que é uma atividade que pode consumir dias de trabalho;

- **Avaliar e comparar outras tecnologias de segurança dos dados:** esse trabalho se focou em Intel SGX devido à grande parcela de mercado que a Intel possui no mercado de servidores, em criptografia homomórfica por ser implantável em qualquer provedor de serviço e em SCONE por facilitar o uso de Intel SGX em diversas linguagens. Avaliar tecnologias como AMD SME/SEV e ARM TrustZone não foi prioritário, já que os provedores de serviço não disponibilizavam instâncias compatíveis até o final desse estudo. Por outro lado, há previsões de crescimento dos processadores AMD EPYC [92], além disso, grandes companhias, como a Amazon estão investindo em processadores ARM [93; 94];

- **Estudar e aplicar técnicas de ataque às soluções desenvolvidas e avaliar os impactos de segurança:** o foco principal do trabalho foi em elaborar um modelo para atender os requisitos de escalabilidade e segurança de aplicações orientadas a eventos e avaliá-lo em termos de desempenho. Aplicar ataques às tecnologias utilizadas não foi prioritário, pois já existiam muitos estudos sobre possíveis ataques, porém, faz-se necessário um conhecimento específico para encontrar falhas nos componentes do modelo, o que eleva a complexidade dessa linha investigativa;

- **Realizar análises do custo financeiro das alternativas propostas nessa tese em relação às abordagens atuais:** os provedores de serviço cobram de maneiras distintas, pelo potencial de processamento dos recursos, pela taxa de dados de entrada e saída, por períodos de tempo (por hora ou por mês), pelo uso de tecnologias de segurança e por outros fatores, logo, elaborar um modelo de custo não é uma tarefa

trivial. Além disso, essa análise de custo não foi essencial no contexto desse trabalho.

Bibliografia

[1] IBM MARKETING CLOUD. 10 key marketing trends for 2017. Disponível em: <https://ibm.co/36O9LsQ>. Acesso em: 11 jan. 2019.

[2] INTEL. A guide to the internet of things: How billions of online objects are making the web wiser. Disponível em: <https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html>. Acesso em: 11 jan. 2019.

[3] THE WASHINGTON POST. A security breach in india has left a billion people at risk of identity theft. Disponível em: <https://wapo.st/2K2E6dn>. Acesso em: 05 set. 2019.

[4] THE NEW YORK TIMES. Marriott hacking exposes data of up to 500 million guests. Disponível em: <https://www.nytimes.com/2018/11/30/business/marriott-data-breach.html>. Acesso em: 05 set. 2019.

[5] MANWORREN, N.; LETWAT, J.; DAILY, O. Why you should care about the target data breach. *Business Horizons*, Elsevier, v. 59, n. 3, p. 257–266, 2016.

[6] GEER, D. Are companies actually using secure development life cycles? *Computer*, IEEE, v. 43, n. 6, p. 12–16, 2010.

[7] PATTERSON, D. A. 20th century vs. 21st century c&c: the spur manifesto. *Commun. ACM*, v. 48, n. 3, p. 15–16, 2005.

[8] KUHN, R.; RAUNAK, M.; KACKER, R. It doesn't have to be like this: Cybersecurity vulnerability trends. *IT Professional*, IEEE, v. 19, n. 6, p. 66–70, 2017.

[9] BORITZ, J. E. Is practitioners' views on core concepts of information integrity. *International Journal of Accounting Information Systems*, Elsevier, v. 6, n. 4, p. 260–279, 2005.

[10] SILVA, L. et al. Agregação de dados na nuvem com garantias de segurança e privacidade. In: SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. *Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. [S.l.], 2016. p. 240–253.

[11] SILVA, L. V. et al. Security and privacy preserving data aggregation in cloud computing. In: *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*. [s.n.], 2017. p. 1732–1738. Disponível em: <https://doi.org/10.1145/3019612.3019795>.

[12] SILVA, L. V. et al. Security and privacy aware data aggregation on cloud computing. *J. Internet Services and Applications*, v. 9, n. 1, p. 6:1–6:13, 2018. Disponível em: <https://doi.org/10.1186/s13174-018-0078-3>.

[13] PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, ACM, v. 17, n. 4, p. 40–52, 1992.

[14] GARLAN, D.; SHAW, M. An introduction to software architecture. In: *Advances in Software Engineering and Knowledge Engineering*. [S.l.]: Publishing Company, 1993. p. 1–39.

[15] MARY, S.; DAVID, G. Software architecture: Perspectives on an emerging discipline. *Prentice-Hall*, 1996.

[16] ISO/IEC/IEEE. *Systems and software engineering — Architecture description*. 2011.

[17] BREDEMEYER CONSULTING. Motivating software architecture. Bloomington, 11 jul. 2012. Disponível em: <http://www.bredemeyer.com/why.htm>. Acesso em: 11 mai. 2013.

[18] SILVA, L. J. V. Mestrado em Ciência da Computação, *Documentação e Validação de Arquiteturas de Software: Uma Proposta Concreta*. 2013.

[19] EMERY, D. Applying IEEE 1471: Thoughts on an architecting process. 13 out. 2011. Disponível em: <http://www.iso-architecture.org/ieee-1471/applying-the-standard.html>. Acesso em: 11 mai. 2013.

[20] IIBA. *A guide to the Business Analysis Body of Knowledge (BABOK guide)*. 2. ed. Toronto: International Institute of Business Analysis, 2009.

[21] BREDEMEYER CONSULTING. Architectural requirements in the visual architecting process. Bloomington, 26 abr. 2013. Disponível em: <http://www.bredemeyer.com/ArchitectingProcess/ArchitecturalRequirements.htm>. Acesso em: 11 mai. 2013.

