UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS CRATEÚS

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WERMESON ROCHA DA SILVA

ESTUDO COMPARATIVO DO IMPACTO DA SEGURANÇA EM AMBIENTES DE *MOBILE CLOUD COMPUTING*

CRATEÚS - CEARÁ

2019

WERMESON ROCHA DA SILVA

ESTUDO COMPARATIVO DO IMPACTO DA SEGURANÇA EM AMBIENTES DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Me. Francisco Anderson de Almada Gomes.

CRATEÚS - CEARÁ
2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S584e Silva, Wermeson Rocha da.
Estudo Comparativo do Impacto da Segurança em Ambientes de Mobile Cloud Computing / Wermeson Rocha da Silva. – 2020.
63 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Ciência da Computação, Crateús, 2020.
Orientação: Prof. Me. Francisco Anderson de Almada Gomes.

1. Mobile Cloud Computing. 2. Dispositivos Móveis. 3. Offloading. 4. Segurança. 5. Computação em Nuvem. I. Título.

CDD 004

WERMESON ROCHA DA SILVA

ESTUDO COMPARATIVO DO IMPACTO DA SEGURANÇA EM AMBIENTES DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Me. Francisco Anderson de Almada Gomes (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Filipe Fernandes dos Santos Brasil de Matos
Universidade Federal do Ceará (UFC)

Prof. Me. Adriano Lima Cândido
Faculdade Vale do Salgado (FVS)

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Primeiramente a Deus por todas as bençãos ao longo da minha vida.

Aos meus pais, Manoel José e Maria Audene, pelo amor, carinho, apoio incondicional e por me fornecer força sempre que fraquejei durante essa caminhada.

Ao meu irmão, Werton Fabian, pelo apoio, incentivo e também por me ajudar em questões pessoais durante esse tempo.

À minha tia, Desterro Rocha, pelos cuidados, atenção, incentivo e também pelos momentos de descontração que me ajudaram bastante nessa caminhada.

Ao meu orientador, professor Anderson Almada, por todo o acompanhamento e tutoria para a realização deste trabalho bem como os conselhos e incentivos que e me fortaleceram durante esse processo.

Ao professor Arnaldo e professoras Lilian e Lisieux pelo apoio durante toda minha graduação e também pela preocupação que demonstraram possuir tanto no âmbito pessoal como acadêmico sobre mim, disponibilizando muitas vezes de seu tempo para conversar e me fazer se sentir melhor.

Aos professores Filipe e Adriano que compõem a banca examinadora e certamente contribuirão para o enriquecimento da solução desenvolvida.

À Universidade Federal do Ceará, pela oportunidade de fazer a graduação e ser minha casa durante todo esse período.

À todos os amigos e colegas que acompanharam e compartilharam experiências durante a graduação. Em especial ao Matheus Sampaio, Matheus Cavalcante, Davi Barros, Marcus Vinicius e João Paulo.

À todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“A melhor maneira de prever o futuro é criá-lo.”

(Peter Drucker)

RESUMO

A evolução da internet, dos dispositivos e das aplicações móveis trouxeram novas necessidades para as pessoas, como a de estar sempre conectado seja com o intuito de se comunicar, de se manter informado ou mesmo para o entretenimento pessoal com a utilização de aplicativos. Porém, mesmo com todos os avanços, os dispositivos móveis continuam limitados em recursos e as aplicações modernas demandam cada vez desses recursos para oferecer melhores experiências ao usuário. Buscando amenizar este problema, a *Mobile Cloud Computing* se utiliza da Computação em Nuvem para "expandir"os recursos dos dispositivos móveis através do *offloading*, onde o *offloading* diz respeito a migração de dados e/ou processamento do dispositivo móvel para um ambiente remoto (Nuvem). Entretanto este processo traz problemas de segurança, pois os dados podem ser migrados pela rede sem qualquer tipo de proteção. Visando contornar este problema existem soluções utilizando técnicas de criptografia para garantir a segurança dos dados. Este trabalho visa conduzir um estudo analítico sobre os impactos causados no desempenho das aplicações móveis pela utilização da criptografia na operação de *offloading*, utilizando a métrica de tempo de processamento. Como principal resultado, tem-se que quanto maior o objeto de requisição para o *offloading* maior será o tempo gasto com a comunicação e consequentemente o tempo total em todo o processo e que dependendo do algoritmo criptográfico, a influência dele é insignificante para o processo como um todo.

Palavras-chave: Mobile Cloud Computing. Dispositivos Móveis. Offloading. Segurança. Computação em Nuvem

ABSTRACT

The evolution of the internet, mobile devices and applications have brought new needs to the people, such as being always connected, whether for communication, stay informed or even for personal entertainment with the use of applications. But even with all the advancements, mobile devices remain resource-constrained, and modern appli1cations are increasingly demanding these features to deliver better user experiences. To mitigate this problem, Mobile Cloud Computing uses Cloud Computing to "expand"the capabilities of mobile devices through offloading, where offloading concerns data migration and / or mobile device processing for a remote environment (Cloud). However, this process has security problems, as data can be migrated over the network without any protection. To work around this problem there are solutions using encryption techniques to ensure data security. This work aims to conduct an analytical study on the impacts on mobile application performance by using encryption in offloading using the processing time metric. The main result is that the larger the request object for offloading, the greater the time spent on communication and consequently the total time in the whole process and that depending on the cryptographic algorithm, its influence is insignificant. for the process as a whole.

Keywords: Mobile Cloud Computing. Mobile Devices. Offloading. Security. Cloud Computing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3DES	*Triple Data Encryption Standard*
ACM	*Association for Computing Machinery*
AES	*Advanced Encryption Standard*
CAOS	*Context Acquisition and Offloading System*
DES	*Data Encryption Standard*
e.g.	*Por exemplo*
ECC	*Elliptic Curve Cryptography*
GPS	*Global Positioning System*
GREat	*Grupo de Redes de Computadores Engenharia de Software e Sistemas*
i.e.	*Isto é*
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
MCC	*Mobile Cloud Computing*
NIST	*National Institute of Standards and Technology*
NSA	*National Security Agency*
OMN	*Opportunistic Mobile Networks*
RSA	*Rivest-Shamir-Adleman*
SecOMN	*Secured Opportunistic Mobile Network*
SSP	*Storage Service Provider*

SUMÁRIO

1 INTRODUÇÃO

A tecnologia está cada vez mais presente na sociedade e os dispositivos móveis (*Por exemplo (e.g.)*, *smartphones, tablets*) tornaram-se objetos essenciais para as pessoas (CISCO, 2017). Vidal (2015) afirma que os *smartphones* são parte expressiva da rotina da grande maioria dos indivíduos das gerações Y e Z. As aplicações móveis atualmente têm se tornado abundantes em várias categorias, como por exemplo: entretenimento, saúde, negócios, viagens e notícias (FERNANDO *et al.*, 2013). Este crescimento deve-se ao fato da procura e consumo desse tipo de conteúdo ter aumentado com a evolução dos dispositivos móveis, pois a comodidade da utilização dos mesmos e a grande quantidade de aplicações que podem ser executadas nesses dispositivos, facilita a busca pelas mais diversas informações, tanto do próprio usuário do dispositivo, como também acesso a notícias que o usuário gostaria de ler.

De acordo com Perez (2010), a evolução dos dispositivos móveis e das aplicações é evidente em vários estudos. Especificamente, em Juniper (2014), é apresentado um estudo de abril de 2014 que revelou que o valor dos pagamentos globais via dispositivos móveis alcançaram em torno de $507 bilhões de dólares no ano, um aumento de quase 40% em relação aos anos anteriores. Outro estudo encontrado em Juniper (2019) afirma que os dispositivos móveis são responsáveis por mais de 50% dos usuários ingressos digitais em 2019.

1.1 Justificativa

Apesar de todos os avanços nos dispositivos móveis e na infraestrutura capaz de prover serviços para os mesmos, esses dispositivos ainda contam com algumas limitações, como por exemplo a escassez de recursos e energia finita como afirmado em Fernando *et al.* (2013).

Trabalhos como Qureshi *et al.* (2011), afirmam que *"A demanda por aplicativos móveis está aumentando, exigindo que mais recursos sejam fornecidos para tornar a experiência do usuário mais aprimorada. [...]"*, pois os dispositivos móveis são utilizados para várias tarefas atualmente (e.g., comunicação por meio de chamadas de voz ou vídeo, entretenimento pessoal com o uso de aplicativos de redes sociais, jogos, edição e aplicação de filtros em imagens). Algumas estratégias foram desenvolvidas com o intuito de mitigar esses problemas, como por exemplo a *Mobile Cloud Computing* (MCC), que consiste na adaptação dos recursos da

Nuvem para aumentar o poder computacional e/ou armazenamento de dados dos dispositivos móveis (SHIRAZ *et al.*, 2013; FERNANDO *et al.*, 2013).

Assim, a MCC traz diversos benefícios, tais como: a extensão da vida útil da bateria, melhoria na capacidade de armazenamento de dados e do poder de processamento e melhora da confiabilidade dos dispositivos (DINH *et al.*, 2013). Dentre as estratégias utilizadas na MCC, pode-se citar o *offloading* de dados e/ou processamento que diz respeito a migração dos mesmos do dispositivo móvel para o ambiente da Nuvem, onde lá serão armazenados, no caso da migração de dados, ou processados e retornados ao dispositivo móvel no caso da migração de processamento. Essa estratégia, apesar de benéfica, traz consigo problemas inerentes a transmissão de dados do dispositivo móvel para a Nuvem, como por exemplo, a privacidade e a segurança desses dados e também a conectividade necessária entre as entidades envolvidas.

Este trabalho tem por finalidade estudar as estratégias presentes na literatura, em artigos publicados a respeito das áreas de MCC e Segurança da Informação (mais especificamente a parte de criptografia) colhidos em repositórios online (*Institute of Electrical and Electronics Engineers* (IEEE)/*Association for Computing Machinery* (ACM)) e a partir deste estudo analisar os impactos causados pela utilização de técnicas criptográficas em ambientes de MCC, com o intuito de mitigar os problemas de segurança principalmente no que diz respeito aos dados transmitidos através do *Offloading*.

1.2 Objetivos

1.2.1 Objetivo Geral

Avaliar os impactos causados pela adição da criptografia na MCC em termos de tempo de processamento e selecionar uma estratégia que se mostra capaz de mitigar o problema da segurança nesse ambiente sem causar grandes perdas de desempenho.

1.2.2 Objetivos Específicos

- Analisar as estratégias de segurança na migração de dados;
- Analisar o impacto da segurança no desempenho de tarefas computacionais no *offloading*;
- Realizar um estudo comparativo entre as estratégias utilizadas e selecionar a melhor delas.

1.3 Organização do Trabalho

Este trabalho está organizado em 6 (seis) capítulos. O primeiro tratou de uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, os objetivos e as contribuições deste trabalho.

O Capítulo 2 trata da fundamentação teórica, apresentando os conceitos importantes adotados na pesquisa. Dentre eles, pode-se citar: o paradigma de MCC e a Segurança.

O Capítulo 3 apresenta os trabalhos relacionados a solução proposta. Foram analisados trabalhos que propõem segurança em ambientes de MCC e apresentem pelo menos o seguinte critério: criptografia de dados como segurança desses ambientes.

O Capítulo 4 apresenta a proposta do Trabalho de Conclusão de Curso (TCC), onde é descrito a metodologia que será utilizada no mesmo, bem como seus objetivos a serem alcançados.

O Capítulo 5 apresenta uma aplicação desenvolvida durante o trabalho e os resultados obtidos para a validação do mesmo.

Por fim, o último capítulo descreve as conclusões do trabalho e as possíveis melhorias a serem consideradas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A *Mobile Cloud Computing* é um paradigma de integração entre a *Computação em Nuvem* e a *Computação Móvel*, o qual serve para mitigar problemas nos dispositivos móveis utilizando os serviços de Nuvem e o *framework* CAOS apresenta este paradigma na prática. O outro tema desse trabalho é a *Segurança da Informação*, mais especificamente a *Criptografia*, em que dado a necessidade de enviar os dados desses dispositivos para a Nuvem, os meios utilizados para a transmissão devem ser seguros, garantindo a confidencialidade desses dados. As seções seguintes abordam cada um dos temas mencionados.

2.1 Computação em Nuvem

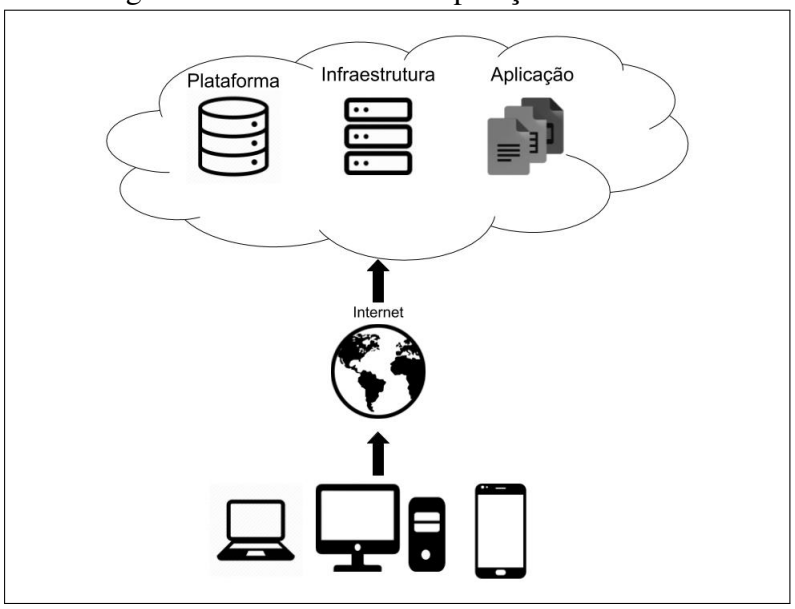
É uma área da Computação que vem ganhando grande notoriedade por apresentar possibilidades de acesso à recursos (*e.g.*, processamento, armazenamento) por meio da conexão com à Internet, tornando possível armazenar e acessar dados a qualquer momento e em qualquer lugar, sem necessidade de instalar outros programas (DURAO *et al.*, 2014).

Segundo Mell *et al.* (2011), a Computação em Nuvem é um modelo que permite acesso de rede onipresente e conveniente a um conjunto compartilhado de recursos de Computação configuráveis que podem ser rapidamente provisionados e liberados com esforço mínimo de gerenciamento. A Figura 1 ilustra o modelo citado com a representação de *notebooks, desktops* e *smartphones* utilizando a conexão com a internet para obter acesso à recursos de computação.

Além da definição de Computação em Nuvem, é interessante compreender o modelo de Nuvem que é composto de cinco características essenciais: (i) **Autoatendimento sob demanda** que corresponde a possibilidade de um usuário poder requisitar recursos da Nuvem quando necessitar sem que precise interagir com seu provedor de serviços; (ii) **Amplo acesso à rede** devido ao fato dos recursos estarem disponíveis na rede, tornando-os acessíveis de qualquer dispositivo com conexão à internet; (iii) **Agrupamento de recursos** onde os recursos são agrupados, atribuídos e reatribuídos dinamicamente com o intuito de atender os consumidores, abstraindo dos mesmos informações (*e.g.*, localização exata do recurso); (iv) **Elasticidade rápida** representando a capacidade da Nuvem aumentar ou diminuir a quantidade de recursos de acordo com a demanda, transparecendo ao usuário que a quantidade de recursos é ilimitada;

e (v) **Serviço medido** que controla a utilização dos serviços de forma automática, proporcionando o pagamento pela quantidade de serviço consumido, aumentando o controle e fornecendo transparência tanto para o provedor quanto para o consumidor dos recursos.

Figura 1 – Modelo de Computação em Nuvem



Fonte – Próprio autor

A Computação em Nuvem é essencial a proposta deste trabalho, pois os dados que serão migrados do dispositivo móvel terão como destino a Nuvem, que posteriormente podem ser migrados para ou acessados de volta pelo dispositivo móvel. Assim, a Nuvem irá funcionar tanto como um ambiente capaz de processar tarefas do dispositivo móvel como também armazenar informações deste. Para uma melhor compreensão dos dispositivos móveis, a seguir, é apresentado os conceitos de Computação Móvel.

