A Study about the Impact of Encryption Support on a Mobile Cloud Computing Framework

Francisco A. A. Gomes, Paulo A. L. Rego, Fernando A. M. Trinta, Windson Viana,
Francisco A. Silva, José A. F. de Macêdo and José N. de Souza
Group of Computer Networks, Software Engineering and Systems (GREat) Federal University of Ceará (UFC),

Keywords: Offloading, Encryption, Performance Evaluation.

Abstract: Mobile Cloud Computing joins two complementary paradigms by allowing the migration of tasks and data from resource-constrained devices into remote servers with higher processing capabilities in an approach known as offloading. An essential aspect of any offloading solution is the privacy support of the information transferred among mobile devices and remote servers. A common solution to address privacy issues in data transmission is the use of encryption. Nevertheless, encryption algorithms impose additional processing tasks that impact both on the offloading performance and the power consumption of mobile devices. This paper presents a study on the impact of using cryptographic algorithms in the CAOS offloading platform. Results from experiments show that the encryption time represents 2.17% to 5.35% of the total offloading time depending on the amount of offloaded data, encryption key size, and the place to run the offloaded task. Similar behavior occurred regarding energy consumption.

1 INTRODUCTION

According to (Chen and Hao, 2018), Mobile Cloud Computing (MCC) is a novel approach for mobile applications (apps) aiming at providing a range of services, equivalents to the cloud, adapted to the capacity of resource-constrained devices, besides performing improvements of telecommunications infrastructure to improve the service provisioning. According to a research report published by MarketsandMarkets[1], the edge computing market size is expected to grow from $2.8 billion in 2019 to $9.0 billion by 2024. The key factors driving the edge computing market include the growing adoption of the Internet of Things (IoT) across industries and rising demand for low-latency processing and real-time.

MCC addresses especially apps that are very sensitive to high network delays due to the communication overhead between mobile devices and data centers resources located in the core of the current Internet infrastructure, and far away from the network edge. Some examples include real-time mobile games, crowdsensing systems, and augmented reality applications (Varghese and Buyya, 2018). In MCC, the most common research topic is offloading, which represents the idea of moving data and processes from mobile devices with scarce resources to more powerful machines (Abbas et al., 2018). Many research has been done on the offloading topic, and several frameworks have been proposed to provide offloading features in mobile apps (Rego et al., 2016). One of these solutions is CAOS (Gomes et al., 2017b), a software infrastructure to support the development of mobile context-aware applications based on the Android platform. CAOS provides offloading features to enable the processing of contextual data from mobile devices into cloud platforms. CAOS also has a version called CAOS Device-to-Device (D2D), which supports offloading between mobile devices (Dos Santos et al., 2018). Both CAOS and CAOS D2D monitor the application life-cycle on a local mobile device, and decide whether it is worthy or not to offload a method and its parameters to a remote mobile device. Despite its potential, MCC has several challenges, such as the privacy and security of sensitive data used on offloadable processes (Gupta et al., 2018). Protecting the user privacy enforces consumers' trust in a mobile or cloud platform. However, it is challenging to achieve privacy on MCC systems once the data

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

Gomes, F., Rego, P., Trinta, F., Viana, W., Silva, F., F. de Macêdo, J. and N. de Souza, J.
A Study about the Impact of Encryption Support on a Mobile Cloud Computing Framework.
DOI: 10.5220/0009420804000407
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 400-407
ISBN: 978-989-758-424-4

transferred between mobile devices and remote nodes (such as methods parameters) may include user's sensitive data (Mollah et al., 2017). So, MCC scenarios require techniques such as encryption to offload methods into untrusted nodes. However, this approach increases the overall round-trip time to exchange messages between mobile devices and offloading servers, such as cloudlets(Satyanarayanan et al., 2009). Encryption also increases energy consumption, which is a crucial problem for current mobile applications in general.