[22] CLEMENTS, P. et al. *Documenting Software Architectures: Views and Beyond*. 2. ed. [S.l.]: Addison-Wesley Professional, 2010.

[23] BARBOSA, G. M. G. *Um Livro-texto para o Ensino de Projeto de Arquitetura de Software*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2009.

[24] KRUCHTEN, P. B. The 4+ 1 view model of architecture. *IEEE software*, IEEE, v. 12, n. 6, p. 42–50, 1995.

[25] STONEBURNER, G. *NIST Special Publication 800-33, Underlying Technical Models for Information Technology Security-Recommendations of the National Institute of Standards and Technology*. [S.l.]: National Institute of Standards and Technology, Gaithersburg, USA December, 2001.

[26] PAULSEN, C.; PAULSEN, C.; TOTH, P. *Small business information security: The fundamentals*. [S.l.]: US Department of Commerce, National Institute of Standards and Technology, 2016.

[27] BARKER, E. et al. *NIST Special Publication 800-56B, Revision 2, Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography*. [S.l.]: National Institute of Standards and Technology, Gaithersburg, USA March, 2019.

[28] NIST, S. 800-32. *Introduction to Public Key Technology and the Federal PKI Infrastructure*, 2001.

[29] OLDEHOEFT, A. E. *Foundations of a Security Policy for Use of the National Research and Educational Network*. [S.l.]: US Department of Commerce, National Institute of Standards and Technology, 1992.

[30] RIVEST, R. L.; ADLEMAN, L.; DERTOUZOS, M. L. On data banks and privacy homomorphisms. *Foundations of secure computation*, v. 4, n. 11, p. 169–180, 1978.

[31] GENTRY, C. *A fully homomorphic encryption scheme*. Tese (Doutorado) — Stanford University, 2009.

[32] NAEHRIG, M.; LAUTER, K.; VAIKUNTANATHAN, V. Can homomorphic encryption be practical? In: ACM. *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. [S.l.], 2011. p. 113–124.

[33] BARBOSA, P. Y. S. *Privacy by evidence : a software development methodology to provide privacy assurance*. Tese (Doutorado) — Universidade Federal de Campina Grande, 2018.

[34] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In: ____. *Advances in Cryptology — EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 223–238.

[35] KAPLAN, D.; POWELL, J.; WOLLER, T. *AMD memory encryption*. 2016.

[36] GUAN, L. et al. Trustshadow: Secure execution of unmodified applications with arm trustzone. In: ACM. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. [S.l.], 2017. p. 488–501.

[37] NGABONZIZA, B. et al. Trustzone explained: Architectural features and use cases. In: IEEE. *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*. [S.l.], 2016. p. 445–451.

[38] JING, X.; JIAN-JUN, Z. A brief survey on the security model of cloud computing. In: IEEE. *2010 ninth international symposium on distributed computing and applications to business, engineering and science*. [S.l.], 2010. p. 475–478.

[39] PAHL, C.; LEE, B. Containers and clusters for edge cloud architectures–a technology review. In: IEEE. *2015 3rd international conference on future internet of things and cloud*. [S.l.], 2015. p. 379–386.

[40] HENDRE, A.; JOSHI, K. P. A semantic approach to cloud security and compliance. In: IEEE. *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.], 2015. p. 1081–1084.

[41] TIAN, Y. et al. A privacy preserving location service for cloud-of-things system. *Journal of Parallel and Distributed Computing*, Elsevier, v. 123, p. 215–222, 2019.

[42] ALMORSY, M.; GRUNDY, J.; MÜLLER, I. An analysis of the cloud computing security problem. *arXiv preprint arXiv:1609.01107*, 2016.

[43] KIM, Y.-J. et al. A secure decentralized data-centric information infrastructure for smart grid. *IEEE Communications Magazine*, IEEE, v. 48, n. 11, p. 58–65, 2010.

[44] KIM, Y.-J. et al. Sedax: A scalable, resilient, and secure platform for smart grid communications. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 30, n. 6, p. 1119–1136, 2012.

[45] ZHANG, J.; LI, Q.; SCHOOLER, E. M. ihems: An information-centric approach to secure home energy management. In: IEEE. *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*. [S.l.], 2012. p. 217–222.

[46] ZHAO, Y. et al. Secure pub-sub: Blockchain-based fair payment with reputation for reliable cyber physical systems. *IEEE Access*, IEEE, v. 6, p. 12295–12303, 2018.

[47] GRAVINA, R. et al. Cloud-based activity-aaservice cyber–physical framework for human activity monitoring in mobility. *Future Generation Computer Systems*, Elsevier, v. 75, p. 158–171, 2017.

[48] HAMDAQA, M.; LIVOGIANNIS, T.; TAHVILDARI, L. A reference model for developing cloud applications. In: *CLOSER*. [S.l.: s.n.], 2011. p. 98–103.

[49] SAMPAIO, L. et al. Secure and privacy-aware data dissemination for cloud-based applications. In: ACM. *Proceedings of the10th International Conference on Utility and Cloud Computing*. [S.l.], 2017. p. 47–56.

[50] BELOKOSZTOLSZKI, A. et al. Role-based access control for publish/subscribe middleware architectures. In: ACM. *Proceedings of the 2nd international workshop on Distributed event-based systems*. [S.l.], 2003. p. 1–8.

[51] PIRES, R. et al. Secure content-based routing using intel software guard extensions. In: *Proceedings of the 17th International Middleware Conference*. New York, NY, USA: ACM, 2016. (Middleware '16), p. 10:1–10:10. ISBN 978-1-4503-4300-8. Disponível em: <http://doi.acm.org/10.1145/2988336.2988346>.

