# Characterizing the Performance of Security Functions in Mobile Computing Systems

Abdulmonem M. Rashwan, *Member, IEEE*, Abd-Elhamid M. Taha, *Senior Member, IEEE*, and Hossam S. Hassanein, *Senior Member, IEEE*

***Abstract*—The next-generation mobile networks will be equipped with sophisticated communication security, ensuring the safety and authenticity of the transmitted information. However, many of today's prominent security measures are cryptography-based and present several operational challenges in mobile computing systems. Thus, enforcing security measures can greatly affect communication performance, particularly, when it comes to time-based guarantees such as delay and jitter. Moreover, mobile computing systems have limited energy sources, which can be depleted quickly by improperly enforcing such resource-intensive operations. Therefore, it becomes vital to understand the computational characteristics of security measures from a communication perspective. By observing these characteristics, it may be possible for existing and future mobile systems to be suited with security functions that provide the sufficient communication security while maintaining both the power-efficiency and the delay/jitter requirements. In this paper, we propose a benchmarking environment for evaluating cryptography-based security functions from a communication perspective. The paper investigates how mobile systems' design and operation characteristics have a significant impact on the computational characteristics of security functions. The paper explores the evaluation metrics that can be used in benchmarking security functions within various communication settings and proposes the use of a simple and effective delay-based metric for the benchmarking process. The computational characteristics of some selected security functions are evaluated under the proposed benchmarking environment and presented in this paper. While the main focus of the work is the widely utilized mobile communication settings, the proposed evaluation scheme can be applied for other communication settings and for noncryptographic security functions.**

***Index Terms*—Benchmark testing, communication system security, message authentication, mobile security, next-generation Internet, resource management.**

## I. Introduction

INFORMATION security represents an integral element for both future computing applications and next generation networks. Without applying proper security measures, future computing services and networks will not sustain the growing number and variety of security attacks, risking potentially catastrophic consequences for governments, businesses, and the people depending on these services. This threat motivated decades of designing and proposing various functions, protocols, and frameworks, targeting a wide range of security aspects at different levels within the computing industry. It also yielded security architectures and systems such as IPSec, SSL/TLS, Kerberos, TCP-AO, SCTP-AUTH, to name a few [1].

Given their strength advantage, several cryptographic functions have found their way as software-based and hardware-based solutions to provide information encryption and verification services. The use of cryptography, however, is computationally demanding in terms of both resources and processing time, which is why early Internet developers avoided adopting such measures in various existing real-time and/or high-speed services, depending only on noncryptographic and less-secure approaches such as tough passwords and questions. These approaches, however, are unfit for changing landscape of the Internet, including the rapid development of wireless communications and the increasing popularity of smart mobile systems, and the introduction of new networking paradigms such as peer-to-peer (P2P), Internet-of-Things (IoT), and information-centric-networks (ICNs).

Recently, several performance and cryptanalysis studies for existing and new cryptography-based security functions have focused on evaluating the strength of cryptography-based functions and their absolute physical resource demands. However, the evaluation of the absolute resource demands may not be beneficial for effectively handling communication quality-of-experience (QoE), since a resource demand does not always translate into a required information processing latency in a communication session. Moreover, today's mobile computing architectures make it more challenging to find a deterministic relation between the resource demand and the processing time. Only slight considerations were made for how the functions behave under the constantly varying resource capabilities of a mobile computing system. Without considering the dynamic nature of mobile systems for the evaluation, an accurate estimation cannot be properly obtained for both latency and jitter in the established communication sessions. Such voids in understanding can lead to undesirable communication and system performance issues, especially in next-generation networks such as IoT and ICN, where many of the communicating systems are mobile with controlled/limited resource capabilities.

Manuscript received August 08, 2014; accepted September 11, 2014. Date of publication September 25, 2014; date of current version October 21, 2014. This work was supported in part by the Qatar National Research Fund (a member of The Qatar Foundation) under NPRP Grant NPRP4-553-2-210 and in part by the National Sciences and Engineering Research Council (NSERC) of Canada.

A. M. Rashwan and H. S. Hassanein are with the Telecommunications Research Laboratory, School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada (e-mail: arashwan@cs.queensu.ca; hossam@cs.queensu.ca).