This paper addresses this issue by presenting a study about the impact of supporting encryption in both CAOS and CAOS D2D frameworks. Our research goal is to measure the overhead of introducing cryptographic algorithms in both frameworks. The remainder of this paper is organized as follows. Section 2 presents related works to our study. The third section presents both CAOS and CAOS D2D frameworks. Section 4 describe evaluation experiments performed to measure CAOS performance and energy consumption after including encryption mechanisms. Finally, Section 5 concludes the paper and outlines possible future work.

2 RELATED WORK

We can find in the scientific literature several works focusing on the computation offloading topic (Sanaei et al., 2014), (Fernando et al., 2013), (Zhang et al., 2012), (Dinh et al., 2011). However, few of them take into account message privacy using encryption in the offloading between mobile devices and remote environments. Most of them does not evaluate the impact of encryption on offloading performance or energy consumption. For instance, two studies in the literature have designed frameworks to ensure privacy and authentication in mobile cloud services. The Mobile Cloud Authenticator framework (Donald and Arockiam, 2015) provides authentication of mobile devices in MCC. The framework protects users credentials and prevents unauthorized access to cloud services. (Gomes et al., 2017a) introduces COP, a service to support the design of mobile apps that use contextual data from multiple users, such as those on crowdsourcing scenarios. The COP service aims at storing and processing the contextual data gathered from multiple mobile devices into cloud/cloudlet servers. COP enables privacy by filtering the data exchanged between mobile devices and remote services using specific policies. The end-user is responsible for choosing which contextual information he wants to share. However, COP does not support encryption on the exchanged data. Silva *et al.* (Silva et al., 2017) studied how to improve mobile apps running on devices with scarce resources by using the MCC paradigm. The authors extended the MpOS framework (Costa et al., 2015) to add cryptographic mechanisms - symmetric (AES) and asymmetric (RSA) algorithms - in an offloading environment. Their experiments show that encryption affects energy consumption and performance, but the authors did not measure the amount of time spent on encrypting/decrypting tasks separately, neither presented the impact of the network connection in the performance result. Besides that, they did not evaluate any solution that performs offloading among mobile devices. In (Diro et al., 2018), the authors state that the Internet of Things (IoT) presents new challenges concerning security support and claim that these challenges should not be addressed by IoT devices, once they have scarce resources to process any additional task, such as encryption ones. A possible solution is Fog Computing, where encryption tasks may be offloaded to fog nodes to reduce computational and storage loads on IoT devices. Diro *et al.* state that lightweight cryptographic functions, such as elliptic curve cryptography, have proven to be suitable for embedded systems, rather than solutions such as asymmetric cryptography. Unlike our work, they did not measured the impact of network quality on the offloading procedures, neither they evaluated energy consumption on their experiments. Padhi *et al.* (Padhi et al., 2016) propose a cloudlet-based solution for opportunistic mobile networks called SecOMN. Their solution introduces a hybrid encryption algorithm that combines both symmetric and asymmetric key cryptography systems. SecOMN relies on offloading tasks encryption to remote nodes with better processing capabilities. However, they presented no experiments results.

3 CAOS

CAOS is a software infrastructure for the development of mobile context-aware applications based on the Android platform, which provides offloading mechanisms to delegate the migration and processing of contextual data from mobile devices into cloud platforms (Gomes et al., 2017b). CAOS allows Android programmers to mark which methods should be offload to remote servers. The framework uses a hybrid decision-making strategy to decide if it is worthy to migrate a particular method to a remote node, such a cloud or cloudlet node. CAOS D2D is an alternative version of CAOS that provides a subset of the

original CAOS framework but specially designed to offload methods into other mobile devices (Dos Santos et al., 2018). Both CAOS and CAOS D2D are based on a client/server architecture, as shown in Figure 1. Components present two corresponding modules, on both client and server side. Figure 1 shows the CAOS/CAOS D2D architecture where its main components are divided into three tier: CAOS API, CAOS Server, and CAOS D2D Server.