[52] ARNAUTOV, S. et al. Pubsub-sgx: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems. In: IEEE. *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. [S.l.], 2018. p. 123–132.

[53] SUN, A. et al. One quantifiable security evaluation model for cloud computing platform. In: IEEE. *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*. [S.l.], 2018. p. 197–201.

[54] LIANG, H. et al. Architectural protection of trusted system services for sgx enclaves in cloud computing. *IEEE Transactions on Cloud Computing*, IEEE, 2019.

[55] WANG, C. et al. Security issues and requirements for internet-scale publish-subscribe systems. In: IEEE. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. [S.l.], 2002. p. 3940–3947.

[56] ERKIN, Z.; TSUDIK, G. Private computation of spatial and temporal power consumption with smart meters. *Proceedings of the 10th international conference on Applied Cryptography and Network Security*, p. 561–577, 2012. ISSN 03029743.

[57] ARMEL, K. C. et al. Is disaggregation the holy grail of energy efficiency? the case of electricity. *Energy Policy*, Elsevier, v. 52, p. 213–234, 2013.

[58] ANDERSON, R.; FULORIA, S. On the security economics of electricity metering. In: CITESEER. *WEIS*. [S.l.], 2010.

[59] GREVELER, U. et al. Multimedia content identification through smart meter power usage profiles. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING (WORLDCOMP). *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*. [S.l.], 2012. p. 1.

[60] BUSOM, N. et al. Efficient smart metering based on homomorphic encryption. *Computer Communications*, Elsevier, v. 82, p. 95–101, 2016.

[61] ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. In: SPRINGER. *Workshop on the Theory and Application of Cryptographic Techniques*. [S.l.], 1984. p. 10–18.

[62] CRAMER, R.; GENNARO, R.; SCHOENMAKERS, B. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, Wiley Online Library, v. 8, n. 5, p. 481–490, 1997.

[63] DWORKIN, M. J. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Gaithersburg, MD, United States, 2007.

[64] ANATI, I. et al. Innovative technology for cpu based attestation and sealing. In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*. [S.l.: s.n.], 2013. v. 13.

[65] MCKEEN, F. et al. Innovative instructions and software model for isolated execution. In: *HASP@ ISCA*. [S.l.: s.n.], 2013. p. 10.

[66] PAVERD, A.; MARTIN, A.; BROWN, I. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *University of Oxford, Tech. Rep*, 2014.

[67] BARBOSA, M. et al. Foundations of hardware-based attested computation and application to sgx. In: IEEE. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. [S.l.], 2016. p. 245–260.

[68] RESEARCHGATE. Typically, how many houses does a distribution transformer supply in a residential area? Disponível em: <https://www.researchgate.

net/post/Typically_how_many_houses_does_a_distribution_transformer_supply_in_a_residential_area>. Acesso em: 15 set. 2019.

[69] SILVA, R. Mestrado em Ciência da Computação, *Desafios no desenvolvimento de aplicações seguras usando Intel SGX*. 2018.

[70] XU, Y.; CUI, W.; PEINADO, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. Washington, DC, USA: [s.n.], 2015. p. 640–656.

[71] BRASSER, F. et al. Software grand exposure: SGX cache attacks are practical. *CoRR*, abs/1702.07521, 2017.

[72] OLEKSENKO, O. et al. Varys: Protecting {SGX} enclaves from practical side-channel attacks. In: *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. [S.l.: s.n.], 2018. p. 227–240.

[73] BONEH, D. *Basic Control Hijacking Attacks*. Disponível em: <http://crypto.stanford.edu/cs155/lectures/02-ctrl-hijacking.pdf>. Acesso em: 14 ago. 2017.

[74] RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, ACM, v. 21, n. 2, p. 120–126, 1978.

[75] POLLARD, J. M. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, v. 32, p. 918–924, 1978. ISSN 0025–5718.

[76] POMERANCE, C. A tale of two sieves. *Notices Amer. Math. Soc*, v. 43, p. 1473–1485, 1996.

[77] BONEH, D. Twenty years of attacks on the rsa cryptosystem. *NOTICES OF THE AMS*, v. 46, p. 203–213, 1999.

[78] WIENER, M. J. Cryptanalysis of short rsa secret exponents. *IEEE Trans. Inf. Theor.*, Piscataway, NJ, USA, v. 36, n. 3, p. 553–558, set. 2006. ISSN 0018-9448.

[79] COPPERSMITH, D. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology*, v. 10, n. 4, p. 233–260, Sep 1997. ISSN 1432-1378.

[80] HASTAD, J. Solving simultaneous modular equations of low degree. *SIAM J. Comput.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 17, n. 2, p. 336–341, abr. 1988. ISSN 0097-5397.

[81] COPPERSMITH, D. et al. Low-exponent rsa with related messages. In: *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*. Berlin, Heidelberg: Springer-Verlag, 1996. (EUROCRYPT'96), p. 1–9.

[82] BONEH, D.; DURFEE, G.; FRANKEL, Y. An attack on rsa given a small fraction of the private key bits. In: *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*. London, UK, UK: Springer-Verlag, 1998. (ASIACRYPT '98), p. 25–34.

[83] PLUMMER, J. et al. *The online advertising playbook: Proven strategies and tested tactics from the advertising research foundation*. [S.l.]: John Wiley & Sons, 2007.

[84] JANSEN, B. J.; MOORE, K.; CARMAN, S. Evaluating the performance of demographic targeting using gender in sponsored search. *Information Processing & Management*, Elsevier, v. 49, n. 1, p. 286–302, 2013.