2.2 Computação Móvel

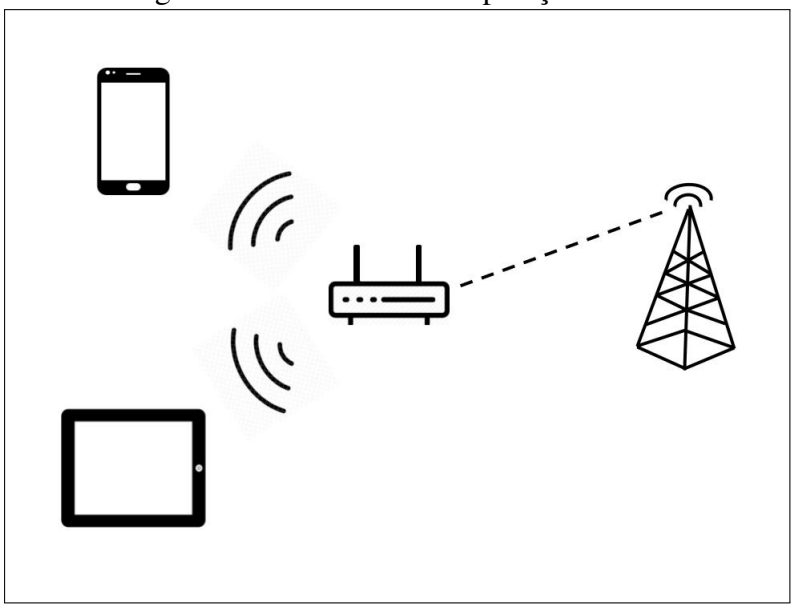
Segundo Gupta (2008), a Computação Móvel pode ser definida como a combinação da infraestrutura de comunicação sem fio e dispositivos portáteis de Computação que permite aos usuários acessar as informações e colaborar com outras pessoas enquanto estão em movimento. A Figura 2 apresenta uma abstração da mesma, representando dispositivos de Computação Móvel (e.g., *tablets*, *smartphones*) conectados a uma rede sem fio por meio de um *Access-Point* [1]. A expansão desse paradigma deu-se dentre outros motivos pela necessidade atual da comunicação

[1] Dispositivo de rede utilizado para estender a cobertura de redes de internet, conectado via cabo a um roteador e distribui o sinal *Wi-Fi*

e troca de informações entre as pessoas.

De acordo com Forman e Zahorjan (1994), os avanços na tecnologia permitem que computadores portáteis sejam equipados com interfaces sem fio, permitindo comunicação em rede, mesmo quando móveis. O paradigma da Computação Móvel fornece aos usuários móveis uma flexibilidade na comunicação entre eles e permite acesso contínuo aos serviços e recursos da rede. A combinação de rede e mobilidade proporciona novas oportunidades de aplicativos e serviços, como *softwares* para a resolução da mobilidade urbana (e.g., Uber, 99).

Figura 2 – Modelo de Computação Móvel



Fonte – Próprio autor

Gupta (2008) afirma que apesar dos benefícios citados, existem limitações desse paradigma. Com o passar do tempo os dispositivos móveis passaram por muitas evoluções, no entanto, elas ainda não foram suficientes para acompanhar as grandes demandas da atualidade. Algumas limitações como de energia, devem ser abordadas em termos do que um aplicativo deve ou não executar, pois grandes processamentos consomem muita energia. Outra limitação diz respeito a rede, em que as redes sem fio possuem largura de banda limitada, alta latência e desconexão frequente devido a limitações de energia, espectro disponível e mobilidade. Mais uma limitação da Computação Móvel associada à rede é a restrição de segurança, pois o tráfego dos dados muitas vezes ocorre sem nenhuma medida protetiva que impeça o acesso não autorizado as informações em trânsito, o que trazer problemas relacionados a privacidade e integridade desses dados.

Baseado no que foi discutido sobre os temas de Computação em Nuvem e Computação Móvel, a próxima Seção apresenta um novo paradigma, que consiste na integração entre os dois temas, chamado de *Mobile Cloud Computing*.

2.3 *Mobile Cloud Computing (MCC)*

A MCC é um paradigma de integração da Computação Móvel e Computação em Nuvem e tem como objetivo se utilizar de serviços em Nuvem para mitigar as limitações da Computação Móvel (*Isto é* (i.e.), desempenho computacional e consumo energético), tendo sido definido pela primeira vez no trabalho de Qureshi *et al.* (2011), mas atualmente ainda não existe um consenso entre os pesquisadores sobre seu conceito.

Para Shiraz *et al.* (2013), MCC é o mais recente paradigma computacional prático, que estende a visão da Computação Utilitária (do inglês, *Utility Computing*) de Computação em Nuvem para aumentar os recursos limitados dos dispositivos móveis, onde os serviços são fornecidos por demanda. Já para Sanaei *et al.* (2014), MCC é uma rica tecnologia de Computação Móvel que utiliza recursos elásticos unificados de Nuvens variadas e tecnologias de rede com funcionalidade irrestrita, armazenamento e mobilidade para servir uma multidão de dispositivos móveis em qualquer lugar, a qualquer momento através da *Internet* independentemente de ambientes heterogêneos e plataformas baseadas no princípio *pay-as-you-use*[2]. Já em Rahimi *et al.* (2014), a MCC é definida como o conjunto de técnicas que usam recursos de Nuvem para capacitar aplicativos com o objetivo principal de proporcionar uma experiência melhor para usuários móveis cujos dispositivos possuem recursos e capacidades limitados, como computação, armazenamento e bateria. Dentre o conjunto de técnicas desenvolvidas para a MCC, a seguir será detalhado melhor sobre o *offloading*, que é uma técnica que se relaciona diretamente com o presente trabalho.

2.3.1 Offloading

De acordo com Santos *et al.* (2017), o *offloading* (descarregamento) é um procedimento onde é feito a migração de dados e/ou processamento dos dispositivos móveis para a Nuvem, com o objetivo de "aumentar" o poder de processamento e armazenamento, bem como

[2] Termo referente a forma de pagamento pelo consumo, ou seja, ao invés de pagar um valor fixo mensal o contratante só pagará pelo recurso consumido mensalmente

reduzir o consumo de energia destes dispositivos.

Em Fernando *et al.* (2013), é afirmado que existem basicamente dois tipos principais de *offloading*, de processamento e de dados. O *offloading* de processamento refere-se a entrega de um processamento que demanda um maior poder computacional para outro ambiente de execução (*e.g.*, Nuvem, *laptop*) com o objetivo de economizar energia poder de processamento do dispositivo móvel. O *offloading* de dados diz respeito a migrar o armazenamento dos dados do dispositivo móvel para um ambiente de maior capacidade de armazenamento, com a intenção de estender a capacidade de armazenamento do mesmo. De acordo com Fernando *et al.* (2013) e Kumar *et al.* (2013), existem várias questões que devem ser avaliadas antes de se fazer uso do *offloading*, a exemplo pode-se citar: (i)**Por que realizá-lo ?**; (ii)**Onde realizá-lo ?**; (iii)**Quando realizá-lo ?**; (iv)**Fazer *offloading* de que ?** e; (v)**Como executá-lo ?**. Os seguintes parágrafos irão tratar sobre cada uma dessas questões.

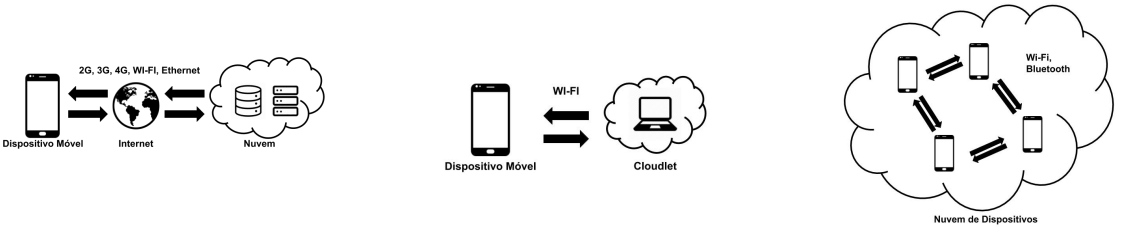
A utilização do *offloading* é defendida pelo fato de que apesar dos avanços dos dispositivos móveis em questões de *hardware* e *software*, eles ainda possuem recursos limitados, dito isso, Gomes *et al.* (2017) afirma que o uso desta técnica é proposta principalmente para melhorar o desempenho das aplicações móveis, economizar bateria do dispositivo e também para melhorar a colaboração entre usuários das aplicações móveis . Por ser uma técnica que tem por finalidade migrar dados, em relação a localização, ela poderá ser usada de um dispositivo móvel para qualquer dispositivo com capacidade de armazenamento e/ou processamento maior que a sua.

A execução do *offloading* pode acontecer em três locais de acordo com a propriedade citada anteriormente: uma Nuvem pública, um *cloudlet*[3] e outro dispositivo móvel, como representado na Figura 3. Com a *Nuvem pública*, os dispositivos móveis podem usar redes celulares (*e.g.*, 2G, 3G e 4G) e redes *Wi-Fi* para realizar o *offloading*. Como ilustrado na Figura 3 (b), Satyanarayanan *et al.* (2009) propôs o uso de *cloudlets* e tem como principal objetivo aproximar os serviços de *offloading* dos dispositivos móveis, ao invés de utilizar uma Nuvem pública. A vantagem encontrada nessa abordagem é que as redes locais, geralmente, possuem velocidades maiores e latência menores em relação aos servidores hospedados em infraestruturas mais remotas, e comumente acessados por redes celulares (COSTA *et al.*, 2015). O *offloading* também pode ser realizado para *outro dispositivo móvel*, nessa abordagem os dispositivos móveis

[3] Máquinas disponíveis na mesma rede local dos usuários das aplicações móveis (*e.g.*, *desktops, laptops*)

são normalmente conectados usando a rede *peer-to-peer*[4], para resolver uma tarefa em comum a todos com os outros dispositivos (FERNANDO *et al.*, 2013), representado na Figura 3 (c).

Figura 3 – Locais de execução do *offloading*



(a) Nuvem pública (b) Cloudlet (c) Dispositivos Móveis

Fonte – Próprio autor

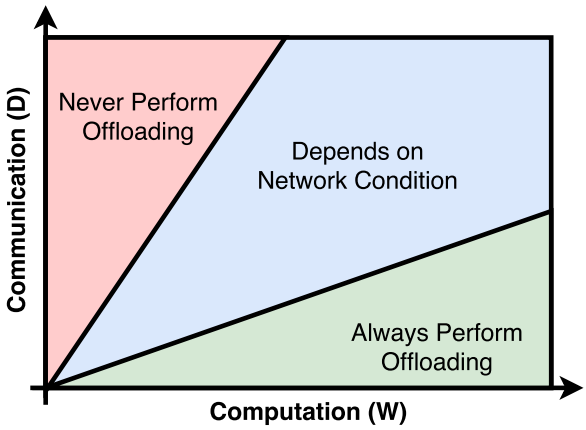
Para saber quando realizar ou não o *offloading*, é preciso considerar alguns critérios a fim de verificar se esta técnica de fato irá melhorar o desempenho do dispositivo móvel. Não existe um consenso sobre os critérios a serem analisados, pois a melhoria no desempenho desejada depende do contexto em que a aplicação está inserida, então um critério como a condição da rede pode ser relevante para uma aplicação que necessita de uma resposta rápida e nem tanto para os demais tipos de aplicações.

Em Kumar *et al.* (2013) é apresentado um modelo analítico para verificar quando o *offloading* melhora o desempenho dos dispositivos móveis. No modelo descrito, são comparados o tempo de processamento de uma tarefa executada no dispositivo móvel e o tempo para enviar, realizar a tarefa em um ambiente remoto e retornar o resultado para o dispositivo. Com o estudo foi obtido um gráfico apresentado na Figura 4, ao analisá-lo é possível notar que o autor concluiu alguns cenários para a utilização do *offloading*, são eles: (i) Enquanto a tarefa a ser realizada demandar muito poder computacional e o tempo de comunicação entre o dispositivo móvel e a Nuvem for satisfatório, sempre será indicado a realização do *offloading*; (ii) Enquanto a tarefa a ser realizada não demandar muito poder computacional e o tempo de comunicação entre o dispositivo móvel e a Nuvem for elevado, é recomendado sempre que o processamento seja realizado localmente, no dispositivo móvel; e por fim (iii) Enquanto o poder computacional necessário para realizar a tarefa e o tempo de comunicação entre o dispositivo móvel e a nuvem estiverem em valores intermediários, a realização (ou não) do *offloading* irá depender de outras

[4] É uma arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central

condições, como a situação da bateria ou o modo de consumo de energia.

Figura 4 – Modelo analítico de Kumar *et al.* (2013)



Fonte – Kumar *et al.* (2013)

Para Kharbanda *et al.* (2012), a decisão se deve ou não realizar o *offloading* vem em torno da economia de energia proporcionada no dispositivo móvel. Já em Rego (2016), é proposta uma abordagem que utiliza dados históricos para criar árvores de decisão e auxiliar na tomada de decisão, reduzindo o consumo energético causado pelo monitoramento periódico, mas com todas as operações de Computação relacionadas à criação da árvore executadas em servidores remotos, enquanto os dispositivos móveis só precisam analisá-la. A abordagem de Rego (2016) é uma extensão do modelo de Kumar *et al.* (2013), considerando o tempo de espera no servidor e o tempo para encriptar/decriptar os dados.

No que diz respeito ao o que deve ser migrado no *offloading*, Gomes *et al.* (2016) afirma que para a utilização desta estratégia em uma aplicação se faz necessário que parte ou toda a aplicação esteja disponível no servidor para que a requisição do cliente possa ser atendida. No mesmo trabalho, o autor ainda afirma que as soluções de *offloading* diferem em relação a que partes da aplicação devem ser migradas para execução no ambiente remoto, sendo que as partes mais migradas são: métodos, componentes/módulos, *threads* e aplicação completa.

Sobre a execução do *offloading*, ela pode ocorrer de diversas formas e por este motivo, em Rego (2016) é afirmado que não existe uma resposta única e simples para a questão de executar o *offloading*, no mesmo trabalho o autor apresenta uma taxonomia para auxiliar na classificação dessas soluções. Nessa taxonomia, essa questão tem as seguintes categorias:

- **Anotação do Método:** Quando a solução de *offloading* suporta a granularidade de método;

- **Decisão de *offloading*:** Está relacionada com a forma que é feita a decisão de realizar ou não o *offloading*;
- **Localização do Módulo de Decisão:** Está relacionada com a localização do módulo responsável pela decisão de realizar o *offloading*;
- **Abordagem do Módulo de Decisão:** Está relacionada com as abordagens e técnicas utilizadas pelo módulo de decisão;
- **Métricas Utilizadas para Decisão:** Diz respeito às métricas avaliadas no módulo de decisão do *offloading*;
- **Plataforma/Linguagem de Programação Suportada:** Diz respeito às plataformas móveis e linguagens de programação suportadas pela solução; e
- **Mecanismo de Descoberta:** Essa categoria leva em conta se a solução usa algum tipo de mecanismo de descoberta do ambiente remoto (*e.g.*, *cloudlet*).

A seguir é apresentado o *framework* CAOS (GOMES *et al.*, 2017), que é uma infraestrutura de software que realiza a operação de *offloading* de processamento e dados contextuais na plataforma Android, desenvolvida no laboratório *Grupo de Redes de Computadores Engenharia de Software e Sistemas* (GREat). O CAOS será utilizado para implantação do módulo de segurança e posterior análise do impacto desta na execução do *offloading*, sendo a proposta deste trabalho.