3.1 CAOS API

The CAOS API runs on client mobile device and is composed by 6 (six) components: *Discovery and Deployment Client*, *Profile Monitor*, *Authentication Client*, *Security Service*, *Offloading Client*, and *Context Client*. The *Discovery Client* uses a mechanism based on UDP/Multicast to discover CAOS Servers running in the user's local network (i.e., CAOS D2D server or cloudlets). The *Deployment Client* injects dependencies on the CAOS Server. Dependency is a mobile app copy (APK) that needs to be saved on the server. The *Authentication Client* sends device data to the server-side to keep the list of devices attached to a specific CAOS infrastructure and allow the exchange of keys between the client and server-side of the framework *Authentication Service* (Section 3.3). CAOS monitors the mobile application life-cycle and intercepts its execution flow whenever an annotated method is called. In CAOS, the application's methods can be marked with a Java annotation - *@Offloadable*, which denotes that those methods must run, preferably out of the device. After intercepting the method call, the CAOS starts the offloading process or not. CAOS performs a decision-making process to decide whether it is worthy or not to perform a method offloading. The process is performed in two steps: one at the server-side, and another on the mobile side. The cloud side keeps receiving profiling data from each mobile device connected to its infrastructure and creates a decision tree-based structure with the main metrics used on the decision-making process, such as latency, parameter types, amount of data, and so on. This data structure is sent back to the mobile device that has only to enforce the decision based on the current values of the monitored data (Rego et al., 2019). By using a decision tree-based structure created on the CAOS server, the CAOS can decide locally on the mobile device when it is worth to perform an offloading process. If the answer is negative, the method execution flow is resumed and the method is performed locally. Otherwise, the CAOS requests the *Offloading Client* to start the method offloading process, which in turn, transfers the method and its parameters to the *Offloading Service* in the cloud side. The *Profile Monitor* module is responsible for monitoring the mobile device environment (e.g., network bandwidth and latency, power, and memory status). It sends such information periodically to the *Profile Services*. CAOS server-side uses these data to create the decision tree structure based on the mobile device information and then sends it back to the mobile side. All context information of each mobile device connected to the CAOS is sent by *Context Client* to the *Context Service* to keep a database of contextual information history. The idea is to explore the global context (i.e., the context of all mobile devices) to provide more accurate and rich context information. The *Context Client* exchanges contextual data between the mobile and the cloud sides. In CAOS, filters can be performed in context information repositories. If an application has an *@Offloadable* marked method that accesses context information using the filter concept, and it can benefit itself from the offloading process to access the global context repository. CAOS provides two classes of filters: one to be performed locally (on the mobile device) and other that runs in the global context repository when the method is offloaded to the cloud. The components of the contextual data do not exist in CAOS D2D.

3.2 The Server Mobile/Cloudlet/Cloud Tier

The CAOS server tier is divided by mobile and cloudlet/cloud. The first runs the CAOS D2D version and second runs the CAOS version for cloudlet/cloud. Both versions present five components: *Discovery and Deployment Service*, *Profile Service*, *Authentication Service*, *Security Service* and *Offloading Service*. The *Discovery Service* provides the correct endpoints for clients access to the CAOS Services. The *Deployment Service* receive dependencies from client applications, and store them in the server file system, to enable offloading for those applications. The *Authentication Service* controls which devices are currently connected to the CAOS services, besides ensuring secure authentication by means of the *Security Service* (see Section 3.3). The *Profile Service* is a set of services that receive device data related to connectivity quality and local execution time of offloadable methods, to keep a historical evaluation of the elapsed time for these methods. These records may be used to decide if a method should be offloadable or not. The *Offloading Service* receives offloading requests directly from *Offloading Client* and redirects to *VM Pool Service*, in cloudlet/cloud tier, or directy on mobile server. When the offloading process finishes, the

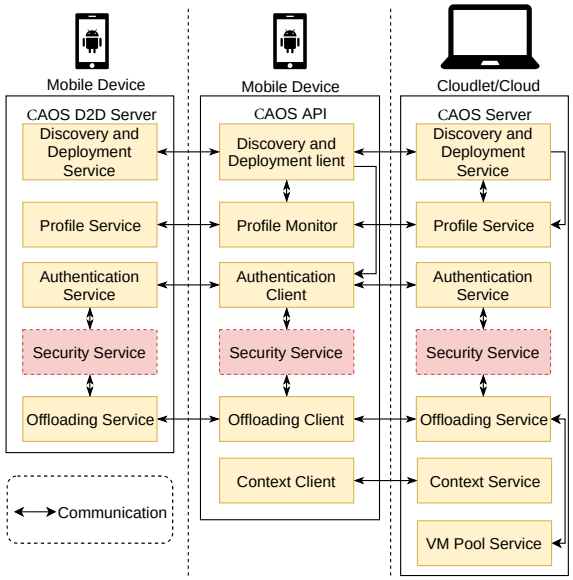


Figure 1: Overview of the CAOS/CAOS D2D Architecture.