A.-E. M. Taha is with the Department of Electrical Engineering, Alfaisal University, Riyadh 11533, Saudi Arabia (e-mail: ataha@alfaisal.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JIOT.2014.2360217

This work introduces a comprehensive benchmarking environment to evaluate cryptography-based security functions from a communication perspective. In this paper, we propose a benchmarking procedure for evaluating cryptography-based security functions in communication environments. In doing so, we suggest the use of a delay-based metric called apparent processing to facilitate a meaningful evaluation for communication purposes as seen by the initiating mobile systems. The metric borrows from the notion of apparent parallelism utilized in the context of parallel computing. The evaluation presented in this paper is focused on the important types of cryptography-based security functions that generate *message authentication code* (MAC); a security measure that provides information integrity protection service for communication sessions.

This paper is organized as follows. Section II surveys related work and evaluation/benchmarking architecture and illustrates the need for benchmarking security functions in mobile computing systems from a communication perspective. Section III describes how a cryptography-based security function is evaluated in terms of the resource demands and offers guidelines on how to properly evaluate it for communications. Section IV details the benchmarking environment used to evaluate selected MAC security functions. A benchmarking analysis for the evaluated security functions results is presented in Section V. Section VI lists observed considerations and challenges faced by the fellow researchers who may be interested in replicating the benchmarking analysis. Finally, conclusion and future directions are noted in Section VII.

## II. Related Work and Motivation

Cryptography-based security functions include stream/block ciphers, public key cryptography, cryptographic hashing functions, and MAC functions. Security measures offered by these functions mainly include information encryption/decryption, validation, and signing. Conducting performance studies for cryptography-based security functions has been the focus of several works in the literature, such as evaluating different implementations directly and indirectly for various applications [2], [3], architectures [4]–[7], and network structures [8], [9]. Those studies usually aim to illustrate the performance superiority of certain security functions or the shortcomings of some other security functions. If we exclude cryptanalysis-related studies, an evaluation of security function can mainly analyze one or more of the following metrics.

1) *Processing power:* Concerns the absolute processor cycles or number of execution steps required by the evaluated security functions [4], [6], [10]. Evaluation of the processing power is essential to ensure that the evaluated functions operate in a timely manner under several computing systems and to observe the resistance of the evaluated functions against certain availability attacks, such as denial-of-service.
2) *Demanded memory space:* Concerns the memory size, measured in bits or bytes, required statically and/or at run time by the evaluated security functions [8], [10], [11]. The evaluation of memory space demands is mainly used for investigating the suitability of implementing security functions in space-constrained computing systems, such as sensors and smart cards, where the memory space is limited to few kilobytes.
3) *Throughput:* Concerns the output rate, in terms of bits or bytes, of the evaluated security functions [5], [10], [12], [13]. This study usually is linked to the processing power and represents the suitability of integrating security functions in high-speed networks and services. Usually, these types of networks and services, such as terrestrial networks, do not require certain communication latency/jitter guarantees.

In addition to the aforementioned efforts, several benchmarking suites for evaluating cryptographic performance have been introduced. The main aim of these suites is to evaluate how fast a cryptography-based function computes under diverse conditions. Examples include NPCryptBench [14], eBACS [15], and Crypto++ [8]. These suites also feature benchmarking for different message sizes (as with eBACS), for operating under network processors (as with NPCryptBench), and for multiprocessing performance (as with Crypto++).

Most of the performance studies and benchmarking suites presented above are centered on evaluating the security functions under different computing architectures. Such evaluations do not capture the dynamic nature of mobile computing, which is extremely dependent on the active computing context, not on the computing environment context where they are evaluated at or on the application/service where they are evaluated for. While it is essential to analyze a security function to understand its nominal demands and characteristics, such an analysis may not be beneficial in practice as the characteristics and operation behaviors of today's computing systems and applications are complex and diverse. For example, today's mobile systems are powered by various single-core and multiple-core processors whose operations vary on the run [16]. Meanwhile, many mobile systems are equipped with additional hardware-assisted components to offload computing-intensive operations such as graphics and cryptography from their main processors, while the base applications and services are still maintained by the main processors [17]. The existing benchmarking approaches are thus limited in understanding the operational impact of security functions on a communication session, especially since the benchmarking is handled outside the core or processor where the communication session is handled.

To further elaborate, different considerations including the following must be taken into account when evaluating any cryptographic security function from a communication perspective.

1) *Context:* The resource capability of a computing system can be a reflection to the surrounding context. For example, factors such as the operating temperature, power source, local time, demands of the calling applications, and activities of other communicating entities on the local network can greatly affect the processing timeline of the operating security functions, regardless of the actual resources required. Thus, in communications, it is more informative to be aware of how much time is needed to

apply communication security measures instead of the amount of resources required. Such awareness can be helpful for a more effective quality-of-service (QoS) management in communications guarantees, which is mainly correlated to the transmission latency, not to the absolute resource utilization.