2.4 CAOS: *Context Acquisition and Offloading System*

Context Acquisition and Offloading System (CAOS) é um *framework* que surgiu devido a evolução da complexidade das aplicações móveis, pois segundo Gomes *et al.* (2017) atualmente é cada vez mais necessário a utilização de dados contextuais do usuário (*e.g.*, localização, temperatura e umidade) para melhorar sua experiência e usabilidade. Esses dados advém principalmente de sensores presentes nos dispositivos móveis (*e.g.*, *Global Positioning System* (GPS) e acelerômetro). Os dispositivos móveis por sua essência de mobilidade trazem consigo algumas restrições (abordadas na Seção 2.2). Devido a estas limitações ao aumento da quantidade de dados detectados e a complexidade dos procedimentos de inferência de contexto, os dispositivos móveis não possuem mais recursos suficientes para realizar algumas computações. (GOMES *et al.*, 2017)

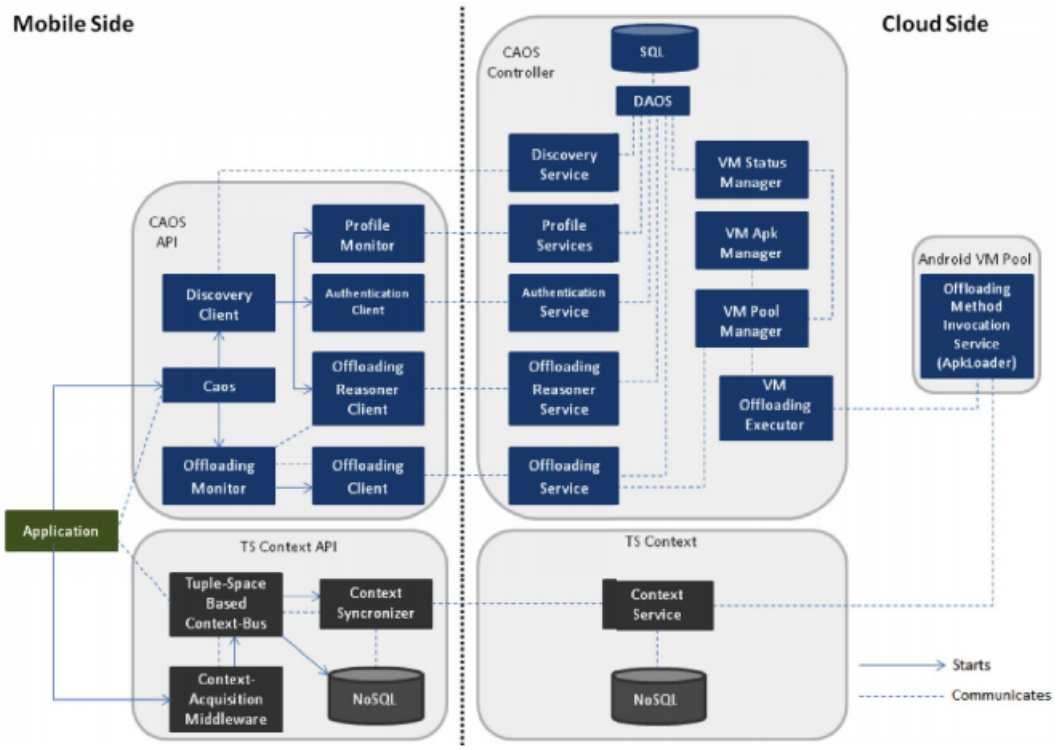
Buscando mitigar este efeito, o CAOS é um *framework* para auxiliar no desenvolvi-

mento de aplicações sensíveis ao contexto para a plataforma *Android*, trazendo a possibilidade da migração dos dados de processamento e/ou contextuais para a Nuvem por meio do *offloading* (abordado na Seção 2.3.1), podendo propiciar ganhos relacionados a economia de recursos (*e.g.*, energia, armazenamento). A seguir, será descrito a arquitetura e os componentes do *framework* supracitado.

2.4.1 Arquitetura

Em sua proposta, Gomes *et al.* (2017) adota uma arquitetura que se baseia em um modelo arquitetural Cliente/Servidor tradicional, onde os dispositivos móveis são os Clientes para os serviços executados na parte superior de infraestruturas de Nuvem ou parte Servidora (e.g., *cloudlet*). Sua arquitetura também é dividida basicamente em dois lados, o lado móvel e o lado da Nuvem. Cada uma delas apresenta um grupo de componentes necessários para seu funcionamento. A Figura 5 mostra a arquitetura do CAOS e mais adiante será discutido um pouco sobre seus dois grupos e seus componentes.

Figura 5 – Arquitetura do CAOS de Gomes *et al.* (2017)



Fonte – Gomes *et al.* (2017)

O lado móvel é formado por 10 (dez) componentes, sendo eles: (i) **CAOS**, respon-

sável por sincronizar a inicialização dos outros componentes do lado móvel; (ii) ***Offloading Monitor*** responsável por monitorar a execução do aplicativo e interceptar o fluxo de execução sempre que um método anotado é chamado, pois no CAOS os métodos dos aplicativos Java podem ser marcados (anotados) com a anotação *@Offloadable* que aponta que o método poderá ser executado no ambiente remoto; (iii) ***Offloading Client***, responsável pelo processo de transferência do método e seus parâmetros para a Nuvem; (iv) ***Discovery Client*** usa um mecanismo baseado em UDP [5]/*Multicast* [6] para descobrir um *cloudlet* em execução na rede local; (v) ***Authentication Client***, envia os dados do dispositivo móvel para o lado do servidor com o intuito de manter a lista de dispositivos conectados a uma infraestrutura específica do CAOS; (vi) ***Profile Monitor***, responsável pelo monitoramento do ambiente do dispositivo móvel e enviar as informações do mesmo periodicamente para o lado da Nuvem; (vii) ***Offloading Reasoner Client*** auxilia na decisão de *offloading* usando uma árvore de decisão que é enviada pelo lado da Nuvem; (viii) ***Context-Acquisition Middleware*** é um componente responsável pela aquisição contextual no CAOS; (ix) ***Tuple Space Based Context-Bus*** é responsável pelo armazenamento e gerenciamento das informações contextuais em formato de tupla e fornece essas informações à aplicação; e (x) ***Context Synchronizer*** que realiza a migração de dados contextuais do dispositivo móvel para Nuvem.

Já o lado da Nuvem é formado por 11 (onze) componentes, sendo eles: (i) ***Discovery Service***, responsável por fornecer aos Clientes os endereços do *cloudlet* para que os mesmos possam acessar os serviços do CAOS; (ii) ***Profile Services*** são um conjunto de serviços responsáveis por receber os dados do dispositivo relacionados à qualidade da conexão e o tempo de execução local dos métodos migrados com a intenção de manter um histórico do tempo de execução dos mesmos e usar esse histórico para decidir se o método deve ou não ser enviado para o ambiente remoto; (iii) ***Authentication Service***, responsável por armazenar quais são os dispositivos móveis conectados ao CAOS, bem como controlar quais dispositivos atualmente podem ter acesso aos serviços em Nuvem do CAOS; (iv) ***Offloading Reasoner Service*** realiza o processamento dos dados do perfil do usuário e gera uma estrutura de dados de decisão para decidir sobre a execução do processo de *offloading*; (v) ***Offloading Service*** recebe as solicitações de *offloading* dos clientes e as redireciona para o (vi) ***VM Pool Manager***, que é responsável por

[5] Protocolo da camada de transporte que permite que a aplicação envie dados, porém sem nenhum tipo de garantia que os mesmos cheguem corretamente.

[6] Estratégia utilizada para transmitir uma informação para múltiplos destinatários ao mesmo tempo presentes na mesma rede.

gerenciar um conjunto de máquinas virtuais Android, onde de fato irá correr a execução das requisições; (vii) ***VM Apk Manager*** aloca todos os arquivos APK para as máquinas virtuais disponíveis; (viii) ***VM Status Manager*** que é responsável por fazer o monitoramento e manter informações sobre as máquinas virtuais; (ix) ***VM Offloading Executor***, responsável por executar o *offloading* em uma máquina virtual; (x) ***Context Service*** trabalha como um repositório global que armazena todos os dados de informações de contexto enviados por todos que utilizam os serviços do CAOS no *cloudlet/cloud*; e (xi) ***Offloading Method Invocation Service*** que é responsável por executar o método requisitado para o *offloading*.

Como dito anteriormente, o CAOS é utilizado para apoiar o desenvolvimento de aplicações móveis e sensíveis ao contexto da plataforma Android. Durante a migração dos dados contextuais ou do envio dos dados para processamento no ambiente remoto, não existe qualquer componente responsável por garantir a confidencialidade e a segurança dos dados transmitidos. Assim, é necessário o desenvolvimento de um novo módulo que realize essa tarefa. A próxima Seção apresenta conceitos fundamentais para concepção desse módulo, bem como algoritmos criptográficos que serão comparados para analisar o impacto desses na técnica de *offloading*.

2.5 Segurança da Informação

A evolução da Computação Móvel trouxe uma nova gama de funcionalidades para os dispositivos móveis (*e.g.*, *smartphones, tablets*) e juntamente com elas, vieram algumas necessidades por parte dos usuários (*e.g.* armazenamento, processamento). Como dito anteriormente, para mitigar esses problemas surgiu a MCC (abordada na Seção 2.3), que faz uso da Computação em Nuvem para expandir os recursos dos dispositivos móveis. Mas a mesma traz consigo alguns problemas principalmente no tocante a segurança das informações, pois os dados percorrem a rede até chegarem na Nuvem muitas vezes desprotegidos, ficando sujeitos a ataques como *Man in the Middle*, que consiste no roubo de informações de usuários por meio do sequestro de uma conexão TCP (ANDRADE *et al.*, 2008).

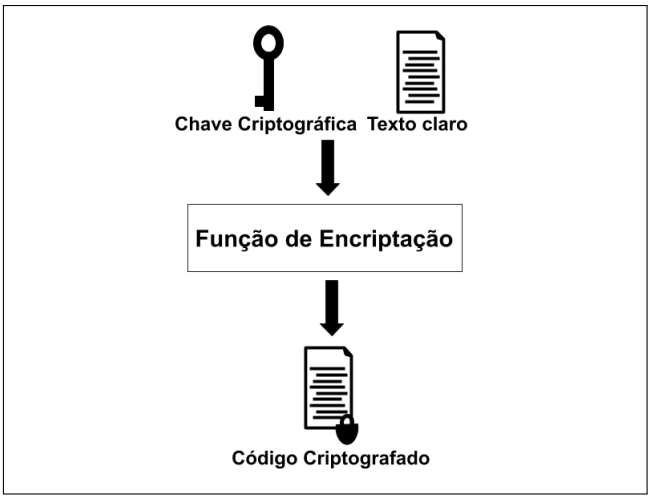
De acordo com Vijayarani e Tamilarasi (2011), o avanço da tecnologia permitiu que as organizações (*e.g.*, agências de recenseamento, hospitais) coletassem e armazenassem uma grande quantidade de dados pessoais sensíveis (*e.g.*, renda, registros médicos), e também que frequentemente liberem os mesmos para análise em mineração de dados, e isso cria um risco para a privacidade do indivíduo. Assim, aplicações de uso cotidiano podem gerar dados sensíveis do

usuário, que devem manter um nível de segurança e privacidade. Dessa forma, faz-se necessário a utilização de técnicas de segurança que garantam que esses dados trafeguem na rede e sejam armazenados na Nuvem de forma segura.

Segundo Hussein *et al*. (2016), as propriedades de segurança que devem ser garantidas pelos provedores da Nuvem são: Disponibilidade, Integridade e Confidencialidade. A **Disponibilidade** é definida como a garantia por parte dos Provedores de Serviços de Armazenamento, do inglês *Storage Service Provider* (SSP), que os dados e/ou processamento continuarão disponíveis em um nível de desempenho exigido em situações que variam de normais a desastrosas, a **Integridade** é um processo de proteção de dados contra exclusão, modificação (em caso de armazenamento ou transmissão) ou fabricação não autorizada e a **Confidencialidade** trata-se de assegurar que apenas as partes ou sistemas autorizados terão acesso aos dados protegidos.

Em Sujithra *et al.* (2015), é dito que os algoritmos de criptografia são as técnica mais usadas para proteger os dados no ambiente de Nuvem. Pode-se definir criptografia como uma técnica utilizada para garantir segurança na comunicação e a mesma funciona usando uma função de encriptação que recebe uma chave e uma mensagem, transformando a mensagem em um código criptografado antes de enviá-la, a Figura 6 ilustra o funcionamento descrito. Se a mensagem encriptada for interceptada por algum atacante, não poderá ser compreendida pelo mesmo. Para ter acesso ao conteúdo original da mensagem é necessário que o receptor da mensagem tenha acesso a chave de criptografia e faça o processo de decriptação com ela.

Figura 6 – Funcionamento da Criptografia



Fonte – Próprio autor

De acordo com Jhuria *et al.* (2013), a criptografia consiste em três grupos de algo-

ritmos: algoritmos de chave simétrica, algoritmos de chave assimétrica e *hashing*. A seguir é descrito um pouco sobre cada um deles.

2.5.1 Criptografia de Chave Simétrica

Esse método de criptografia utiliza a mesma chave para os processos de encriptação e decriptação, dessa forma a chave criptográfica deve ser conhecida pelo emissor e pelo receptor da mensagem. Como esta chave é o principal elemento deste método, ela deve ser transmitida de forma segura, este processo é chamado de "Distribuição de Chaves"e é exatamente nesta operação que se encontra a maior dificuldade, pois a gerência da distribuição de chaves para muitos indivíduos é um processo custoso, dado que o número de chaves cresce rapidamente (BELLARE *et al.*, 1997).

De acordo com AMARO (2008), os algoritmos de chave simétrica são classificados de acordo com o número de *bits* utilizados no processo de encriptação. Os algoritmos que utilizam tamanhos de *bits* variados são chamados de *Stream*, enquanto os que trabalham com um número fixo de *bits* são chamados de Bloco. Caso a mensagem tenha tamanho inferior ao tamanho definido, *bits* sem significados são adicionados somente para completar o bloco.

Algumas vantagens deste método em relação aos outros citados, ainda descritas no trabalho de AMARO (2008) são que: o método não causa grande impacto em relação a velocidade das operações executadas, pois os algoritmos de encriptação são executados rapidamente, permitindo cifrar uma grande quantidade de dados em pouco tempo; As chaves utilizadas são relativamente pequenas e simples, o que permite gerar cifradores extremamente robustos; e atinge seus objetivos de confidencialidade e privacidade, mantendo as informações transmitidas em segredo.

Assim como vantagens, AMARO (2008) também apresenta as desvantagens associadas a utilização desse método. A chave secreta deve ser compartilhada, podendo complicar a gerência de chaves; Não oferece suporte a ações para a autenticação do remetente, uma vez que qualquer pessoa poderá enviar uma mensagem criptografada com qualquer chave que esteja em seu domínio; e não permite o não-repúdio do remetente, onde o não-repúdio se refere a garantia que o autor não negue ter criado e assinado o documento e/ou mensagem.

Além das vantagens e desvantagens, AMARO (2008) apresenta os principais algo-

ritmos desse tipo de criptografia: o *Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), Advanced Encryption Standard (AES), RC4, RC6*. Além dos algoritmos citados pelo autor, existem dois outros algoritmos que serão de grande importância para este trabalho, sendo eles o *Blowfish* e o *ChaCha20*. A seguir será descrito cada um deles, onde os 5 primeiros baseiam-se na definição de AMARO (2008):

- **DES:** É um algoritmo simétrico com blocos de 64 *bits* dos quais 8 *bits* são dedicados ao teste de integridade, assim sendo, apenas 56 *bits* são realmente usados para a encriptar a mensagem. O algoritmo funciona efetuando operações de combinação, substituição e permutação entre o texto a ser codificado e a chave, de forma que essas operações possam ser desfeitas no processo de decodificação. Mais detalhes sobre o processo de encriptação está descrito no trabalho de Adhie *et al.* (2018);
- **3DES:** Uma variação do DES que utiliza 3 chaves de 64 *bits*, com o tamanho máximo de chave 168 *bits* (fora os *bits* de paridade). O processo de encriptação ocorre com 3 ciframentos em sequência, onde a mensagem é encriptada com a primeira chave, decriptada com a segunda e finalmente encriptado novamente com a terceira, estas operações fazem o método ser mais lento que o DES, mas oferece maior segurança. Mais detalhes sobre o algoritmo está descrito no trabalho de Adhie *et al.* (2018);
- **AES:** É o padrão atual para ciframento recomendado pelo *National Institute of Standards and Technology* (NIST). Trabalha com chaves de 128, 192 e 256 *bits*, e o número de rodadas r que serão utilizadas no processo depende do tamanho da chave. O algoritmo possui uma chave principal, utilizada para a criptografia antes da primeira rodada, e a partir da mesma são geradas Nr + 1 chaves, onde cada uma delas será utilizada em uma rodada diferente. Em cada etapa são utilizadas operações de substituição e transposição;
- **RC4:** Também conhecido como ARC4, algoritmo simétrico de *Stream* desenvolvido por Rivest e conhecido por ser bem simples e rápido em *software*. Seu funcionamento se dá com operações simples de combinações de *bits* por meio de operações XOR, de forma a combinar os *bits* da chave com os bits do texto sem formatação;
- **RC6:** A última versão de uma série de cifradores (RC2, RC3, RC4, RC5) desenvolvidos por Rivest. Mais seguro contra criptoanálise[7] e mais veloz que o RC5, pois o mesmo utiliza várias iterações, onde em cada uma dessas iterações é utilizada uma sub-chave. Possui o número de iterações e o número de *bits* utilizados nas chaves variáveis e baseia-se

[7] Arte de tentar descobrir o texto cifrado e/ou a lógica utilizada em sua encriptação.

em operações de rotação e multiplicação.