[85] AUBLIN, P.-L. et al. Talos: Secure and transparent tls termination inside sgx enclaves. *Imperial College London, Tech. Rep*, v. 5, 2017.

[86] NIELSEN, J. Enhancing the explanatory power of usability heuristics. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1994. (CHI '94), p. 152–158. ISBN 0-89791-650-6. Disponível em: <http://doi.acm.org/10.1145/191666.191729>.

[87] EYE-TRACKING retrospective think-aloud as a novel approach for a usability evaluation. *International Journal of Medical Informatics*, v. 129, p. 366 – 373, 2019. ISSN 1386-5056.

[88] EXPLORING the use of think aloud within Women's artistic gymnastics judging education. *Psychology of Sport and Exercise*, v. 40, p. 135 – 142, 2019. ISSN 1469-0292.

[89] PROFILING upper primary school students' self-regulated learning through self-report questionnaires and think-aloud protocol analysis. *Learning and Individual Differences*, v. 70, p. 155 – 168, 2019. ISSN 1041-6080.

[90] AMAZON WEB SERVICES. Building applications with serverless architectures. Disponível em: <https://aws.amazon.com/pt/lambda/serverless-architectures-learn-more/>. Acesso em: 12 set. 2019.

[91] BUSINESS INSIDER. The massive aws outage hurt 54 of the top 100 internet retailers — but not amazon. Disponível em: <https://www.businessinsider.com/aws-outage-hurt-internet-retailers-except-amazon-2017-3>. Acesso em: 07 set. 2019.

[92] MUJTABA, H. Amd all set to capture 10% of the total server cpu market by 2020, report indicates – will secure more deals with 7nm epyc cpus due to strong price / performance leadership. Disponível em: <https://wccftech.com/amd-epyc-server-cpu-7nm-market-share-2020-report/>. Acesso em: 01 ago. 2019.

[93] AMAZON WEB SERVICES. New - ec2 instances (a1) powered by arm-based aws graviton processors. Disponível em: <https://amzn.to/33vg8iD>. Acesso em: 02 ago. 2019.

[94] AMAZON WEB SERVICES. Aws lança exemplos pré-configurados adicionais para o freertos no armv8-m. Disponível em: <https://amzn.to/36ICjE9>. Acesso em: 02 ago. 2019.

Apêndice A

Guia de Um Modelo Arquitetural para Processamento Seguro de Dados

1. Introdução

A proposta deste guia é apresentar um modelo de arquitetura de software com foco em requisitos de privacidade e segurança, que faz uso de: tecnologias de hardware e software para assegurar sigilo no processamento de dados críticos. Visto que cada aplicação tem suas peculiaridades, diferentes arquiteturas podem ser desenvolvidas de forma específica seguindo o modelo proposto, bem como diferentes técnicas podem ser aplicadas.

O restante do documento está organizado da seguinte forma: o **Capítulo 2** contém os termos e definições necessários para o entendimento do modelo. O **Capítulo 3** é composto pelo modelo arquitetural, citando todos os seus elementos e como ele pode ser aplicado em problemas reais. Por fim, o **Capítulo 4** contém os passos necessários para utilização do modelo.

2. Termos e definições

2.1. Arquiteturas de Software

Uma arquitetura de software é caracterizada pelo conjunto de componentes de um sistema, como eles se comportam, como se relacionam e pelas diretrizes que nortearam o desenvolvimento dos mesmos.

Com relação a sua utilidade, podemos citar [1; 2]:

- **Comunicação:** Desenvolvedores, gerentes de desenvolvimento e arquitetos; gerentes de projeto; testadores; documentadores e até mesmo clientes precisam entender o sistema, sendo assim, quanto mais alto nível, melhor a comunicação;
- **Contexto do sistema:** Os desenvolvedores e futuros mantenedores precisam entender o sistema como um todo. Em grandes sistemas, é difícil para os desenvolvedores compreenderem todos os seus os detalhes de forma eficiente. A falta do entendimento de estruturas relacionadas pode reduzir a qualidade do sistema em termos de desempenho, segurança e disponibilidade;
- **Alocação de trabalho:** É possível particionar o trabalho dos desenvolvedores de forma eficiente a partir da decomposição do sistema em estruturas relativamente independentes;

- **Redução dos custos de manutenção e evolução:** Arquiteturas podem minimizar esses custos com a antecipação de possíveis mudanças no sistema, garantindo que o projeto do mesmo facilite tais mudanças e documentando como elas podem afetar os elementos arquiteturais envolvidos;
- **Reúso e integração com sistemas legados e softwares de terceiros:** Uma arquitetura pode ser projetada para permitir e facilitar o reúso de componentes, arcabouços, bibliotecas, sistemas legados e softwares de terceiros.

No contexto desse trabalho, uma arquitetura de software é composta por partes interessadas, requisitos, decisões arquiteturais, pontos de vista e visões arquiteturas. Cada um desses elementos serão discutidos a seguir.

2.2. Partes Interessadas

Uma parte interessada (ou *stakeholder*) é qualquer entidade – seja ela um indivíduo, time ou organização – que possui interesse em um sistema. Seguem abaixo alguns tipos de partes interessadas:

- **Usuários finais:** aqueles que irão, de fato, utilizar o sistema;
- **Clientes:** responsáveis pela contratação do sistema. Em muitos projetos, são também os usuários finais;
- **Desenvolvedores:** são responsáveis pela implementação do sistema de acordo com as restrições impostas pela arquitetura e outras partes;
- **Testadores:** partes que irão verificar se o sistema se comporta de maneira esperada em diversos cenários;
- **Implantadores:** partes encarregadas de instalar o sistema e mantê-lo operacional;
- **Gerentes de desenvolvimento:** coordenam a equipe de desenvolvimento, participam da escolha de tecnologias e orientam os programadores;
- **Arquitetos:** indivíduos responsáveis pela definição da arquitetura e, mais formalmente, pela produção da descrição arquitetural [3].