2) *Resources involved:* Next-generation Internet protocols and their related operating systems will have several security measures as a standard feature. The execution of the supported security functions can be locked on certain processor cores and/or memory space, so those functions do not render the system unusable in case of high load. Previously conducted performance studies commonly focus on the processor power consumed by an evaluated function without taking into consideration how many cores are involved by executing that function.
3) *Extra-processor computation:* There are implementations of security functions that execute outside the main processor, e.g., an external cryptographic processor. Yet, in such instances, the main processor is still responsible for handling the communication session utilizing the security function. Evaluating the processing power on either the main processor or the external processor alone may not sufficiently indicate the effective performance analysis of that communication session.
4) *Implementation heterogeneity:* The resource capability of a system can refer to the implementation of the security functions on the communicating systems. Some systems may have hardware circuitry or hardware-assisted instructions for the cryptographic encryption or hashing engines. Some systems use security functions that are designed to run on multiple simultaneous processors, while other systems use functions that are designed to run on single processors. Even systems that have the same physical capabilities can have different software implementations for the same security function. Such implementation dependency leads to diverse performance trends between communicating systems. This diversity can cause performance or availability issues for limited-resource systems such as sensors.
5) *Resource management:* Most mobile systems today are equipped with some sort of resource management technique, aimed mainly at providing an energy-efficient operation. In such systems, the useful resource capabilities are subject to the mobile system's resource management, which normally operates dynamically based on the processing demands. As with the implementation heterogeneity, communications between mobile systems with dynamic resource management techniques can have diverse performance trends, including the nondeterministic performance expectations of security functions operating on those communications. In addition, expectations on how mobile system resources are controlled do not necessarily reflect on how security functions will behave. For example, doubling the processor clock speed on a mobile system does not mean that the operating MAC functions will take exactly half the processing time, as they are subject to other architecture factors such as the design of the processor execution pipeline and the I/O bus' speed.
6) *Workload concurrency:* Previous studies did not consider the use of concurrent workloads (e.g., to simulate multiple connections) for their evaluations. With concurrent workloads, effects of scheduling, memory and I/O demands can be reflected on the processing time, giving more realistic performance determination from a communication perspective, such as computing the system's capacity with respect to the number of concurrent connections.

Therefore, it becomes obvious that there are substantial considerations to be made beyond processing power, memory demands, throughput, or even the strength of the evaluated security functions themselves. It is clear that a security function may not be implemented or executed across different mobile computing architectures, so that many existing security protocols with predetermined underlying security functions cannot suit every existing and future mobile system. Moreover, existing protocols usually rely on having a predetermined security function that operates statically during a connection lifetime, which makes them vulnerable for easier attacks as opposed to if they have a methodology to frequently change the operated security function during that connection lifetime. Future security protocols may thus need to implement some sort of adaptive functionality that securely serve current and future communicating applications regardless of their requirements or the capabilities of their computing environments.

The objective of the work presented here is to realize an environment that accurately and effectively characterizes the effect of aforementioned considerations on the computational behavior of the security functions operating in communication environments. This work extends efforts made in [18] and [19] and emphasizes the significance of evaluating security functions from the communication perspective and beyond their actual demands and requirements.

## III. Understanding the Performance Evaluation of a Security Function

Basic functionalities for the cryptography-based security functions existing today can be categorized into one of the following categories.

1) *Keying*, where security functions assign a "secret" phrase to be used in cryptographic operations.
2) *Encryption*, where security functions use a cryptographic methodology, in conjunction with the "secret" phrase, to alter information such that it cannot be restored without having the "secret" phrase.
3) *Decryption*, where security functions use a cryptographic methodology, in conjunction with the "secret" phrase, to restore previously encrypted information.
4) *Hashing*, where security functions use a cryptographic computation to generate a fixed and unique checksum tag (or hash) for given information.

Other well-known functions and frameworks, such as public key infrastructure, digital information signing, and message

TABLE I
METRICS FOR EVALUATING BASIC SECURITY FUNCTIONS

| Function | Metric | Evaluation unit |
|---|---|---|
| Keying | Processing power | Processor's clock cycles (cycles) |
| Encryption | Processing power | Processor's clock cycles per byte(cycles/byte) |
| Decryption | Processing power | Processor's clock cycles per byte(cycles/byte) |
| Hashing | Processing power | Processor's clock cycles per byte (cycles/byte) |
| All | Demanded memory space | Byte |
| All (except keying) | Throughput | Bits per second (bps) |

authentication, utilize one or more of the aforementioned basic security functions.