- **Blowfish:** É um algoritmo de codificação de chave simétrica que utiliza blocos de tamanho de 64 *bits* e comprimento de chave variável de 32 a 448 *bits*. Projetado no ano de 1993 por Bruce Schneider com a intenção de ser um algoritmo mais rápido do que os já existentes na época. O funcionamento do algoritmo se dá em duas partes: (i) Expansão das chaves, que converte a chave de no máximo 448 *bits* em várias matrizes de subchaves com o total de 4168 *bytes*; e (ii) Criptografia dos dados que ocorre por meio de uma rede de Feistel[8] de 16 voltas em que cada uma das rodadas consiste em uma permutação dependente da chave e dos dados. Todas as operações que ocorrem no processo são XOR's, adições em palavras de 32 de *bits* e quatro operações adicionais de pesquisas de dados na matriz (SCHNEIER, 1993);
- **ChaCha20:** É um algoritmo de codificação projetado por Daniel J. Bernstein que utiliza uma chave secreta de 256 *bits* de comprimento, uma função de bloco (nomeada ChaCha20) e que requer um *nonce* [9] de 8 ou 12 *bytes*. Seu funcionamento ocorre executando a função ChaCha20 sucessivamente com a mesma chave e *nonce*, aumentando para cada chamada o contador de blocos de parâmetros, logo após ele serializa o estado resultante escrevendo os números na ordem *little-endian* [10], criando um bloco de *keystream*. Concatenar os blocos de *keystream* dos blocos sucessivos forma um fluxo de chaves. A função ChaCha20 executa um XOR do *keystream* com o texto simples. O processo de descriptografia é feita da mesma maneira. (NIR; LANGLEY, 2018)

2.5.2 Criptografia de Chave Assimétrica

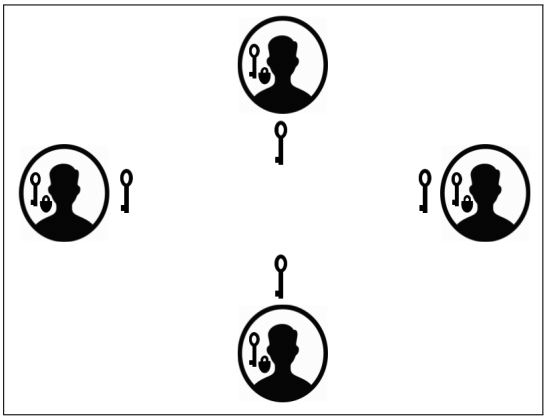
Este método de criptografia é baseado na utilização de pares de chaves diferentes, sendo uma delas privada e a outra pública. Nesse processo, as chaves são relacionadas por meio de um processo matemático e a informação é codificada por meio de funções unidirecionais (STINSON, 2005). A chave pública fica disponível e é visível para qualquer pessoa que queira se comunicar com outra, enquanto a chave privada é visível apenas para seu respectivo dono, pois é com ela que o mesmo poderá descodificar a mensagem que foi criptografada usando sua chave pública, a Figura 7 representa o esquema das chaves descritas anteriormente.

[8] Uma estrutura simétrica usada na construção de cifras de bloco

[9] Um número (ou letra) arbitrário(a) que só poderá ser utilizado uma vez, motivo no qual deu origem ao seu nome (N = Number (Número) e Once = Uma vez, do inglês)

[10] Forma de representação de números binários que considera o último bit da direita para a esquerda, como o mais significativo.

Figura 7 – Esquema de chaves da Criptografia Assimétrica



Fonte – Próprio autor

Oliveira (2012) afirma que a grande vantagem desse método se encontra em permitir que qualquer um envie uma mensagem secreta apenas utilizando a chave pública de quem a mensagem se destina, pois o fato das chaves públicas estarem sempre disponíveis à todos faz com que não haja a necessidade de enviar chaves como ocorre no modelo simétrico. A segurança na troca de mensagens é garantida enquanto as chaves privadas estiverem em segurança, caso contrário, quem estiver de posse das mesmas poderá ter acesso ao conteúdo secreto das mensagens trocadas.

Entretanto, no mesmo trabalho o autor aponta como desvantagem deste método a complexidade empregada no desenvolvimento dos algoritmos, pois eles devem ser capazes de reconhecer a dupla de chaves existentes bem como relacioná-las no momento oportuno, o que demanda de um grande poder de processamento computacional.

Dentre os algoritmos desenvolvidos, Oliveira (2012) cita como principais: **RSA**, **Diffie-Hellman** e **Criptografia de Curva Elíptica**, do inglês ***Elliptic Curve Cryptography* (ECC)**. A seguir é relatado um pouco sobre cada um deles segundo o mesmo:

- **RSA:** O RSA possui este nome devido a seus inventores: Ron Rivest, Adi Shamir e Len Adleman, que o criaram em 1977 no MIT. Atualmente, é o algoritmo de chave pública mais usado, bem como uma das mais poderosas formas de criptografia de chave pública conhecidas até o momento. Este método utiliza números primos e sua premissa consiste na facilidade de multiplicar dois números primos para obter um terceiro número e na dificuldade de recuperar os dois primos a partir daquele terceiro número. Gerar a chave pública envolve multiplicar dois primos grandes e derivar a chave privada a partir da chave

pública envolve fatorar um grande número. Se o número for grande o suficiente e bem escolhido, então ninguém pode fazer isto em uma quantidade de tempo razoável. Logo, sua segurança baseia se na dificuldade de fatoração de números grandes;

- **Diffie-Hellman:** Baseado no problema do logaritmo discreto definido no trabalho de Dullius (2001), o Diffie-Hellman é o cripto sistema de chave pública mais antigo ainda em uso. Não permite nem ciframento nem assinatura digital, pois o sistema foi projetado para permitir que dois indivíduos entrem em um acordo ao compartilharem um segredo tal como uma chave, muito embora eles somente troquem mensagens em público;
- **Criptografia de Curva Elíptica (ECC):** Consiste em modificações em outros sistemas, passando a trabalhar no domínio das curvas elípticas definido no trabalho de Portnoi (2005), ao invés de trabalhar no domínio dos corpos finitos[11]. Possuem o potencial de sistemas criptográficos de chave pública mais seguros, com chaves de menor tamanho. Muitos algoritmos de chave pública (*e.g.*, Diffie-Hellman, o ElGamal e o Schnorr) podem ser implementados em curvas elípticas sobre corpos finitos, resolvendo assim, um dos maiores problemas dos algoritmos de chave pública, o grande tamanho de suas chaves.

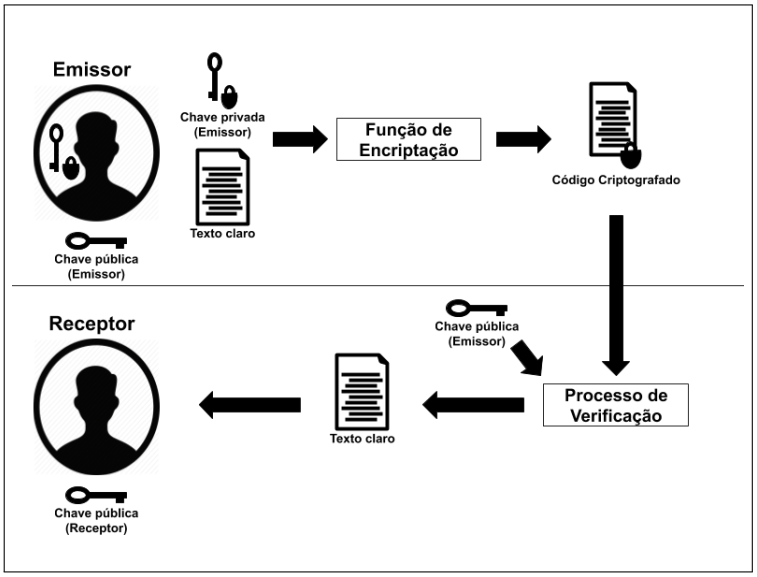
2.5.3 Hashing

A assinatura digital consiste no processo de autenticação e segurança que se baseia em uma inversão do sistema de criptografia assimétrica, onde o emissor cifra a mensagem com sua chave privada e a envia para receptor. Cada um que receber a mensagem deverá verificar a validade da assinatura digital, utilizando a chave pública do emissor e reconhecendo de fato, se a mensagem foi adulterada ou não, todo este processo está representado na Figura 8. A utilização de tal estratégia se torna inviável principalmente para grandes mensagens, pois o processo poderia durar minutos ou mesmos horas para ser concluído (BELLARE *et al.*, 1996). Função *Hashing* (ou resumo criptográfico), segundo Preneel (1993), é um mecanismo criado para tornar adequado a utilização da assinatura digital.

Essas funções são caracterizadas por gerarem uma sequência única e de quantidade fixa de *bits* para a mensagem transmitida, independente do tamanho do conteúdo original da mesma, e por serem funções unidirecionais, ou seja, a partir da cadeia de *bits* gerada não é possível retornar ao seu valor original. A seguir, são apresentadas as principais funções

[11] Um conjunto no qual é definido as operações de soma, subtração, multiplicação e divisão, bem como as propriedades usuais dessas operações. Esse mesmo conjunto possui uma quantidade finita de elementos.

Figura 8 – Processo de comunicação usando Assinatura Digital



Fonte – Próprio autor

segundo Oliveira (2012):

- **SHA-1:** Uma função de espalhamento unidirecional inventada pela *National Security Agency* (NSA), gera um valor *hash* de 160 *bits*, a partir de um tamanho arbitrário de mensagem;
- **SHA-2:** Desenhado pela NSA é uma família de duas funções *hash* similares, com diferentes tamanhos de bloco, conhecido como SHA-256 e SHA-512, onde as mesmas diferem na quantidade de *bits* utilizados, 256 e 512 *bits*, respectivamente;
- **MD5:** Este algoritmo produz um valor *hash* de 128 *bits*, para uma mensagem de entrada de tamanho arbitrário. Foi inicialmente proposto em 1991, após alguns ataques de criptoanálise terem sidos descobertos contra a função *hashing* prévia de Rivest: a MD4. O algoritmo foi projetado para ser rápido, simples e seguro;
- **MD2 e MD4:** O MD4 é o precursor do MD5 inventado por Ron Rivest e substituído após terem sido descobertas fraquezas no mesmo, não sendo mais usado atualmente. O MD2 é uma função de espalhamento unidirecional simplificada que produz um *hash* de 128 *bits*. A segurança do MD2 é dependente de uma permutação aleatória de *bytes*.

2.6 Considerações Finais

Este capítulo apresentou os conceitos básicos dos paradigmas: *Computação em Nuvem*, *Computação Móvel*, *Mobile Cloud Computing*. Também foi apresentado uma visão geral

do tema de *Segurança da Informação* e sobre o *framework* CAOS. Com relação à *Computação em Nuvem* foram apresentadas definições e conceitos básicos, características essenciais e modelos de serviços de Nuvem, assim como os modelos de implantação de Nuvem. Sobre *Computação Móvel*, foram apresentados conceitos relacionados ao tema, bem como limitações e desafios desse paradigma. Com relação ao paradigma de *Mobile Cloud Computing* foram apresentado conceitos da área, a técnica de *offloading*, além de algumas das principais questões de pesquisa relacionadas à essa técnica. Por fim, uma visão geral da *Segurança da Informação*, como conceitos, propriedades, e sobre o tema de *Criptografia*, abordando os tipos de criptografia e os algoritmos mais utilizados.

3 TRABALHOS RELACIONADOS

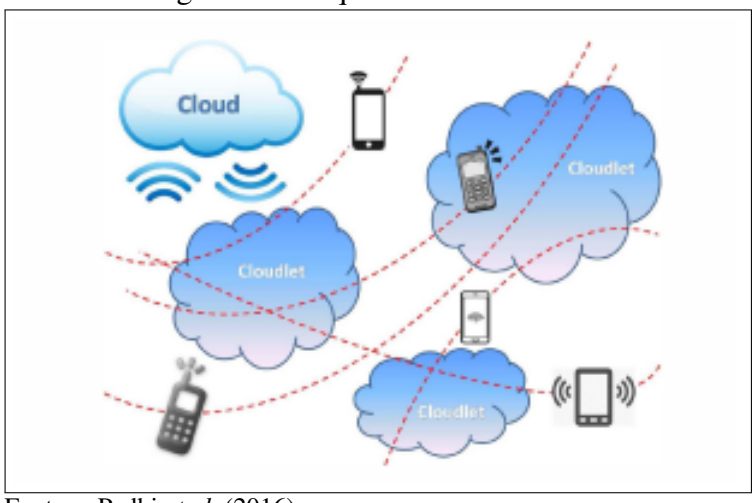
Com o objetivo de apresentar trabalhos relacionados à presente proposta, os trabalhos abordados neste capítulo, retirados dos repositórios IEEE e ACM, envolvem infraestruturas de software que proveem segurança no processo de *offloading* computacional do dispositivo móvel para um ambiente remoto.

3.1 *SecOMN: Improved Security Approach for Opportunistic Mobile Networks Using Cyber Foraging*

Padhi *et al.* (2016) apresenta um trabalho intitulado Rede Móvel Oportunista Segura, do inglês SecOMN, que tem por objetivo aumentar o nível de segurança e reduzir o gasto energético dos nós das Redes Móveis Oportunistas, do inglês *Opportunistic Mobile Networks* (OMN) definida por Chaintreau *et al.* (2007) como um gráfico com um conjunto estático de nós e um conjunto de arestas que podem mudar com o tempo. Segundo os autores, a maioria das redes OMN dependem dos nós participantes para fornecer segurança, o que torna o modelo de segurança heterogêneo e computacionalmente mais intensivo para os mesmos. O SecOMN é executado em ambiente de *cloudlets* e o aumento da segurança é obtido pela introdução da criptografia de chave simétrica e assimétrica, garantindo também a integridade e autenticidade dos pacotes transmitidos entre os dispositivos e o *cloudlet*.

Os autores introduziram os *cloudlets* nas Redes Móveis Oportunistas, onde os mesmos são implantados em locais fixos para que cada nó faça acesso a qualquer um deles. A Figura 9 ilustra a arquitetura do sistema proposto. O modelo utiliza uma estratégia de armazenamento de chaves *offline*, baseado em Nagar e Alshamma (2012), onde as chaves usadas são salvas em um banco de dados compartilhado por todos os *cloudlets*, eliminando a necessidade de enviar a chave junto da mensagem, enviando apenas o índice da chave utilizada no processo. Cada *cloudlet* é composto por um *gerenciador de chaves*, que é responsável por obter no banco de dados as chaves necessárias para o processo de encriptação ou decripção da mensagem, um *criptossistema*, que é visto como um núcleo do modelo responsável por lidar com as operações de criptografia e decriptografia e por fim, um *gerenciador de dados* que realiza a gerência dos dados trafegados.

Figura 9 – Arquitetura do SecOMN



Fonte – Padhi *et al.* (2016)

O criptossistema adotado utiliza uma abordagem híbrida, criando dois níveis de segurança e com a utilização de chaves de diversos tamanhos. O processo de criptografia é dividido em duas fases, na primeira o texto plano é criptografado utilizando o AES e na segunda, é utilizado o RSA para criptografar a mensagem e as chaves parciais para que os mesmos possam ser transmitidos juntos, eliminando a necessidade de um canal seguro de comunicação. As chaves parciais são usadas para garantir a autenticação das mensagens usando o *hash* criptográfico **SHA-256** (abordado na Seção 2.5.3).

Para realizar tais ações, o módulo principal de encriptação, requer outros 2 (dois) módulos adicionais, sendo eles o ***keyShuffler*** que é usado para trazer mais segurança, implementando o conceito de atomicidade na rede, usando chaves parciais para que a decriptação da mensagem completa ocorra apenas quando todos os seus pacotes estiverem disponíveis, pois apenas de posse de todos os pacotes é possível gerar uma Chave Mestra M, capaz de decriptar a mensagem completa e o ***PacketGenerator***, que é um módulo responsável por retornar pacotes de mensagens prontos para o armazenamento e transmissão.

Segundo os autores, com o SecOMN, usar *cloudlets* no modelo de segurança homogêneo é melhor do que os modelos existentes que dependem dos nós móveis participantes para fins de segurança, tornando o modelo de segurança heterogêneo. Padhi *et al.* (2016) afirma que o SecOMN oferece um modelo de segurança versátil e facilmente implementável com os protocolos de roteamento existentes no ambiente OMN.