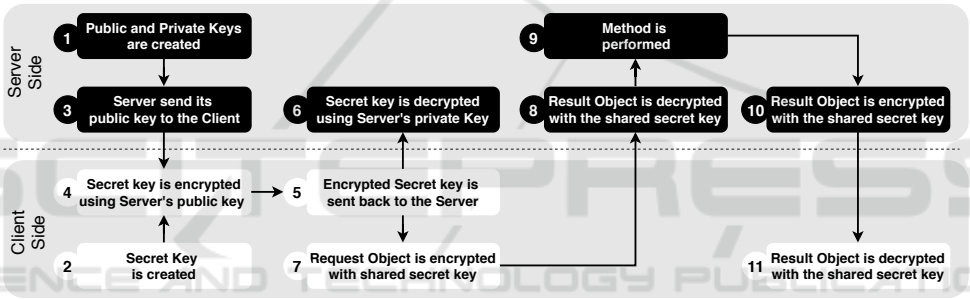


Figure 2: Security Service Flow.

Offloading Service returns the result to the *Offloading Client* and persists offloading information. The *VM Pool Service*, in cloudlet/cloud tier, is responsible for providing an environment that redirects offloading requests to a proper Android Virtual Machine where the offloading execution happens. The *Context Service*, in cloudlet/cloud tier, acts as a global context repository, and stores all context information data sent from all mobile devices connected to the CAOS Services.

3.3 Security Service

In order to protect offloaded data, the *Security Service* component has been added to the CAOS framework. This service provides modules on both client and server sides and is responsible for providing encryption support to the entire migration process. Symmetric cryptographic schemes have lower computational cost and faster encryption/decryption speed but less secure as compared to asymmetric key schemes. A hybrid cryptosystem, including both symmetric and asymmetric algorithms, provides an efficient security solution without compromising on any of their features(Bhatia and Verma, 2017). Our proposed module uses a hybrid encryption solution, which consists of combining symmetric (AES) and asymmetric (RSA) encryption algorithm recommended for key exchange by NIST(Barker et al., 2018). The Figure 2 represents the flow of the security service. Each flow action is detailed as follows: **1.** When the server is started, both public and private keys (asymmetric encryption) are generated; **2.** When the client is started, the secret key (symmetric encryption) is generated; **3.** The generated public key is sent to the client through the *Authentication Service*; **4.** On the client-side, we use the server's public key to encrypt the secret key created by the client; **5.** The encrypted secret key is sent back to the server; **6.** The encrypted secret key is decrypted using server's private key; **7.** Before offloading a method, the client encrypts the request object (the method and its parameters) using the shared secret key; **8.** The server decrypts the request object

using the shared secret key; **9.** Then the method is executed on the server side; **10.** After executing the method, the server encrypts the result object using the shared secret key and sends it to the client; **11.** The client receives the result object, and decrypts it using the shared secret key.

4 EVALUATION

This section presents the experiments performed to evaluate the performance of mobile applications and energy consumption of a mobile device when using CAOS and CAOS D2D to offload methods with encrypted data. To the best of our knowledge, there are no available benchmarks for MCC. For this reason, we used an image processing application (Rego et al., 2016). That allows users to apply photo effects using filters into images with different resolutions, where each one of the filters requires distinct computation requirements.