2.3. Requisitos

Um requisito pode ser definido como [2; 4]:

- Uma condição ou capacidade necessária para uma parte interessada resolver um problema ou atingir um objetivo;
- Uma condição ou capacidade que deve ser alcançada ou possuída por uma solução, ou componente de solução, para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos;
- Uma representação documentada de uma condição ou capacidade como em (1) ou (2).

Os requisitos podem ser: **do negócio** – objetivos, metas e necessidades da empresa; **de partes interessadas**; **da solução** – **funcionais** ou **não-funcionais**, os primeiros descrevem o comportamento e a informação que a solução irá gerenciar, os últimos descrevem qualidades que a solução deve possuir; **de transição**, descrevem capacidades que a solução deve possuir com o objetivo de facilitar a transição do estado atual da organização para um estado futuro desejado [2].

Os **requisitos de solução** citados anteriormente podem ser **requisitos arquiteturais**, isso acontece quando os mesmos capturam funcionalidades essenciais para o sistema; atingem vários elementos arquiteturais; possuem exigências rigorosas à arquitetura ou envolvem comunicação e sincronização com sistemas externos [5].

2.4. Decisões, Pontos de Vista e Visões Arquiteturais

Uma **decisão** pode ser considerada **arquitetural** quando a mesma se propõe a alcançar um ou mais atributos de qualidade do sistema [6; 7].

Um **ponto de vista** é um arcabouço conceitual que define elementos, conexões e técnicas que compõem uma visão arquitetural, além de especificar seu propósito de acordo com seus interessados [7]. Já as **visões** expressam a arquitetura de um sistema ou de parte dele a partir da perspectiva de um conjunto de interesses relacionados [8]. Visto isso, um ponto de vista é utilizado para elaborar diversas **visões arquiteturais**.

O uso dessas visões facilita o projeto, a documentação e o entendimento da **arquitetura**, já que é possível abstrair detalhes dependendo de cada **visão**.

3. Modelo Arquitetural

O modelo arquitetural se baseia no padrão de projeto *publish-subscribe*. Nesse padrão, as entidades que enviam mensagens são chamadas de *publishers* – eventualmente referenciados como **produtores** no decorrer desse documento. Por sua vez, os **consumidores** (*subscribers*) se registram para receber mensagens. Para reduzir o acoplamento e aumentar escalabilidade, um produtor não se conecta diretamente aos consumidores, já que as mensagens são trocadas por meio de um ou mais **barramentos**.