Como deficiência desse trabalho, foi verificado que os autores não apresentaram

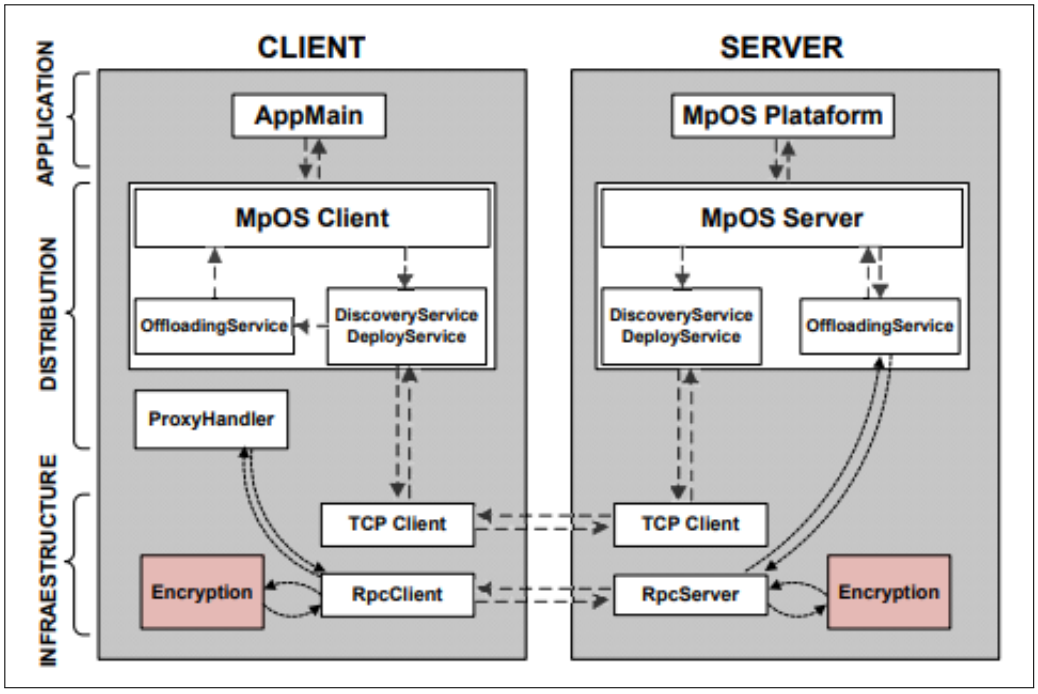
experimentos para comprovar a eficiência do sistema proposto. Por conta disto não é possível mensurar os resultados deste trabalho, tornando-o apenas um modelo teórico.

3.2 *Performance Evaluation of Cryptography on Middleware-Based Computational offloading*

Silva *et al.* (2017) propõem a utilização de formas de criptografias em um modelo de *offloading* (abordado na Seção 2.3.1) baseado em um *middleware* para garantir a integridade e autenticidade dos dados migrados para o *cloudlet*, sem prejudicar o desempenho das aplicações e o consumo de energia do dispositivo móvel.

Os autores utilizaram o *middleware* MpOS, que segue o modelo Cliente/Servidor de aplicações distribuídas e o modificaram para receber um módulo criptográfico, em que a aplicação que utiliza a solução pode optar por usar ou não a criptografia. A utilização do MpOS é defendida pois o mesmo lida com a complexidade de distribuição e facilita o desenvolvimento das aplicações que querem utilizar a técnica de *offloading*. A arquitetura e as interações entre as camadas de aplicação, distribuição e infraestrutura utilizadas por Silva *et al.* (2017) estão descritas na Figura 10.

Figura 10 – Arquitetura do *middleware* de Silva *et al.* (2017)

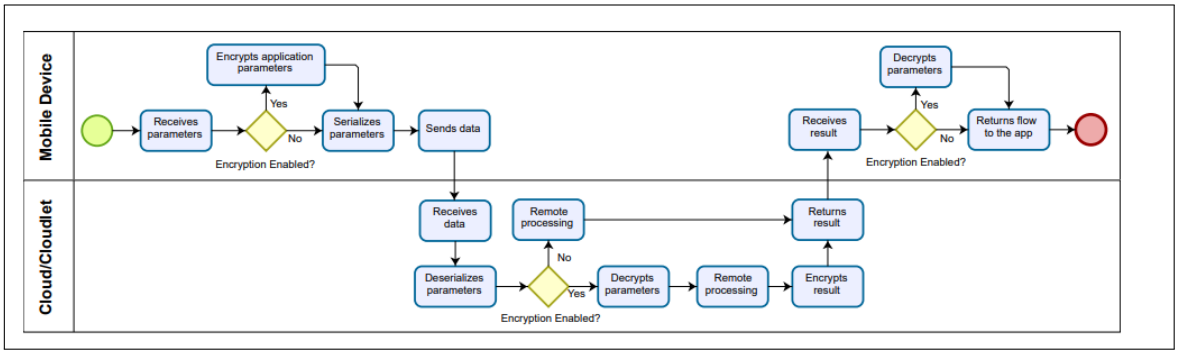


Fonte – Silva *et al.* (2017)

Para a criptografia foi utilizado um modelo híbrido, com um algoritmo de criptografia

simétrica (abordado na Seção 2.5.1) para codificar os dados (AES) e um algoritmo de criptografia assimétrica (abordado na Seção 2.5.2) para fazer o compartilhamento das chaves (RSA). Todo o fluxo de execução está representado na Figura 11 e o funcionamento da solução ocorre da seguinte maneira:

Figura 11 – Fluxo de execução da solução proposta por Silva *et al.* (2017)



Fonte – Silva *et al.* (2017)

1. O *middleware*, em tempo de execução, recebe os parâmetros da aplicação e em seguida, verifica se foi habilitado a criptografia ou não;
2. Caso tenha sido habilitado, o módulo de criptografia é acionado para criptografá-los e os repassa para a serialização e posteriormente ocorre o envio (i.e., requisição do cliente);
3. Caso contrário, é feita apenas a serialização dos parâmetros e logo em seguida eles são enviados;
4. Ao receber os dados, o *middleware* do servidor desserializa os parâmetros e também verifica se a criptografia foi ativada;
5. Caso tenha sido ativada, o módulo de criptografia é acionado novamente, mas dessa vez para descriptografar os parâmetros para que possam ser processados no ambiente de execução remota e criptografar a resposta do servidor para o cliente que solicitou;
6. Caso contrário, apenas o processamento remoto será executado e a resposta do servidor é enviada para o cliente;

Os experimentos de avaliação do *middleware* foram feitos nos aplicativos móveis *BenchImage* e *BenchFace*. O *BenchImage* é um aplicativo para o processamento de imagens que aplica efeitos em fotos de diferentes resoluções e que pode executar essas tarefas inteiramente no dispositivo móvel ou apenas parcialmente por meio do *offloading* (abordado na Seção 2.3.1), se o usuário optar pelo *offloading* a aplicação transferirá a foto para o um servidor remoto (*e.g.*, Nuvem, *cloudlet*) e a receberá após a aplicação dos efeitos. (SILVA *et al.*, 2017)

Já o *BenchFace* é um aplicativo para a detecção de faces que usa uma abordagem de aprendizado de máquina, treinando funções em cascata com imagens positivas (que possuem faces) e negativas (que não possuem faces). O aplicativo consiste em uma única imagem com 78 faces em ângulos distintos, onde o usuário poderá alterar a resolução da imagem e o algoritmo classificador da mesma. Ele pode ser executado localmente, na Nuvem ou dinamicamente (i.e., simultaneamente nos dois ambientes). No fim de sua execução, o aplicativo exibirá a quantidade de rostos detectados e o tempo decorrido para executar a tarefa. (SILVA *et al.*, 2017)

Com os resultados obtidos, os autores puderam concluir que a natureza da aplicação, a capacidade de processamento dos dispositivos, o volume de dados e o tamanho da chave criptográfica utilizada são fatores que impactam no *offloading* criptografado. No aplicativo *BenchImage*, usando imagens de diferentes tamanhos e operações com chaves de criptografia menores foi alcançado uma melhor relação entre o custo das operações e o benefício trazido por elas em relação ao tempo de uso e consumo de energia, sendo possível concluir também que neste caso o uso de chaves criptográficas maiores não se justifica, pois diminui consideravelmente o desempenho. O aplicativo *BenchFace* demonstrou a mesma tendência, mas foram encontrados resultados semelhantes na comparação entre a utilização do algoritmo AES com tamanho de chave de 128 *bits* e a utilização do AES + RSA, fazendo com que a segunda escolha seja melhor, pois a mesma fornece um nível maior de segurança.

Após a análise da estratégia proposta como solução, foi possível notar alguns problemas que se considerados, poderiam modificar os resultados obtidos pelos mesmos. Pode-se citar: (i) falta de análise real dos impactos trazidos pelo processo de encriptação na transferência dos dados, pois a avaliação só é feita no *offloading* como um todo, não analisando o tempo gasto apenas com as operações de criptografia e qual seu percentual de contribuição para o tempo total; (ii) e também a avaliação do consumo energético das aplicações foi realizado por meio de *software*, o que por muitas vezes pode trazer resultados imprecisos.

3.3 *Analysis of lightweight encryption scheme Fog-to-things communication*

Diro *et al.* (2018) trata da questão do crescimento da Internet das Coisas, do inglês *Internet of Things* (IoT), e o crescimento e a sofisticação dos ataques cibernéticos que são os maiores desafios trazidos pela conexão física do IoT. O trabalho analisa os desafios de segurança em termos de princípios de segurança cibernética e propõe um novo esquema de

criptografia para a comunicação *Fog*[1] em IoT.

No trabalho foi utilizado um mecanismo de criptografia usando o esquema de re-criptografia de *proxy* baseado em ECC (abordado na seção 2.5.2), consistindo em 5 procedimentos, sendo eles:

1. ***Key Generation*:** Produz parâmetros de curva elíptica pública, chaves públicas e secretas do nó coordenador. Os parâmetros de curva elíptica pública PK são enviados para os dispositivos de IoT e os demais nós da rede;
2. **Procedimento de Criptografia do Cliente:** Etapa onde é realizada a criptografia de uma mensagem m por um equipamento IoT usando sua chave privada;
3. **Procedimento de Criptografia *Fog*:** Etapa onde o nó da Névoa recebe uma mensagem m criptografada, os parâmetros da curva elíptica pública e a chave privada do usuário que se destina a mensagem, realiza uma criptografia com a mensagem m e os parâmetros e a retorna;
4. **Procedimento de Decriptação de *Fog*:** Etapa onde o nó da Névoa recebe uma mensagem m criptografada, os parâmetros da curva elíptica pública e a chave privada do usuário que se destina a mensagem, realiza um processo de decriptografia e retorna uma cifra intermediária di (m) que só pode ser descriptografada pelo cliente Ci o qual a mensagem se destina;
5. **Procedimento de Decriptação do Cliente:** Etapa onde o nó da Névoa recebe uma cifra intermediária mi e a chave privada do cliente e realiza a decripção da cifra a partir da chave e retorna a mensagem original.

No esquema de Diro *et al.* (2018), é necessário a eleição de um nó da *Fog* como autoridade ou coordenador confiável, que será responsável pela geração de chaves e parâmetros. Para medir a eficácia de sua estratégia, os autores realizaram alguns experimentos, sendo eles uma série de execuções de sua estratégia e uma estratégia utilizando a criptografia RSA e ao final são comparados os resultados obtidos por ambas estratégias.

Cada experimento foi executado 20 vezes para os parâmetros em determinados tamanhos de dados afim de medir os tempos de execução e a produtividade das funções de

[1] Plataforma virtualizada que fornece serviços de computação, armazenamento e comunicação entre os dispositivos finais e os centros de dados tradicionais de computação em Nuvem, localizada normalmente, mas não exclusivamente, na borda da rede (BONOMI *et al.*, 2014)

segurança. O tempo de execução e o rendimento foram medidos em três categorias de tamanhos de mensagens (32, 64 e 128 bytes) para uma curva elíptica usando três níveis de segurança diferentes (80, 128 e 256 *bits*) como mostrado na Figura 12.

Figura 12 – Parâmetros de curva dos experimentos.

	Security levels		
Parameters	80 bits	128 bits	256 bits
Length of q	512	1536	7680
Length of r	160	256	512

Fonte – Adaptado de Diro *et al.* (2018)

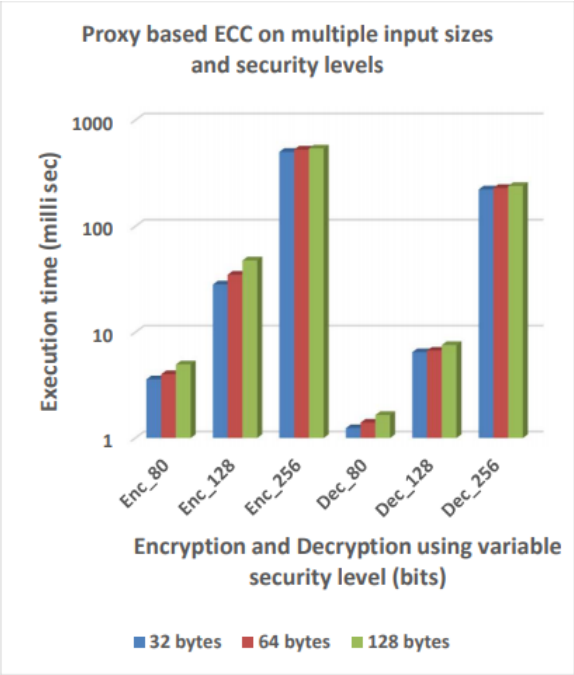
Os resultados obtidos com os testes realizados mostraram que: (i) embora os tempos de criptografia e decriptografia de sua estratégia aumentem com o aumento da segurança, eles se mantém quase os mesmos para vários tamanhos de dados (observe na Figura 13); (ii) a realização do processo de criptografia e decriptografia com vários tamanhos de mensagens se mostra mais rápido com sua estratégia do que utilizando a estratégia com o RSA (observe na Figura 14); (iii) e a decriptografia RSA tende a ser o processo mais lento em relação ao mesmo processo em ECC, sendo assim, ECC se mostra ser o mais rápido em cálculos, o que de acordo com os autores indica que o *offloading* de funções de segurança para nós de nevoeiro podem diminuir o tempo de processamento bem como a utilização de recursos dos dispositivos, o que traz melhorias para os dispositivos com recursos limitados, como IoT.

Logo após uma análise do trabalho, foi possível notar que pode haver uma fragilidade na mesma, pois os testes foram realizados em dados pequenos (32, 64 e 128 *bytes*), não evidenciando o comportamento assumido por sua estratégia em relação ao RSA em dados maiores (*e.g.*, uma foto de 5 *Megabytes*). Uma outra fragilidade encontrada é que os testes foram executados apenas 20 vezes sobre cada instância de teste, o que de acordo com Fischer (2010) não é o suficiente para que os dados apresentem uma distribuição normal (Distribuição Gaussiana).

3.4 Considerações Finais

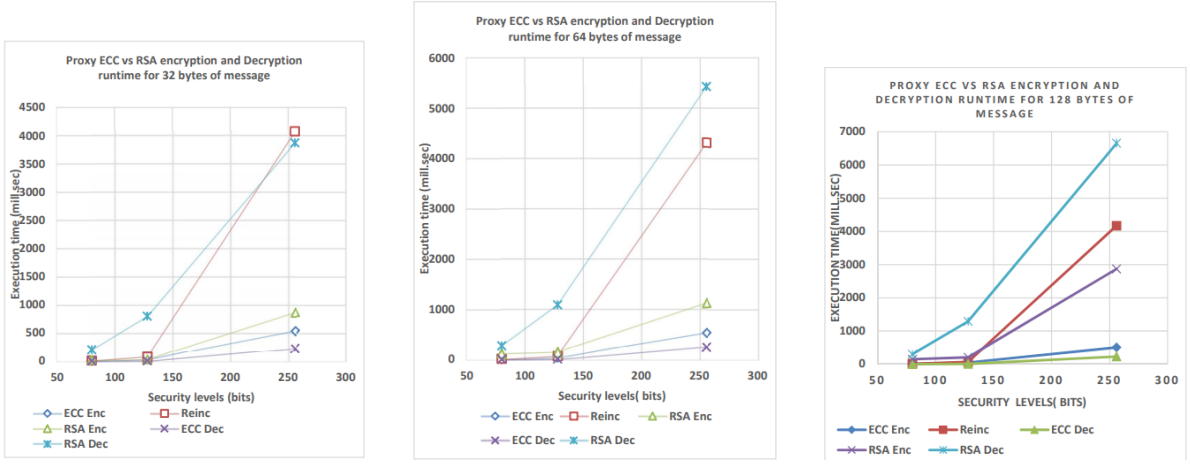
Esse capítulo apresentou os trabalhos relacionados, que envolvem infraestruturas de software que proveem segurança no processo de *offloading* computacional do dispositivo móvel para um ambiente remoto.