4.1 Experimental Groups and Procedure

We divided the experiment into two parts. For the first part of the tests, we used an Android mobile device and a cloudlet running the CAOS platform. For the second part, we used two Android mobile devices - one acting as a client and one acting as a CAOS D2D server. For each of these two scenarios, one mobile device had the image processing application installed. A method of this app was marked using the annotation `@Offloadable`. This method is responsible for applying a red color filter to the selected image (`RedTone`). During the experiments, the method was offloaded to a cloudlet (first scenario) and a mobile device (second scenario). We executed the method using a picture with different resolutions (1MP, 2MP, and 4MP) and varying the AES encryption key size (128, 192, and 256 bits). For each setup (i.e., scenario, image size, key size), we executed 30 times the method, totalising eighteen distinct setups and 540 method executions.

4.2 Material and Methods

For the first part of the tests, we used an Android device and a cloudlet running the CAOS platform. The device was a Samsung S4 Active handset (H1) that runs Android 5.0.2 and has 2 GB of RAM memory and processor Snapdragon 600 Qualcomm (1.9 GHz quad-core). As a cloudlet, we used a laptop running Linux Mint 17.2 64 bit operating system, with 8 GB RAM and Core i5-4200U (1.6 GHz Quad-Core) processor. These devices were connected through a dedicated 802.11n wireless network access point. For the second part of the experiment, the handset used in the first experiment (H1) played the role of a client, while the server device (SD) was an LG G3 Beat handset with Qualcomm Snapdragon 400 MSM8226 Cortex-A7 1.2 GHz Quad-Core and 1 GB RAM, running Android 5.0.2. These devices were connected through the same access point used in the previous experiment. Besides measuring the execution time of the methods, we used the Monsoon Power Monitor[2] to measure the energy consumption of both client and server devices during the methods' execution.

4.3 Experiments Results

Figure 3 presents the mean and the 95% confidence interval of the time required for H1 to offload the image processing method to the cloudlet using CAOS. The total time ranges from the client's request until the response from the server arrives, which includes the times required to upload the method's arguments, execute the method on the cloudlet, download the method's return data, besides encryption-related and other overhead times. These times are presented in different colors in the graphics. As we can see, the larger the image to be processed, the longer the total offloading time. The offloading times range approximately from 1.5 seconds to 6.0 seconds, depending on the image resolution. In most cases, there is no statistical difference when comparing the execution time with different sizes of cryptographic keys. Indeed, in some cases, we can even notice that the offloading time is shorter when using the larger cryptographic key (e.g., when H1 processes a 1MP image) - mainly because of the other parameters that compose the total offloading time. Figure 4 shows the mean and the 95% confidence interval of the time required for H1 to offload the image processing method to the handset running the CAOS D2D server. One can see that the larger the image to be processed, the longer the total offloading time and the total offloading time using CAOS D2D is two to four times greater than using CAOS, mainly because of the method execution time on SD is greater than on the cloudlet.

As we can see, the total offloading time mainly depends on the size of the image that will be processed, the processing power of the equipment where the method will be offloaded to and executed, and the communication (i.e., upload/download speed) between client and server. In order to understand the real impact of encryption on offloading performance, we

[2]https://www.msoon.com/high-voltage-power-monitor

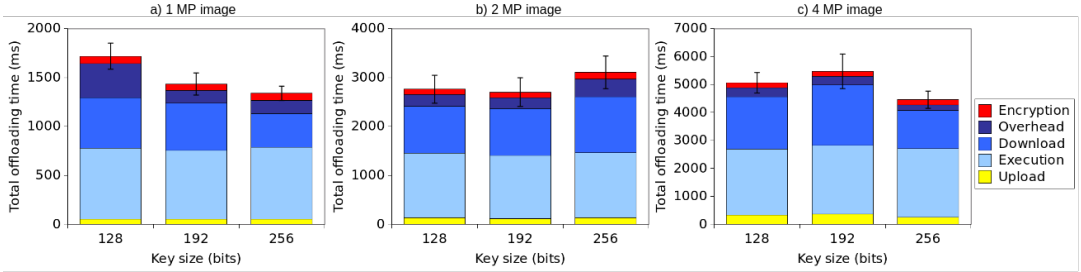


Figure 3: The Total Offloading Time between H1 and the Cloudlet Using CAOS.