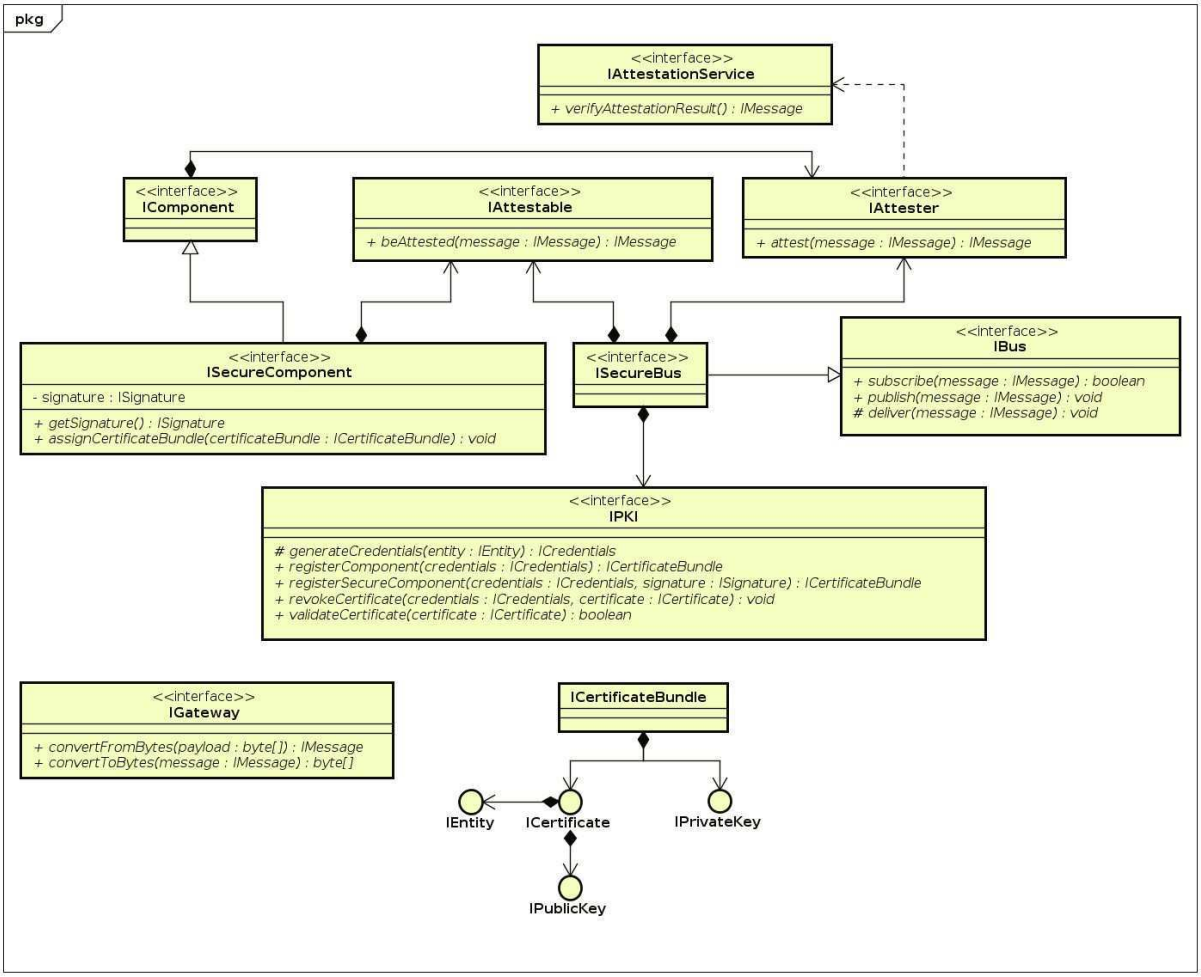

Figura 1: Modelo UML simplificado

Esse modelo pode ser aplicado em situações em que dados confidenciais são coletados e devem ser agregados para anonimizar e produzir informações significativas. Por exemplo, nas eleições, dispositivos eletrônicos podem ser usados para enviar, periodicamente, resultados parciais para agregadores na nuvem, a fim de calcular o resultado final da votação. As identidades e resultados parciais dos eleitores não vazariam porque os agregadores seguros lidariam com essas informações em um ambiente isolado.

No modelo proposto, ilustrado na Figura 1, **IComponent** pode ser um produtor e/ou consumidor conectado ao barramento **IBus**. Quando há necessidade de conversão do formato de mensagens, um **IGateway** pode ser utilizado. Devido aos requisitos de segurança e privacidade, introduziu-se o conceito de um barramento seguro **ISecureBus**, que, além de gerenciar a troca de mensagens, possui mecanismos de autenticação, para garantir a identidade dos componentes registrados ao enviar e receber de mensagens. A autenticação deve ser implementada pelo componente **PKI**, também responsável pela criação e revogação de credenciais de acesso.

As credenciais de acesso (**ICredentials**) são dadas às entidades, que são responsáveis pelo desenvolvimento de componentes a serem conectados no barramento seguro. Para o registro de componentes no barramento, é necessário fornecer as credenciais. Esse registro resulta em um novo **certificado** (**ICertificateBundle**), composto pelas chaves pública e privada, além de conter uma validade. Esse certificado será utilizado para **assinar** mensagens enviadas ao barramento seguro.

Componentes seguros devem, obrigatoriamente, ser atestáveis (**IAttestable**). Em outras palavras, devem existir maneiras para garantir que o componente possua a mesma implementação do momento da sua concepção. O barramento seguro é responsável pela atestação desses componentes no momento do seu registro e/ou periodicamente – com o uso do componente **IAttester**. Com o objetivo de validar a atestação, uma consulta é efetuada a um serviço de atestação – representado pelo componente **IAttestationService**. Nesse modelo, é necessário que o serviço de atestação seja **confiável**.

A seguir, os requisitos funcionais e não-funcionais serão explicitados, assim como as decisões e elementos arquiteturais.

3.1. Requisitos Funcionais

3.1.1. Gerar Credenciais

Para permitir o registro de novos componentes, seguros ou não, no barramento, é necessário que a entidade mantenedora – um desenvolvedor ou uma empresa, por exemplo – possua credenciais de acesso. Tais credenciais devem ser informadas à entidade de forma segura, para evitar vazamento de informações.

3.1.2. Bloquear Credenciais

Permite o bloqueio temporário das credenciais de uma entidade, impedindo que os componentes mantidos por ela acessem o barramento.

3.1.3. Desbloquear Credenciais

Desabilita o bloqueio temporário das credenciais de uma entidade.

3.1.4. Remover Credenciais

É possível remover credenciais para desabilitar o acesso de todos os componentes de uma entidade, de forma permanente.

3.1.5. Registrar Componentes

Para o registro de componentes, deve-se informar as credenciais de acesso da entidade. Em seguida, o **barramento seguro** valida os dados informados e, caso positivo, retorna um **certificado** para ser associado ao componente.

3.1.6. Registrar Componentes Seguros

Para um componente seguro, deve-se informar, além das credenciais, a sua assinatura. Essa assinatura pode ser associada ao desenvolvedor ou à versão específica do código que é executado por esse componente.

3.1.7. Revogar Certificados

Além de gerar, há casos em que um certificado seja revogado, por motivos de desativação de componentes, por exemplo. Desse modo, deve-se ignorar mensagens enviadas ao barramento que são assinadas por um certificado revogado.

3.1.8. Validar Certificados

Para identificar se uma mensagem do barramento deve ser aceita ou não, deve-se validar o certificado que a assinou. Caso ela não seja válida, o barramento irá recusar a mensagem. Deve-se verificar se o

certificado se encontra no banco de dados, se a sua data está no período de validade e se o mesmo não fora revogado.

3.1.9. Recuperar Assinatura

Componentes seguros devem possuir assinaturas, que serão informadas ao barramento seguro, juntamente com o certificado, para autenticar esse componente e permitir o seu registro.

3.1.10. Associar Certificado

Como certificados têm validade e podem ser trocados dinamicamente, os componentes seguros devem permitir sua troca, mediante atestação do código.

3.1.11. Realizar Inscrição

Componentes podem se inscrever em **tópicos** do **barramento** para receber mensagens. Visto isso, a mensagem de inscrição enviada deve ser assinada com o seu certificado. Para componentes seguros, sua assinatura também deve compor a mensagem. No ato dessa inscrição, a assinatura e o certificado devem ser validados.