Figura 13 – Tempo de criptografia e decriptografia.



Fonte – Diro *et al.* (2018)

Figura 14 – Esquema de Diro *et al.* (2018) vs RSA: Tempo de execução.



(a) 32 bytes (b) 64 bytes (c) 128 bytes

Fonte – Diro *et al.* (2018)

Foi verificado que alguns dos trabalhos relacionados apresentam problemas como a falta de uma análise dos impactos causados no desempenho da aplicação por conta da adição da criptografia. Dessa forma, o próximo capítulo apresenta a solução proposta e sua arquitetura.

4 ESTUDO COMPARATIVO DO IMPACTO DA SEGURANÇA EM AMBIENTES DE *MOBILE CLOUD COMPUTING*

As seções anteriores detalharam conceitos necessários para a compreensão deste trabalho, alguns problemas encontrados atualmente no paradigma da Computação Móvel (*e.g.*, baixos níveis de armazenamento e processamento, bateria limitada) e discorreram sobre o paradigma da MCC, que é uma estratégia utilizada para mitigar esses problemas. Foram apontados também os problemas de segurança trazidos pela utilização da MCC. Além disso, foram expostos trabalhos relacionados às áreas da segurança da informação e sobre a utilização da criptografia no *offloading*. No entanto alguns deles apresentam problemas como a falta de uma análise dos impactos causados no desempenho da aplicação por conta da adição da criptografia.

Visando alcançar o objetivo do presente trabalho (apresentado na Seção 1.2.1), para medir o desempenho, será utilizada a métrica de tempo de execução total da tarefa, onde o tempo de execução consiste da diferença entre o instante em que a requisição foi feita pelo dispositivo e o instante em que a resposta da requisição é retornada, dada em milissegundos (ms).

A métrica descrita anteriormente é relevante e foi escolhida para o presente trabalho pois os dispositivos móveis possuem recursos limitados (abordado na Seção 2.2), assim faz-se necessário avaliar o impacto da segurança na técnica de *offloading*, para verificar o quanto os algoritmos podem influenciar no tempo de processamento das tarefas, pois o seu aumento representa um maior consumo de recursos. A seguir é apresentado a solução proposta visando alcançar os objetivos mencionados na Seção 1.2.1.

4.1 Metodologia

Durante o processo de pesquisa foram encontrados trabalhos relacionados a presente proposta, os quais são infraestruturas de *software* que proveem segurança no processo de *offloading* computacional de dispositivos móveis para um ambiente remoto. Desta forma, a presente pesquisa fará uso de alguns algoritmos apresentados em detalhes na Seção 2.5.1. A Tabela 1 lista e traz uma breve definição dos algoritmos utilizados para inserir segurança no *framework* CAOS (ver Seção 2.4).

O processo de escolha dos algoritmos foi baseado principalmente nas características de cada um, priorizando os que foram utilizados pelos autores dos trabalhos relacionados. Os

Tabela 1 – Algoritmos criptográficos escolhidos para o trabalho

Algoritmo	Descrição	Classificação
AES	Trabalha com chaves de 128, 192 e 256 bits, e o número de rodadas r que serão utilizadas no processo depende do tamanho da chave. O algoritmo possui uma chave principal, utilizada para a criptografia antes da primeira rodada, e a partir da mesma são geradas Nr + 1 chaves, onde cada uma delas será utilizada em uma rodada diferente.	Simétrico
ARC4	Algoritmo de criptografia notado por ser bem simples e rápido, trabalha gerando um fluxo pseudo-aleatório de bits e o combinando com o texto sem formatação bit a bit	Simétrico
Blowfish	É um algoritmo que possui blocos de tamanho de 64 bits e comprimento de chave variável de 32 a 448 bits. Funciona convertendo a chave em várias matrizes de subchaves e criptografa os dados por meio de uma rede de Feistel de 16 voltas.	Simétrico
DES	Utiliza blocos de 64 bits dos quais 8 bits são dedicados ao teste de integridade e 56 bits são realmente usados para encriptar a mensagem. O algoritmo funciona efetuando operações de combinação, substituição e permutação entre o texto a ser codificado e a chave, de forma que essas operações possam ser desfeitas no processo de decodificação.	Simétrico
3DES	Variação do DES que utiliza 3 chaves de 64 bits, utilizando 168 bits para criptografia e 24 para integridade. O processo de encripção ocorre com 3 ciframentos em sequência.	Simétrico

Fonte – Próprio Autor

algoritmos AES e *Blowfish* foram inseridos utilizando chaves de 128 *bits*, o ARC4 foi inserido utiliza um fluxo pseudo-aleatório de 480 *bits*, já o DES e 3DES utilizam seu tamanho de chave padrão, sendo 64 e 192 *bits* respectivamente. Os algoritmos AES e o ARC4 realizam a criptografia utilizando operações matemáticas relativamente simples como XOR, o que os torna eficiente tanto a nível de *hardware* como de *software*. Os demais algoritmos utilizam operações de XOR seguidas de outras operações como substituições e transposições para realizar o ciframento, o que torna o processo mais lento.

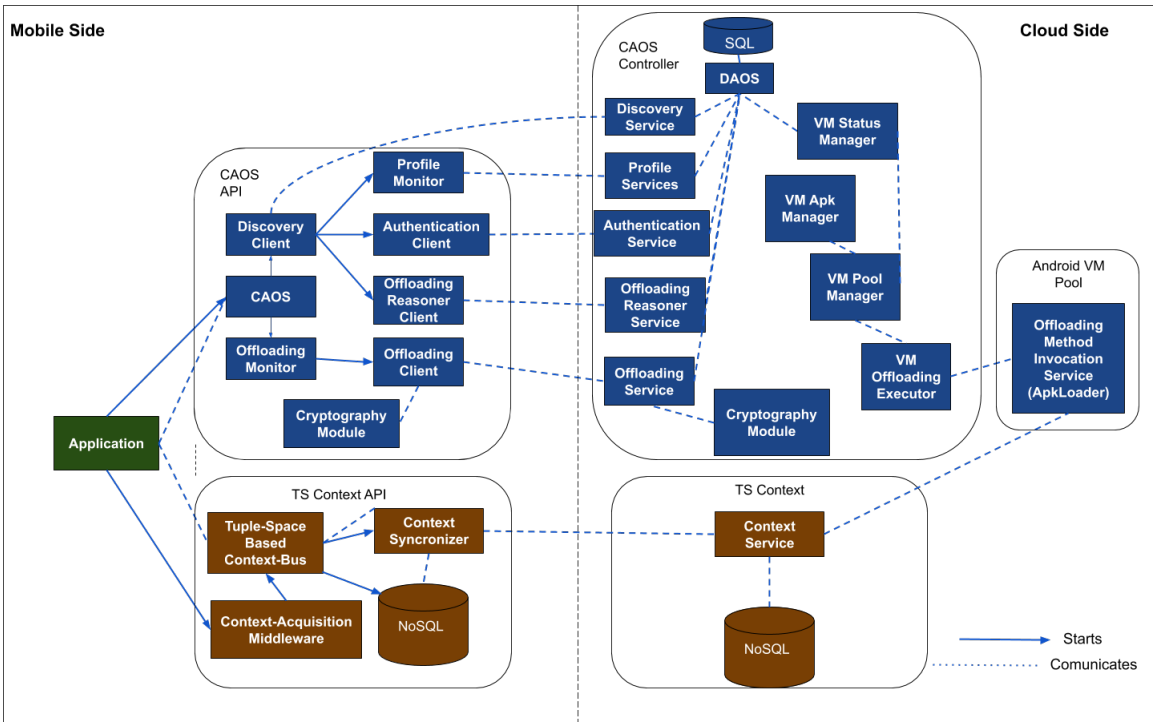
A ideia do presente trabalho é realizar uma análise da execução do *offloading* do dispositivo móvel para um ambiente remoto, utilizando os algoritmos criptográficos escolhidos para garantir a confidencialidade do objeto transmitido.

4.2 Arquitetura

Para alcançar os objetivos do presente trabalho, foram implementados dois módulos adicionais para o *framework* CAOS, um alocado na parte do Cliente e outro no Servidor, onde os mesmos realizam a criptografia dos dados que são migrados do dispositivo móvel para o *cloudlet*.

Os módulos de criptografia foram construídos com o intuito de garantir a segurança e a privacidade dos dados que estão sendo migrados para o *cloudlet* por meio do *offloading*. Foram desenvolvidos através da linguagem Java (mesma linguagem utilizada pelo CAOS) e os algoritmos de criptografia suportados pelo mesmo estão dispostos na Tabela 1, onde uma característica em comum é que todos eles são Simétricos. Também é utilizado o algoritmo Assimétrico RSA (abordado na Seção 2.5.2) para realizar a encriptação da chave para que ela possa ser transmitida de forma segura, sendo assim, o presente trabalho utiliza uma abordagem de criptografia híbrida, com algoritmos Simétricos para a encriptação do objeto e um algoritmo Assimétrico para a encriptação da chave. A Figura 15 ilustra a nova arquitetura do *framework* CAOS com a inserção dos módulos.

Figura 15 – Arquitetura do CAOS com suporte à criptografia



Fonte – Elaborado pelo autor

O funcionamento dos módulos criados ocorre da seguinte maneira: Ao iniciar sua execução, o CAOS Controller gera e armazena seu par de chaves do algoritmo RSA, sendo uma pública e outra privada. Ao iniciar a execução da aplicação, o CAOS API irá se conectar ao Controller e nesse processo ocorre também o envio da chave pública do Controller para a API. Após isso, a API verifica qual o algoritmo de criptografia foi escolhido para essa execução e verifica também a existência da *secretKey* (chave secreta utilizada na criptografia simétrica, discutido melhor na Seção 2.5.1), caso não exista, o módulo de criptografia inserido na API irá tratar de gerar essa chave para o algoritmo em questão, onde cada um dos algoritmos possuem suas próprias restrições de chave, que são abstraídas pelo módulo no processo de criação. Depois de criar a chave, o módulo de criptografia ainda a criptografa utilizando o algoritmo RSA e a chave pública recebida do Controller anteriormente. Após isso, a execução do CAOS continua normalmente.

Ao iniciar o processo de *offloading*, o CAOS API envia sua *secretKey* encriptada para o CAOS Controller e ao recebê-la, realiza as ações necessárias para a decriptação da chave. Logo em seguida, o CAOS API utiliza o módulo de criptografia para encriptar o objeto e só então o transmite para o Controller. Ao receber o objeto encriptado, o CAOS Controller realiza sua decriptação usando o módulo de criptografia e a *secretKey* recebida da API e já decriptada.

Encerrando-se estas etapas, o funcionamento do CAOS continua normalmente e o processo supracitado de encriptação/decriptação ocorre novamente quando o Controller retornar o resultado da execução para a API.

4.3 Considerações Finais

Neste capítulo foi apresentada a proposta de uma extensão do *framework* CAOS, para suportar à criptografia e também uma análise dos impactos no desempenho causados pela sua inserção.

Essa solução busca mitigar o problema da falta de segurança e privacidade trazida na migração de dados e/ou processamento com o uso da técnica de *offloading*. Fez-se necessário a criação de dois módulos adicionais para o *framework* CAOS, responsáveis por lidar com essa criptografia, um inserido na parte do Cliente e outro na parte do Servidor. Para realizar a análise dos impactos causados, fez-se necessário também a criação de uma aplicação que execute

operações que demandam muito poder computacional.

O próximo capítulo apresenta a aplicação supracitada, desenvolvida para a análise da criptografia adicionada no processo de *offloading* utilizando o *framework* CAOS. Além disso, o próximo capítulo irá mostrar os resultados dessa análise em termos de tempo gasto com criptografia, tempo gasto com comunicação, tempo total gasto em todo o processo do *offloading* e uma comparação com e sem a utilização da segurança adicionada com os algoritmos criptográficos.

5 RESULTADOS

Buscando avaliar os impactos causados pela a utilização da criptografia no processo de *offloading*, foi desenvolvida uma aplicação intitulada *MySort*, que utiliza a versão do CAOS proposta nesse trabalho, onde a mesma foi utilizada para realização de testes que avaliaram o tempo de processamento com a utilização da criptografia no dispositivo móvel e no *cloudlet*. A aplicação também simula um cenário no qual o usuário deseja executar uma tarefa que demanda de muito processamento e os dados que serão trabalhados são sensíveis, necessitando então que seja garantido um certo nível de segurança e privacidade para os mesmos.

A Seção 5.1 apresenta a aplicação *MySort*. A Seção 5.2 apresenta uma descrição do ambiente de execução dos testes. A Seção 5.3 apresenta uma descrição do experimento realizado. A Seção 5.4 apresenta os resultados obtidos com os testes realizados com a versão proposta do CAOS.

5.1 Aplicação

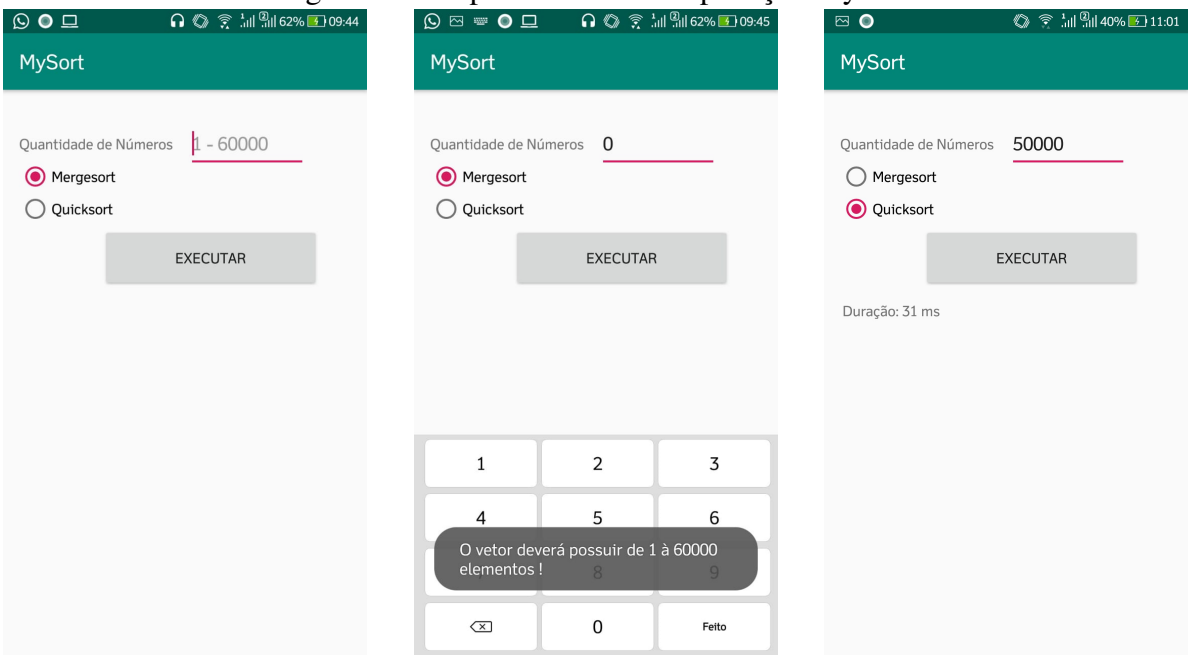
MySort é uma aplicação móvel de *Benchmarking*, isto é, realiza operações computacionalmente complexas possibilitando medir o desempenho em relação ao tempo de execução das operações solicitadas. Ela suporta a utilização do *offloading* com e sem criptografia dos dados utilizando para isso o *framework* CAOS (tratado na Seção 2.4) e o CAOS com a adição dos módulos de criptografia, discutido anteriormente, pois para a utilização dos mesmos basta que seja importado no projeto *Android* a biblioteca pré-codificada de cada um deles (arquivo *.jar*).