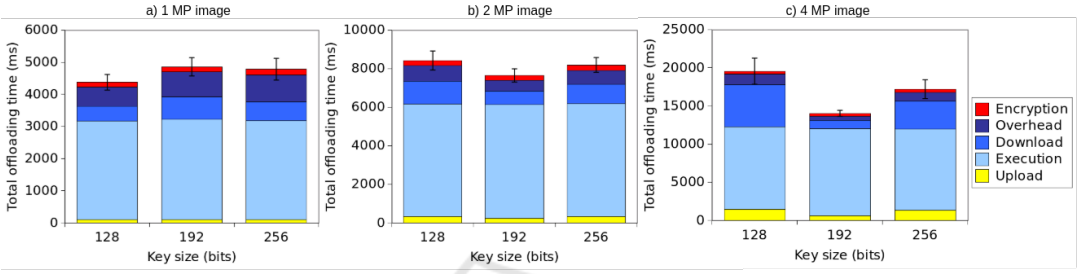


Figure 4: The Total Offloading Time between H1 and the SD Using CAOS D2D.

Table 1: H1's Encryption Time (Ms) When Offloading to Cloudlet and SD Using Different Image Sizes (MP).

	H1 - CAOS			H1 - CAOS D2D		
Key(bits) / MP	128	192	256	128	192	256
1	73.36±13.60	66.56±9.63	72.36±11.28	150.66±13.89	161.83±16.87	172.86±15.53
2	105.63±10.54	114.00±7.82	137.23±14.25	266.76±13.20	245.43±17.31	285.83±20.85
4	181.73±9.50	183.10±12.69	192.03±19.24	404.93±19.91	385.53±15.31	411.63±16.60

Table 2: H1's Energy Consumption (mJ) When Offloading to Cloudlet and SD Using Different Image Sizes (MP).

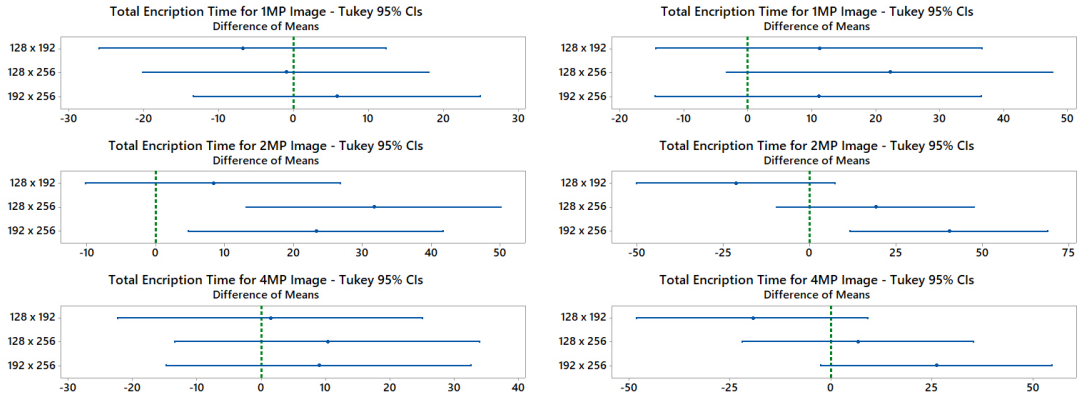
	H1 - CAOS			H1 - CAOS		
Key(bits) / MP	128	192	256	128	192	256
1	156,54 ± 29,02	142,03 ± 20,54	154,40 ± 24,07	321,49 ± 29,63	345,32 ± 35,99	368,86 ± 33,13
2	225,40 ± 22,49	243,26 ± 16,68	292,83 ± 30,40	569,23 ± 28,16	523,72 ± 36,93	609,93 ± 44,49
4	387,79 ± 20,27	390,71 ± 27,07	409,77 ± 41,05	864,08 ± 42,48	822,68 ± 32,67	878,37 ± 35,42

further analyze the results. Thus, Table 1 presents the mean and the 95% confidence interval time required to perform the encryption process (TEP) throughout the method's offloading from client to server (cloudlet or device). To calculate this value, the following equation was used:

$$TEP = TEreq + TDreq + TEres + TDres \quad (1)$$

in which, $TEreq$ is the time to encrypt the client-side request, $TDreq$ is the time to decrypt the server-side request, $TEres$ is the encryption time of the server-side response, and $TDres$ is the time to decrypt the server-side response.