3.1.12. Publicar Mensagem

Antes da publicação efetiva de uma mensagem, verifica-se se ela possui um certificado válido. Se possuir, deve-se entregar a mensagem para cada componente válido que está inscrito no tópico no qual ela é endereçada. Caso contrário, ela é descartada.

3.1.13. Iniciar Atestação

Antes de se inscrever em um tópico, componentes podem atestar o barramento, para se certificar de que o código em execução é o esperado. Quando os componentes são seguros, o barramento também pode fazer a atestação deles, formando uma **atestação de duas vias**.

3.1.14. Validar Atestação

Deve existir uma entidade confiável na qual o resultado da atestação, como por exemplo, um **token**, será enviado e validado.

3.2. Requisitos Não-Funcionais

3.2.1. Acesso Restrito – Segurança

Todas as operações no sistema devem ser feitas somente após autenticação.

3.2.2. Bloqueio de acessos inválidos – Segurança

Caso existam várias tentativas inválidas de acesso dentro de um intervalo estabelecido, o usuário e IP de origem devem ser bloqueados.

3.2.3. Comunicação Segura – Segurança / Privacidade

Todos os dados trafegados pela rede devem ser encriptados.

3.2.4. Processamento Seguro – Segurança / Privacidade

Todos os dados críticos – que possuem dados privados aos seus possuidores – devem ser processados de forma segura, seja ela via hardware ou software.

3.2.5. Armazenamento Seguro – Segurança / Privacidade

Dados críticos devem ser armazenados, em disco, de forma segura. Pode-se criptografar de forma que apenas o uso combinado de várias chaves possa revelá-los.

3.3. Decisões Arquiteturais

3.3.1. Chaves de acesso devem possuir atributos

As chaves de acesso podem conter atributos como validade, nível de acesso e outros. O nível define os dados que o possuidor da chave poderá acessar. Isso permite maior flexibilidade mas torna a implementação mais complexa.

3.3.2. Realizar atestação antes de troca e processamento de dados

Para que o envio e processamento de dados coletados seja seguro, é necessário que exista uma atestação remota para garantir que a comunicação não seja interceptada até que chegue no local seguro. Nesse local, os dados serão desencriptados, processados e, possivelmente, encriptados novamente para novas comunicações.

3.3.3. Criptografar dados com chave negociada no processo de atestação

Como resultado da atestação remota, uma chave derivada pode ser obtida para que um canal seguro seja estabelecido. Além do uso de SSL, os dados podem ser criptografados para garantir uma maior segurança ao canal.

3.3.4. Utilizar tecnologias de hardware e software para processamento e armazenamento de dados

Para permitir o processamento e armazenamento seguro de dados, processadores com suporte a Intel SGX, ARM TrustZone, AMD SME ou AMD SEV podem ser utilizados.

3.4. Elementos Arquiteturais

A arquitetura é composta por: **IMessage**, **IComponent**, **ISecureComponent**, **IAttestable**, **IAttester**, **IAttestationService**, **IBus**, **ISecureBus**, **IPKI**, **IGateway** e **ICertificateBundle**. Eles serão detalhados nas próximas seções, juntamente com suas interfaces.

3.4.1. Mensagens

Para que os componentes se comuniquem, tudo deve ser feito a partir de troca de mensagens – **IMessage**, sejam elas diretas ou indiretas. Uma mensagem possui um conjunto de atributos e ela é, na maioria dos casos, encriptada com alguma chave, seja ela derivada do processo de atestação ou do certificado de quem a assina.

3.4.2. Componentes Seguros & Não-seguros

Um componente dessa arquitetura pode ser **seguro** ou **não-seguro**, e pode ser um **produtor**, **consumidor** ou um **serviço**. Quando o mesmo é seguro, além de um certificado, ele possui uma assinatura, que vai depender da tecnologia aplicada. Com relação a sua natureza, quando ele é um **produtor**, mensagens devem ser enviadas a um tópico do barramento; quando é um **consumidor**, mensagens devem ser consumidas de um tópico e quando é um **serviço**, ele fornece operações para outros componentes da arquitetura. Não há restrições para componentes que possuam múltiplas naturezas, ao contrário da sua segurança, que é excludente.

Operações

As operações abaixo são obrigatórias, porém, cada componente pode ter operações a mais para, por exemplo, fornecer serviços.

- **ISecureComponent.getSignature**: ver Seção 3.1.9;
- **ISecureComponent.assignCertificateBundle**: ver Seção 3.1.10.

3.4.3. Componentes para Atestação

Um componente do tipo **IAttester** pode realizar atestação de outros componentes do tipo **IAttestable**. O processo de atestação é validado com o uso de um serviço de atestação, representado pelo **IAttestationService**. Por exemplo, no caso do Intel SGX, esse componente é implementado pelo **Intel Attestation Service**. Ademais, um componente **IAttester** também pode ser utilizado quando o dispositivo que necessita realizar essa operação não possua capacidade, seja ela de hardware ou software.

Operações

- **IAttestable.beAttested**: o componente **IAttester** monta mensagens para que esse componente as consuma e envie uma resposta. Esse processo pode ter mais de uma etapa, que é identificada por atributos da mensagem;
- **IAttester.attest**: envio de desafio para o componente **IAttestable**. Esse desafio pode conter parâmetros adicionais, que serão enviados por quem está iniciando a atestação;

- **IAttestationService.verifyAttestationResult**: valida o processo de atestação. Essa entidade deve ser confiável por quem inicia o processo de atestação.