A atividade realizada pelo *MySort* é a ordenação de um vetor com elementos, no intervalo de 0 à 1000, gerados aleatoriamente por meio de uma função chamada *generateVector*, que recebe como parâmetro o tamanho do vetor com limiar superior de 60000. O processo de ordenação do vetor é feito utilizando métodos conhecidos, sendo eles o MergeSort Qin (2014) e o QuickSort Yao (2019), sendo que esses algoritmos possuem complexidade, no pior caso O(n * log n) e O (n * n), respectivamente.

A Figura 16a apresenta capturas de tela da aplicação *MySort*, onde o usuário entra com a quantidade de elementos que o vetor deverá possuir, escolhe o algoritmo de ordenação que

deverá ser utilizado na operação e por fim, ao pressionar o botão EXECUTAR será realizado todo o processo de geração dos elementos do vetor bem como a execução do método de ordenação, que pode ser no dispositivo móvel ou no *cloudlet*, através do *offloading* suportado pelo CAOS.

Figura 16 – Capturas de tela da aplicação MySort



(a) Tela inicial

(b) Tela mostrando os limiares de quantidade

(c) Tela mostrando o resultado

Fonte – Próprio autor

5.2 Descrição do Ambiente de Execução

Para o experimento foi utilizado um *smartphone* e um *laptop*. O *smartphone* possui as seguintes características:

- Asus Zenfone 4 ZEK554L
- Processador de 64 *bits* Qualcomm Snapdragon 630 2.2Ghz Octa-Core
- 4GB de memória RAM
- 64GB de memória interna
- Android 8.0.0 (Google API 26)

O *laptop*, que neste experimento funciona como um *cloudlet*, tem as seguintes configurações:

- Samsung Expert + GFX X40

- Processador Intel Core i5-8250U (1.60GHz até 3.40GHz, 6MB L3 Cache) 8ª geração
- 8GB DDR4 de memória RAM
- 1TB de disco rígido
- Placa de Vídeo 2GB
- Linux Mint 19.1 Cinnamon

5.3 Descrição do Experimento

O experimento consistiu na utilização da aplicação *MySort* para medir o desempenho dos algoritmos criptográficos quando utilizado na técnica de *offloading*. Para o trabalho, foi medido o tempo gasto com a criptografia dos dados, o tempo gasto com a comunicação e o tempo total de execução de todo o processo da requisição de *offloading* com criptografia utilizando a versão proposta do CAOS. Além disso, foi utilizado a versão anterior do CAOS para realizar uma comparação do tempo total da execução de todo o processo nas duas versões.

Para a medição do tempo total de execução para o *offloading* sem a criptografia, o cálculo foi feito marcando o instante em que a execução foi solicitada pelo usuário e o instante que a aplicação recebe o resultado do processamento oriundo do *cloudlet* apresentando para o usuário a resposta da execução, posteriormente a métrica é dada pela subtração dessas duas grandezas.

Para a medição do tempo gasto com a criptografia dos dados e o tempo gasto com a comunicação, utilizando a versão proposta do CAOS com criptografia, os cálculos utilizados foram feitos da mesma forma, marcando o instante de início e fim da operação de interesse. O tempo gasto com a criptografia é dado pela soma dos tempos gastos com a encriptação e decriptação dos dados tanto no dispositivo móvel quanto no *cloudlet* e o tempo gasto com a comunicação é dado pela soma dos tempos gastos com o *upload* e *download* dos dados no dispositivo móvel e no cloudlet. A unidade de tempo de todas as métricas está em milissegundos (ms).

Para a realização do experimento, foram escolhidos quatro intervalos de quantidade de números para ordenação na aplicação *MySort*. A escolha desses intervalos ocorreu de forma empírica e a quantidade a ser ordenado dobra para cada cenário de teste, inicialmente com 7500 até chegar em 60000, sendo assim, cada caso de teste consiste na execução do processo para

vetores com as seguintes quantidades de elementos: 7500, 15000, 30000 e 60000. Como o *MySort* possui dois métodos de ordenação, cada uma dessas quantidades foi executado em ambos os métodos.

De acordo com o Teorema Central do Limite (TCL) (FISCHER, 2010), quando se tem uma amostra suficientemente grande, a distribuição de probabilidade da média amostral pode ser aproximada por uma distribuição normal. Essa aproximação é válida a partir de 30 médias amostrais. Assim para cada uma das etapas citadas acima, tanto dos algoritmos de criptografia (i.e., AES, ARC4, Blowfish, DES e 3DES) como dos algoritmos de ordenação, os experimentos foram realizados 30 vezes e os dados coletados foram armazenados em planilhas para posterior análise dos resultados.

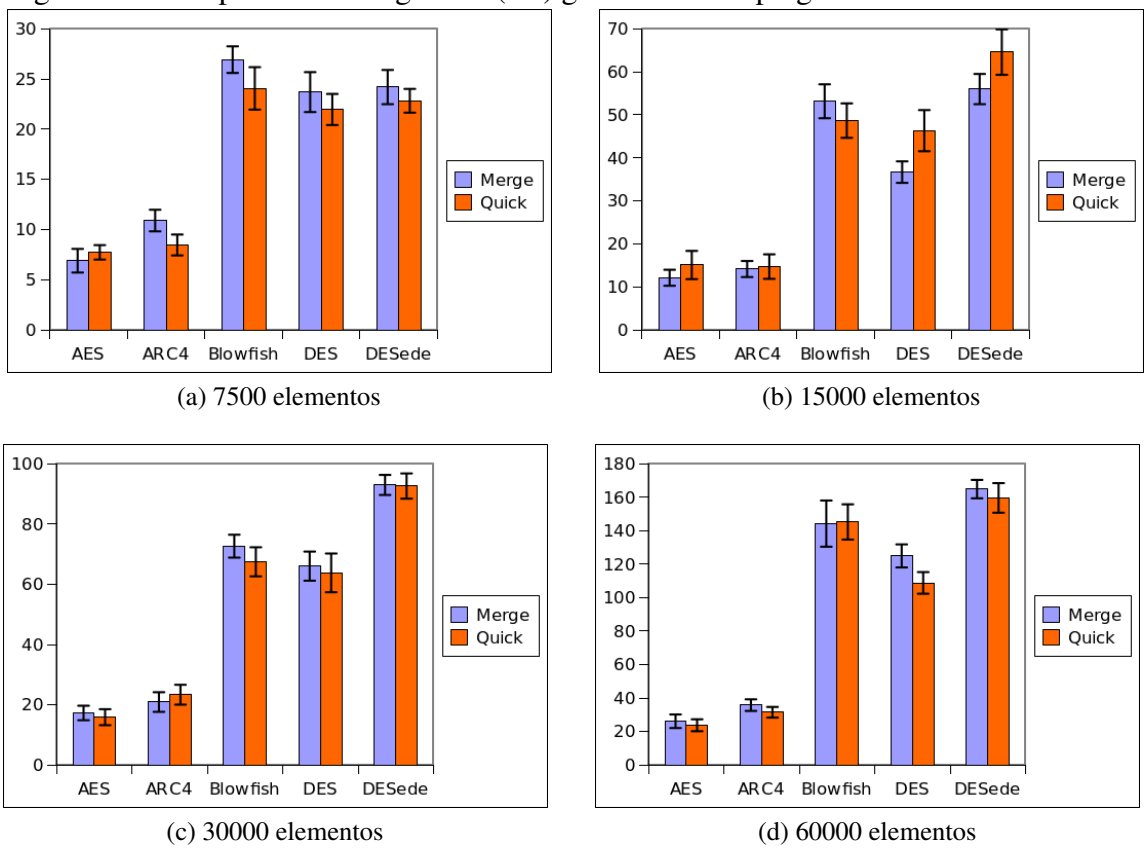
5.4 Resultados Obtidos

Com todos os dados coletados e dispostos em planilhas, foi realizado uma análise dos dados brutos e também foi discutido sobre o que os mesmos mostravam. Como dito anteriormente, cada um dos casos foi executado 30 vezes. Então, para todos os algoritmos de ordenação e criptografia, foram coletadas todas as métricas citadas (i.e., tempo de criptografia, tempo de comunicação e tempo total do processo) e realizado uma média aritmética das 30 execuções realizadas. Para a análise, foi utilizado um intervalo de confiança de 95%. O *software* utilizado na análise dos dados foi o Gnumeric[1].

A Figura 17 apresenta os gráficos contendo o tempo gasto no processo de criptografia analisados (onde a coluna nomeada DESede representa o algoritmo de criptografia 3DES). Assim, possui os dados de cada um dos algoritmos criptográficos para os algoritmos MergeSort e QuickSort, e para os diferentes tamanhos de objeto (i.e., 7500, 15000, 30000 e 60000). A partir dos gráficos apresentados na figura é possível verificar que os algoritmos que se apresentaram mais eficientes foram o AES e o ARC4. Os demais algoritmos criptográficos, independentes dos algoritmos de ordenação, são mais custosos em termos computacionais, assim são ineficientes para o processo de *offloading*. Esses resultados eram esperados, pois como citado no Capítulo 4, as operações matemáticas do AES e ARC4 são mais simples que os demais algoritmos. Portanto, conclui-se que a utilização do AES e ARC4 são os mais indicados quando se desejar realizar o *offloading*, pois irá impactar menos no tempo de execução de tarefas computacionais.

[1] http://www.gnumeric.org/

Figura 17 – Tempo em milissegundos (ms) gasto com a criptografia em todos os cenários



(a) 7500 elementos

(b) 15000 elementos

(c) 30000 elementos

(d) 60000 elementos

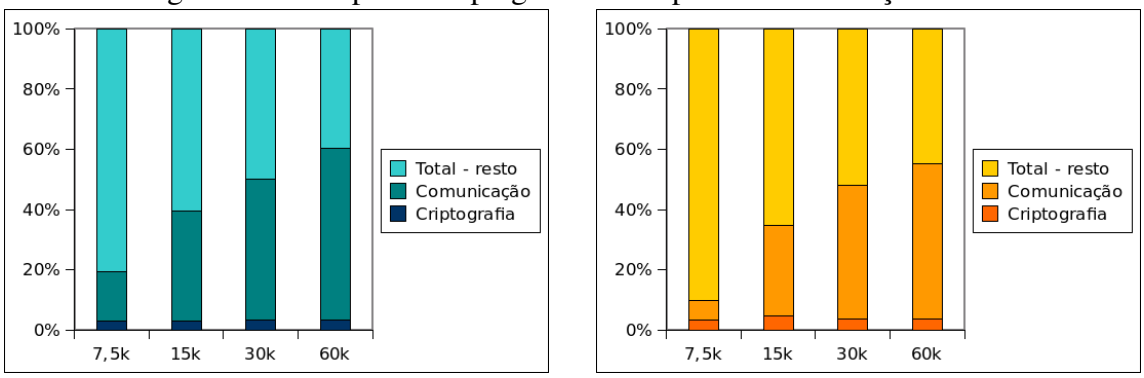
Fonte – Próprio autor

A Figura 18 apresenta o tempo gasto com a criptografia, o tempo gasto com a comunicação e o tempo total gasto no *offloading* descontado todas as medições do tempo gasto com criptografia, comunicação, execução e os *overheads* do processo como um todo. A Figura 18 utiliza o algoritmo criptográfico AES e a Figura 19 utiliza o algoritmo 3DES, as mesmas tem o intuito de mostrar não a influência específica de um algoritmo, mas como a criptografia pode interferir no processo de *offloading*. Esses dois algoritmos foram escolhidos para essa análise pois a ideia é verificar a influência de um algoritmo eficiente (AES) e um algoritmo que mostrou-se computacionalmente mais lento (3DES), de acordo com o análise anterior. Os gráficos apresentam os dados para os dois algoritmos computacionais.

De acordo com os gráficos das figuras, é possível perceber, que quanto maior a quantidade de números na ordenação, maior é o tempo gasto com a comunicação entre o dispositivo móvel e o *cloudlet*. Assim, o tempo gasto com comunicação é uma grandeza diretamente proporcional ao aumento da quantidade de números na ordenação. Em relação a criptografia, verificando o melhor e o pior caso, no AES usando o *MergeSort*, a porcentagem de influência foi de 3.0% $\pm$ 0.51% no caso de 7500 números gerados e de 3.26% $\pm$ 0.51% no caso

de 60000 números gerados e no caso do *QuickSort*, a porcentagem de influência foi de 3.43% $\pm$ 0.31% no caso de 7500 números gerados e de 4.71% $\pm$ 1.02% no caso de 15000 números gerados.

Figura 18 – Tempo de Criptografia x Tempo de Comunicação - AES



(a) MergeSort (b) QuickSort

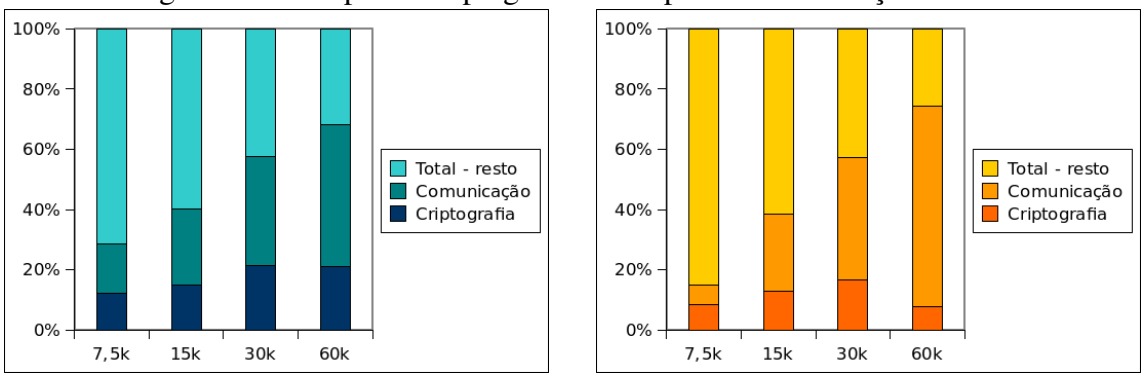
Fonte – Próprio autor

Como no 3DES, o tempo de criptografia foi maior que no AES, o percentual de influência da criptografia foi bem mais significativo, em que verificando o melhor e o pior caso, no *MergeSort* a porcentagem foi de 12.19% $\pm$ 2.47% para 7500 números e 21.54% $\pm$ 0.88% para 30000 e com o *QuickSort* a porcentagem foi de 8.5% $\pm$ 0.44% para 7500 números e 16.51% $\pm$ 0.98% para 30000 números. Assim, estatisticamente, tanto nos objetos com menor e maior tamanho, a criptografia não tem tanto impacto no processo de *offloading* como um todo. Logo, chega-se a conclusão de que utilizar algoritmos criptográficos na técnica de *offloading* trará benefícios, pois sua inserção trará segurança na transmissão dos dados e o aumento no tempo total gasto poderá ser considerado irrelevante, dependendo apenas do desempenho do algoritmo criptográfico empregado.

Após a análise dos resultados atingidos com o *offloading* criptografado em relação ao tempo gasto com as operações de criptografia e a comunicação, foi feita a comparação e análise dos dados obtidos com o tempo total gasto com o processo do *offloading* com e sem criptografia. A Figura 20 apresenta os gráficos com os resultados supracitados onde a coluna nomeada com "WC" representa o *offloading* sem criptografia (do inglês, *Without Cryptography*).