Table 1 shows that the larger the cryptographic key and the image resolution, the longer the total encryption time when offloading to both cloudlet and SD. Also, the encryption time is up to 90% longer when using CAOS D2D because the encryption process also depends on the server's processing capability, which is a mobile device in this case. The same behavior can be observed in Table 2, which presents the mean and the 95% confidence interval for the energy consumed by the mobile device for all encryption-related operations. As we can see, the use of encryption consumes from 142 mJ to 878 mJ depending on the image size, devices, and framework used. Therefore, the longer the encryption key and the bigger the data to be transferred, the higher the energy consumption. We used ANOVA and Tukey statistical tests to evaluate whether the difference between encryption times is significant or not when we change the encryption key size (i.e., 128, 192, 256 bits). As we mentioned, for each image size and scenario, we had three groups of experiments varying the encryption key size. Each ANOVA test compares the



(a) Cloudlet - CAOS Server (b) SD - CAOS D2D

Figure 5: Tukey's Significance Test for Total Encryption Time on H1.

Table 3: Proportion of Encryption Time and Total Offloading Time (%).

Key(bits) / MP	H1 - CAOS			H1 - CAOS D2D		
	128	**192**	**256**	**128**	**192**	**256**
1	4.19±0.56	4.69±0.59	5.35±0.64	3.65±0.29	3.38±0.37	3.65±0.29
2	3.90±0.32	3.82±0.25	3.62±0.30	3.10±0.15	3.06±0.14	3.18±0.19
4	3.70±0.29	3.49±0.29	4.34±0.32	2.17±0.20	2.76±0.13	2.45±0.14

means of these three sample groups. Then, we used ANOVA to test the null hypothesis. If H0 is rejected, we applied the Tukey-Kramer procedure to determine which pairs of means have statistically significant differences. Tukey's plot compares two by two samples on the left side. If the respective confidence interval does not contain zero, the corresponding means are significantly different. Figure 5 illustrates six plots of the Tukey's test for H1 to offload the method to the Cloudlet (a) and the SD (b) using 128, 192 and 256-bits key sizes. The results when offloading to the Cloudlet (plots (a)) show that, for 1 and 4MP images, there is no significant difference between all means when using distinct key sizes. For 1MP images, ANOVA's $p-value = 0.66$, while for 4MP images, $p-value = 0.534$, which means we cannot reject the hypothesis that all means are equal. This result can be explained by the good processing capacity of H1, which performs encryption in a few time and with little variation in computation time when compared to the total offloading time. For the 2MP images, ANOVA's $p-value = 0$, which indicates all means are not equal. But, as we can see in Tukey's result, at least the means using 128 and 192-bits keys are statistically equal. When offloading to the SD (plots (b)), we can see the same behavior when considering 1 and 4MP images. In both cases, ANOVA's $p-value = 0.123$ and $p-value = 0.084$, respectively, which means we cannot reject the hypothesis that all means are equal. Tukey's plots indicate that the difference between the means is not statistically significant because the range does include zero. For the 2MP images, ANOVA's $p-value = 0.005$, which indicates all means are not equal. Indeed, as we can see in Tukey's result, the means using 192 and 256-bits keys are not statistically equal as the range does not include zero. Finally, Table 3 summarizes the mean proportion between encryption and total execution times so we can analyze the impact of adding an encryption module into computation offloading frameworks. The table shows that the encryption time represents 2.17% to 5.35% of the total execution time depending on the image resolution, key size, and where to offload. The proportion ranges from 3.49% to 5.35% when H1 offloads to the cloudlet and 2.17% to 3.65% when H1 offloads to another handset. As we can see, the longer the total execution time, the lower the impact of encryption. This is the reason the total encryption time has less impact when offloading to another mobile device, despite the encryption time be higher in such a case, as we show in Table 1.