3.4.4. Barramentos

Um **barramento** – **IBus** – é responsável por aceitar inscrições de consumidores, aceitar envio de mensagens por produtores e por encaminhá-las para os interessados. Nessa arquitetura, o diferencial se encontra no **barramento seguro** – **ISecureBus**, que além das operações citadas, também é responsável por criar e revogar certificados e credenciais, autenticar, atestar consumidores e validar mensagens.

Operações

- **IBus.subscribe**: ver seção 3.1.11;
- **IBus.publish**: ver seção 3.1.12.
- **IBus.deliver**: entrega a mensagem aos consumidores interessados.

3.4.5. Infraestrutura de Chaves Públicas

O componente **IPKI** é responsável pela Infraestrutura de Chaves Públicas (*Public Key Infrastructure* – PKI). Ele é responsável pela geração, bloqueio e desbloqueio de credenciais; registro de componentes seguros e não-seguros; revogação e validação de certificados.

Um **ICertificateBundle** contém uma chave privada (**IPrivateKey**) e um certificado (**ICertificate**), que possui uma validade e uma chave pública (**IPublicKey**). Esse componente é associado a uma entidade, que pode ser um desenvolvedor, ou uma empresa, por exemplo.

Operações

- **IPKI.generateCredentials**: ver Seção 3.1.1;
- **IPKI.lockCredentials**: ver Seção 3.1.2;
- **IPKI.unlockCredentials**: ver Seção 3.1.3;
- **IPKI.registerComponent**: ver Seção 3.1.5;
- **IPKI.registerSecureComponent**: ver Seção 3.1.6;
- **IPKI.revokeCertificate**: ver Seção 3.1.7;
- **IPKI.validateCertificate**: ver Seção 3.1.8;

3.4.6. Gateways

Quando for necessária a conversão dos formatos de dados, é possível utilizar um componente do tipo **IGateway** para efetuar essa operação. Ele também pode ser utilizado para agrupar e enviar conjuntos de mensagens de vários produtores.

Operações

Essas operações são exemplos de como um **gateway** pode ser implementado.

- **convertFromBytes**: a partir de um conjunto de bytes, uma **mensagem** é criada;
- **convertToBytes**: a partir de uma mensagem, um *array* de bytes é criado.

4. Uso do Modelo

Após a compreensão do modelo arquitetural – incluindo seus **requisitos funcionais** e **não-funcionais**, **componentes** e **decisões arquiteturais**, é possível **instanciar uma arquitetura de software** para embasar o desenvolvimento de um sistema para processamento seguro de dados. Para tal, é necessário realizar as seguintes atividades:

1. Levantar requisitos;
2. Mapear os componentes do modelo para o problema real;
3. Listar decisões arquiteturais;
4. Realizar implementação com base na arquitetura instanciada.

Recomenda-se que essas atividades sejam realizadas de forma **iterativa e incremental**, ou seja, na necessidade de refinamentos, deve-se retornar a passos anteriores antes de seguir em frente.

Nas seções seguintes, cada uma das atividades será detalhada.

4.1. Levantamento de Requisitos

O primeiro passo a ser realizado é o levantamento de requisitos do sistema, para que seja entendido **o que** – requisitos funcionais – o sistema deve fazer e **como** deve fazer – requisitos não-funcionais. O modelo

já possui **um conjunto de requisitos**, mas nem todos precisam ser aplicados na **arquitetura instanciada**, então, isso também deve ser endereçado nessa etapa.

O levantamento de requisitos é importante para entender o escopo do problema e deve ser mitigada no começo, pois mudanças nos requisitos podem ser caras e até mesmo inviabilizar o projeto.

4.2. Mapeamento de Componentes com o Modelo

Uma vez que os requisitos foram definidos, mesmo que parcialmente, deve-se mapear os componentes propostos no modelo para resolver o problema real. Para tal, os componentes instanciados na arquitetura serão baseados nos propostos no modelo.

Assim como os requisitos, nem todos os componentes arquiteturais propostos serão utilizados. Todavia, por ser um modelo baseado em *publish-subscribe*, há componentes que são obrigatórios na **arquitetura instanciada**. Com isso, **IComponent** e **IBus** estarão nessa arquitetura.

4.3. Listagem de Decisões Arquiteturais

Nessa etapa, deve-se listar decisões arquiteturais – aquelas que impactam em vários pontos do sistema. Nela, as tecnologias de hardware e software devem ser definidas para que os requisitos sejam atendidos.

4.4. Realizar implementação

Após a definição **do que** deve ser feito, **como** deve ser feito e quais **tecnologias** serão aplicadas, é possível iniciar a **implementação**.

Nas etapas anteriores, é crucial que seja validado se as tecnologias escolhidas realmente atendem aos requisitos definidos, pois é bastante custosa uma troca de tecnologia durante o andamento do sistema.

5. Bibliografia

[1] Motivating software architecture. 11 jul. 2012.

[2] Leandro J. V. Silva. Documentação e validação de arquiteturas de software: Uma ˜ proposta concreta, 2013.

[3] David Emery. Applying IEEE 1471: Thoughts on an architecting process. 13 out. 2011.

[4] IIBA. A guide to the Business Analysis Body of Knowledge (BABOK guide). International Institute of Business Analysis, Toronto, 2 edition, 2009.

[5] Architectural requirements in the visual architecting process. 26 abr. 2013.

[6] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional, 2 edition, 2010.

[7] Guilherme Mauro Germoglio Barbosa. Um livro-texto para o ensino de projeto de arquitetura de software. Master's thesis, Universidade Federal de Campina Grande, 2009.

[8] ISO/IEC/IEEE. Systems and software engineering — architecture description, 2011.