A partir da análise dos gráficos apresentados na Figura 20, foi possível notar que a influência causada pelas operações de criptografia é ligeiramente dominada pelo tempo consumido com a comunicação entre o dispositivo e o *cloudlet*, onde em alguns casos, o *offloading*

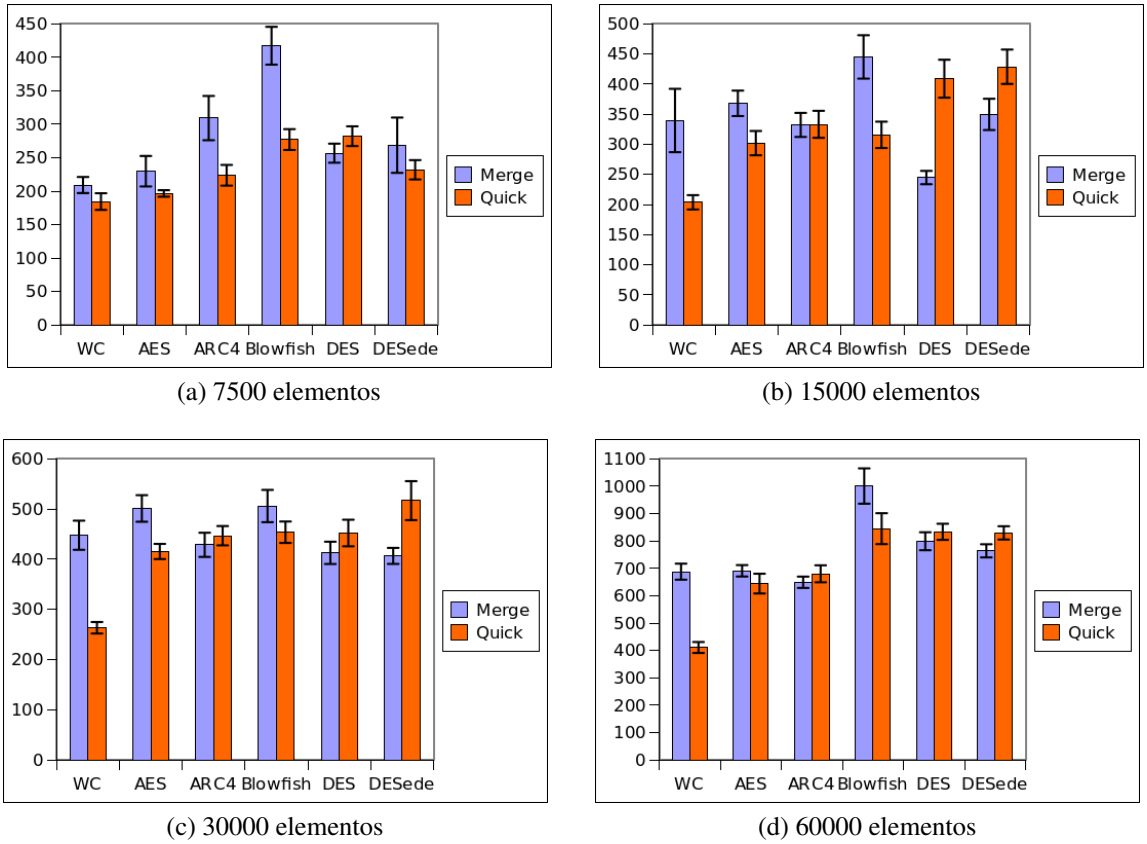
Figura 19 – Tempo de Criptografia x Tempo de Comunicação - 3DES



(a) MergeSort (b) QuickSort

Fonte – Próprio autor

Figura 20 – Tempo total em milissegundos (ms) gasto com o *offloading* em todos os cenários



(a) 7500 elementos (b) 15000 elementos

(c) 30000 elementos (d) 60000 elementos

Fonte – Próprio autor

sem criptografia obteve um tempo superior ao com criptografia. Logo, chega-se a conclusão de que o que de fato é primordial na execução do *offloading*, levando-se em consideração o tempo total, é a comunicação entre eles. Assim, deve-se utilizar da segurança proporcionada pelos algoritmos criptográficos, pois esse não é o fator primordial no aumento do tempo da execução de uma tarefa computacional em um ambiente remoto.

5.5 Considerações Finais

Este capítulo apresentou uma aplicação móvel que utiliza a versão proposta do CAOS, a descrição do ambiente da execução dos testes e a descrição do experimento realizado. Os testes realizados foram em relação ao impacto da segurança utilizando algoritmos criptográficos no processo de *offloading*.

A análise dos resultados foi feita usando comparações do desempenho e tempo em geral em cada um dos algoritmos de criptografia. Em relação ao desempenho foi possível notar que os algoritmos de criptografia que se mostraram mais eficientes foram o AES e ARC4, em que o tempo médio gasto com todo processo de criptografia do primeiro consumiu em torno de 3% do tempo médio total gasto em todo o processo e também que o mesmo não variou tanto com o crescimento do objeto. Já o Blowfish, o DES e o 3DES mostraram ser menos efetivos, consumindo mais tempo no processo de criptografia e em geral variando mais a medida que o objeto cresce. Este resultado pode ser observado nos dois algoritmos de ordenação testados, comprovando então que o método a ser executado é indiferente no processo de *offloading* criptografado. Outra observação que pode ser notada é que a adição da criptografia não foi tão relevante no processo de *offloading* de métodos como um todo, pois como visto em todos os casos e para todos os tamanhos de objeto variando de 30 à 235 *megabytes*, o fator que se mostrou de toda importância é a comunicação, que em muitos cenários representou mais de 50% do tempo total na tarefa.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo resume o que foi discutido e desenvolvido neste trabalho de conclusão de curso. A Seção 6.1 apresenta os resultados alcançados com a realização da proposta, enquanto na Seção 6.2 é apresentada as principais limitações encontradas na solução. Por fim, a última Seção apresenta os trabalhos futuros.

6.1 Resultados Alcançados

O presente trabalho apresentou uma extensão do *framework* CAOS, para que ele suporte a criptografia no processo de *offloading*, onde o mesmo traz como vantagem um aumento significante no nível da segurança dos dados que estão sendo migrados durante todo o processo. Com isso, este serviço também traz a possibilidade de desenvolvedores da plataforma *Android* que já se utilizam dos recursos e vantagens do *offloading* por meio do *framework* CAOS, trafeguem seus dados com maior segurança utilizando os módulos de criptografia adicionados.

Após a realização de todos os experimentos e também a análise dos dados, foi possível concluir que a inserção da criptografia no *offloading* de métodos com abordagens semelhantes ao do *framework* CAOS, geralmente, não vai ser o maior influenciador no aumento do tempo total de processamento, pois o tempo de comunicação entre o dispositivo móvel e o *cloudlet* mostrou uma influência bem maior neste sentido, o qual foi possível notar a existência de uma relação em que quanto maior o objeto de requisição para o *offloading* maior será o tempo gasto com a comunicação e que dependendo do algoritmo, a influência dele é insignificante para o processo como um todo.

Assim, a qualidade e a velocidade da conexão entre o dispositivo móvel e o *cloudlet* será o fator determinante no desempenho da execução do *offloading* com criptografia, bem como sem criptografia, que em alguns cenários foi possível observar que obteve valores do tempo total no processo de *offloading* bem próximos dos com criptografia e em alguns casos, chegou a ultrapassar.

6.2 Limitações Encontradas

No decorrer do presente trabalho foram encontradas algumas dificuldades que acabaram por limitando em algumas execuções, o qual inicialmente foram propostas no trabalho de conclusão 1.

A primeira limitação encontrada foi o fato do *smartphone* utilizado nos testes ser relativamente novo (2017) e não possuir bateria removível, o que nos impossibilitou de realizar a avaliação sobre o consumo energético do dispositivo móvel utilizando a criptografia no CAOS. Outra limitação encontrada foi relacionada à implementação do algoritmo de Criptografia Assimétrica RSA, pois o mesmo apresentou uma limitação de quantidade de bytes (256) na implementação com o Android nativo. Por fim, a última limitação sofrida se relaciona a implementação do algoritmo de Criptografia Simétrica ChaCha20 que por ser muito atual, só é oferecido suporte para *smartphones* com API 28+ e o dispositivos utilizado para realizar os testes possui apenas a API 26, bem como a máquina virtual mais recente encontrada, fato que impossibilitou sua implementação.

6.3 Trabalhos Futuros

Como trabalhos futuros, propõe-se algumas melhorias no trabalho, tais como:

- Implementar para o CAOS a opção de escolher em tempo de execução o algoritmo de criptografia utilizado;
- Implementar criptografia assimétrica para o CAOS;
- Implementar uma Anotação Java, que receberá parâmetros (*e.g.*, nível de segurança, desempenho exigido) e partir deles irá definir o algoritmo criptográfico que será utilizado.
- Realizar uma anaĺise do consumo energético do impacto da segurança no processo de *offloading*.

REFERÊNCIAS

ADHIE, R. P.; HUTAMA, Y.; AHMAR, A. S.; SETIAWAN, M. *et al.* Implementation cryptography data encryption standard (des) and triple data encryption standard (3des) method in communication system based near field communication (nfc). In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2018. v. 954, n. 1, p. 012009.

AMARO, G. **Criptografia simétrica e criptografia de chaves públicas: Vantagens e desvantagens**. 2008.

ANDRADE, L. P.; SOARES, D. N.; COUTINHO, M. M.; ABELÉM, A. J. G. Análise das vulnerabilidades de segurança existentes nas redes locais sem fio: Um estudo de caso do projeto wlaca. **Universidade Federal do Pará, Belém, http://www. lprad. ufpa. br/˜ margalho/wdeec/tcc. pdf**, v. 15, 2008.

BELLARE, M.; CANETTI, R.; KRAWCZYK, H. Keying hash functions for message authentication. In: SPRINGER. **Annual international cryptology conference**. [S.l.], 1996. p. 1–15.

BELLARE, M.; DESAI, A.; JOKIPII, E.; ROGAWAY, P. A concrete security treatment of symmetric encryption. In: IEEE. **Proceedings 38th Annual Symposium on Foundations of Computer Science**. [S.l.], 1997. p. 394–403.

BONOMI, F.; MILITO, R.; NATARAJAN, P.; ZHU, J. Fog computing: A platform for internet of things and analytics. In: **Big data and internet of things: A roadmap for smart environments**. [S.l.]: Springer, 2014. p. 169–186.

CHAINTREAU, A.; MTIBAA, A.; MASSOULIE, L.; DIOT, C. The diameter of opportunistic mobile networks. In: ACM. **Proceedings of the 2007 ACM CoNEXT conference**. [S.l.], 2007. p. 12.

CISCO, C. V. N. I. Global mobile data traffic forecast update, 2016–2021. **white paper**, 2017.

COSTA, P. B.; REGO, P. A. L.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Mpos: A multiplatform offloading system. In: **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2015. (SAC '15), p. 577–584. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695945>.

DINH, H. T.; LEE, C.; NIYATO, D.; WANG, P. A survey of mobile cloud computing: architecture, applications, and approaches. **Wireless communications and mobile computing**, Wiley Online Library, v. 13, n. 18, p. 1587–1611, 2013.

DIRO, A. A.; CHILAMKURTI, N.; NAM, Y. Analysis of lightweight encryption scheme for fog-to-things communication. **IEEE Access**, IEEE, v. 6, p. 26820–26830, 2018.

DULLIUS, M. M. O problema do logaritmo discreto. 2001.

DURAO, F.; CARVALHO, J. F. S.; FONSEKA, A.; GARCIA, V. C. A systematic review on cloud computing. **The Journal of Supercomputing**, Springer, v. 68, n. 3, p. 1321–1346, 2014.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. **Future generation computer systems**, Elsevier, v. 29, n. 1, p. 84–106, 2013.

FISCHER, H. **A history of the central limit theorem: From classical to modern probability theory**. [S.l.]: Springer Science & Business Media, 2010.

FORMAN, G. H.; ZAHORJAN, J. The challenges of mobile computing. **Computer**, IEEE, v. 27, n. 4, p. 38–47, 1994.

GOMES, F. A.; REGO, P. A.; ROCHA, L.; SOUZA, J. N. de; TRINTA, F. Caos: A context acquisition and offloading system. In: IEEE. **2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.], 2017. v. 1, p. 957–966.

GOMES, F. A.; VIANA, W.; ROCHA, L. S.; TRINTA, F. A contextual data offloading service with privacy support. In: ACM. **Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web**. [S.l.], 2016. p. 23–30.

GUPTA, A. K. Challenges of mobile computing. In: **Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology (COIT-2008). Mandi Gobindgarh, India: RIMT-IET**. [S.l.: s.n.], 2008. p. 86–90.

HUSSEIN, N. H.; KHALID, A.; KHANFAR, K. A survey of cryptography cloud storage techniques. **Int J Comput Sci Mobile Comput**, v. 5, n. 2, p. 186–191, 2016.

JHURIA, M.; SINGH, S.; NIGOTI, R. A survey of cryptographic algorithms for cloud computing. **International Journal of Emerging Technologies in Computational and Applied Sciences**, 05 2013.

JUNIPER, R. **Mobile Payments to Rise 40% This Year, Juniper Research Finds**. 2014. <https://www.juniperresearch.com/press-release/mobile-payments-pr1>. Accessed: 2019-03-29.

JUNIPER, R. **Mobile Ticketing Users to Reach 1.9 Billion by 2023, Catalysing Urban Mobility Revolution**. 2019. <https://www.juniperresearch.com/press/press-releases/mobile-ticketing-users-to-reach-1-9-billion>. Accessed: 2019-03-29.

KHARBANDA, H.; KRISHNAN, M.; CAMPBELL, R. H. Synergy: A middleware for energy conservation in mobile devices. In: IEEE. **2012 IEEE International Conference on Cluster Computing**. [S.l.], 2012. p. 54–62.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

MELL, P.; GRANCE, T. *et al.* The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

NAGAR, S. A.; ALSHAMMA, S. High speed implementation of rsa algorithm with modified keys exchange. In: IEEE. **2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)**. [S.l.], 2012. p. 639–642.

NIR, Y.; LANGLEY, A. **ChaCha20 and Poly1305 for IETF Protocols**. [S.l.], 2018.

OLIVEIRA, R. R. Criptografia simétrica e assimétrica-os principais algoritmos de cifragem. **Segurança Digital [Revista online]**, v. 31, p. 11–15, 2012.

PADHI, S.; TIWARY, M.; PRIYADARSHINI, R.; PANIGRAHI, C. R.; MISRA, R. Secomn: Improved security approach for opportunistic mobile networks using cyber foraging. In: IEEE. **2016 3rd International Conference on Recent Advances in Information Technology (RAIT)**. [S.l.], 2016. p. 415–421.

PEREZ, S. Mobile cloud computing: $9.5 billion by 2014. **Technical Reprot, ReadWriteMobile**, 2010.

PORTNOI, M. Criptografia com curvas elipticas'. **ArXiv e-prints**, 2005.

PRENEEL, B. **Analysis and design of cryptographic hash functions**. Tese (Doutorado) — Citeseer, 1993.

QIN, S. merge sort algorithm. **Florida Institute of Technology**, 2014.

QURESHI, S. S.; AHMAD, T.; RAFIQUE, K. *et al.* Mobile cloud computing as future for mobile applications-implementation methods and challenging issues. In: IEEE. **2011 IEEE International Conference on Cloud Computing and Intelligence Systems**. [S.l.], 2011. p. 467–471.

RAHIMI, M. R.; REN, J.; LIU, C. H.; VASILAKOS, A. V.; VENKATASUBRAMANIAN, N. Mobile cloud computing: A survey, state of art and future directions. **Mobile Networks and Applications**, Springer, v. 19, n. 2, p. 133–143, 2014.

REGO, P. A. L. **Applying Smart Decisions, Adaptive Monitoring and Mobility Support for Enhancing Offloading Systems**. Tese (Doutorado) — Universidade Federal do Ceará, 2016.

SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; BUYYA, R. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 369–392, First 2014. ISSN 1553-877X.

SANTOS, G. B. dos; REGO, P. A.; TRINTA, F. Uma proposta de solução para offloading de métodos entre dispositivos móveis. In: SBC. **Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web**. [S.l.], 2017. p. 76–81.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **Pervasive Computing, IEEE**, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1536-1268.

SCHNEIER, B. Description of a new variable-length key, 64-bit block cipher (blowfish). In: SPRINGER. **International Workshop on Fast Software Encryption**. [S.l.], 1993. p. 191–204.

SHIRAZ, M.; GANI, A.; KHOKHAR, R.; BUYYA, R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. **Communications Surveys Tutorials, IEEE**, v. 15, n. 3, p. 1294–1313, Third 2013. ISSN 1553-877X.

SILVA, B.; SABINO, A.; JUNIOR, W.; OLIVEIRA, E.; JÚNIOR, F.; DIAS, K. Performance evaluation of cryptography on middleware-based computational offloading. In: IEEE. **2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.], 2017. p. 205–210.

STINSON, D. R. **Cryptography: theory and practice**. [S.l.]: Chapman and Hall/CRC, 2005.

SUJITHRA, M.; PADMAVATHI, G.; NARAYANAN, S. Mobile device data security: a cryptographic approach by outsourcing mobile data to cloud. **Procedia Computer Science**, Elsevier, v. 47, p. 480–485, 2015.

VIDAL, P. V. C. Dependência mobile: a relação da nova geração com os gadgets móveis digitais. 2015.

VIJAYARANI, S.; TAMILARASI, A. An efficient masking technique for sensitive data protection. In: IEEE. **2011 International Conference on Recent Trends in Information Technology (ICRTIT)**. [S.l.], 2011. p. 1245–1249.

YAO, Y. A detailed analysis of quicksort algorithms with experimental mathematics. **arXiv preprint arXiv:1905.00118**, 2019.