5 CONCLUSION

Security and privacy are essential aspects that any offloading solution should support, but few current ones provide. In this paper, we address this research theme by including encryption support to an existing of-

floading platform called CAOS/CAOS D2D. Our experiments show that encryption support may increase the total offloading time up to 5.35% (±0.64%). But they also show that the larger the offloading time, the less the impact of encryption procedures. The encryption key size seems irrelevant compared with the influence of downloading and uploading times, so bigger keys should be used instead of weaker ones. Regarding energy consumption, our experiments show that the encryption process consumes up to 878 mJ, and the longer the encryption key and the bigger the data to be transferred, the higher the energy consumption. As future work, we intend to expand the experimentation and test a set of other mobile devices leveraging different wireless technologies (e.g., 4G, 5G), besides using public cloud instances as remote execution environments and test other cryptographic algorithms.

ACKNOWLEDGMENTS

The authors would like to thank The Ceará State Foundation for the Support of Scientific and Technological Development (FUNCAP) for the financial support (grant number 6945087/2019).

REFERENCES

Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. (2018). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465.

Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R., and Simon, S. (2018). Recommendation for pair-wise key-establishment using integer factorization cryptography. Technical report, National Institute of Standards and Technology.

Bhatia, T. and Verma, A. (2017). Data security in mobile cloud computing paradigm: a survey, taxonomy and open research issues. *The Journal of Supercomputing*, 73(6):2558–2631.

Chen, M. and Hao, Y. (2018). Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597.

Costa, P. B., Rego, P. A. L., Rocha, L. S., Trinta, F. A. M., and de Souza, J. N. (2015). Mpos: A multiplatform offloading system. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 577–584, New York, NY, USA. ACM.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2011). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*.

Diro, A. A., Chilamkurti, N., and Nam, Y. (2018). Analysis of lightweight encryption scheme for fog-to-things communication. *IEEE Access*, 6:26820–26830.

Donald, A. C. and Arockiam, L. (2015). A secure authentication scheme for mobicloud. In *2015 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6.

Dos Santos, G. B., Trinta, F. A., Rego, P. A., Silva, F. A., and De Souza, J. N. (2018). Performance and energy consumption evaluation of computation offloading using caos d2d. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106.

Gomes, F., Viana, W., Rocha, L., and Trinta, F. (2017a). On the evaluation of a contextual sensitive data offloading service: the cop case. *Journal of Information and Data Management*, 8(3):197.

Gomes, F. A., Rego, P. A., Rocha, L., de Souza, J. N., and Trinta, F. (2017b). CAOS: A context acquisition and offloading system. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, pages 957–966. IEEE.

Gupta, B. B., Yamaguchi, S., and Agrawal, D. P. (2018). Advances in security and privacy of multimedia big data in mobile and cloud computing. *Multimedia Tools and Applications*, 77(7):9203–9208.

Mollah, M. B., Azad, M. A. K., and Vasilakos, A. (2017). Security and privacy challenges in mobile cloud computing: Survey and way ahead. *Journal of Network and Computer Applications*, 84:38–54.

Padhi, S., Tiwary, M., Priyadarshini, R., Panigrahi, C. R., and Misra, R. (2016). Secomn: Improved security approach for opportunistic mobile networks using cyber foraging. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pages 415–421.

Rego, P. A., Trinta, F. A., Hasan, M. Z., and de Souza, J. N. (2019). Enhancing offloading systems with smart decisions, adaptive monitoring, and mobility support. *Wireless Communications and Mobile Computing*.

Rego, P. A. L., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and de Souza, J. N. (2016). Performing computation offloading on multiple platforms. *Computer Communications*.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2014). Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *Communications Surveys Tutorials, IEEE*, 16(1):369–392.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23.

Silva, B., Sabino, A., Junior, W., Oliveira, E., Júnior, F., and Dias, K. (2017). Performance evaluation of cryptography on middleware-based computational offloading. In *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 205–210.

Varghese, B. and Buyya, R. (2018). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861.

Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S. (2012). Refactoring android java code for on-demand computation offloading. *SIGPLAN Not.*, 47(10):233–248.