As with other computing elements, security functions are evaluated in terms of their time and space complexities. However, security functions differ in that their key objective is to resist breakage attacks and process information in a timely manner and are thus further evaluated in terms of cryptanalysis and information throughput.

The following elaborates on how the performance of security functions is commonly evaluated in the literature.

## A. Current Performance Evaluation Approaches

Performance studies in the literature evaluate security functions in one of the two major approaches: 1) analytical and 2) empirical. The analytical approach involves the use of theory and mathematical proofs to determine the time and space complexities of the analyzed function. Such approach also focuses solely on the performance of the analyzed function and does not consider the effects of the computing mobile architecture and the operating context, e.g., impact of scheduling preemption or the processing power optimization.

The empirical approach involves the use of measurement probes to understand the performance of a function at runtime and yields a more realistic performance view. The empirical approach is, therefore, more appealing to application developers as it helps in the proper design of security-critical applications. However, conducting a performance study under this approach is usually subject to the implementation of the analyzed function and the hosting system. The *scope* of an empirical performance study is, therefore, typically subjected to a limited range of computing systems that share similar architectural characteristics.

Regardless of which approach is used, security functions are analyzed in the literature in terms of the processing power, the demanded memory space, or the throughput. Table I summarizes the aforementioned metrics with their most commonly used units.

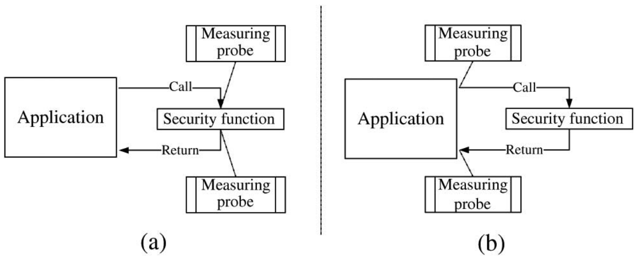


Fig. 1. Placement of the evaluation probes (a) within the evaluated security function and (b) within the calling application.

## B. Evaluation From a Communication Perspective

When it comes to production–computing environments, security functions do not operate in isolation and are always executed as parts of other computing applications or services. This is the case with communication protocols and services. Taking into account the aforementioned considerations, the following guidelines should be followed when implementing an evaluation setup for testing security functions in communications.

1) *Empirical evaluation:* When we consider real-time and delay-critical applications calling auxiliary functions, such applications are highly dependent on how long a called function takes rather than how much resources that function requires [20]. In addition, the existing diversity of computing architectures and software implementations poses complex challenges for researchers who use analytical approaches for evaluating an application's performance.
2) *Understanding communication session:* Modern computing entities establish multiple communication sessions between each other. Within the established sessions, information is transmitted across various network interfaces in the form of packets. Thus, if a security function is employed into a communication session, it will typically operate on the transmitted packets. Thus, the evaluation of a security function should be conducted in a way that reflects aforementioned communication characteristics. For instance, an involved security function should be evaluated under the existence of concurrent communicating sessions that operate on various packet sizes.
3) *Placement of the evaluation probes:* Measurement probes are usually placed into the implementation of the evaluated functions, as illustrated in Fig. 1(a). Such placement does not take into consideration the overhead caused by the switching from and to the calling application. This in turn distorts the performance and availability measurements sought. From a communication perspective, a better placement is within the communication session process, as shown in Fig. 1(b).
4) *Evaluation metrics:* With today's sophisticated mobile computing hardware and software, it is extremely challenging to observe the absolute demands of an evaluated function, especially if such function and its calling application are handled by independent software and/or

hardware components. For instance, a communication session calling a security function handled by an external cryptoprocessor will not be dependent on the function's absolute with respect to the main processor. In such a scenario, it is more practical to evaluate the security function's demands as they appear to the calling application.

## IV. Benchmarking Environment for Evaluating MAC Functions in Mobile Systems

In this section, we introduce our benchmarking environment for evaluating MAC functions in mobile systems from a communications perspective based on the guidelines detailed above. It should be noted, however, that the guidelines can be applied in benchmarking any cryptography-based security function and in any computing system.

A MAC function has three key operations. The first operation is known as *keying*, where a secret key is assigned to the MAC function to be used in generating secure MACs. The second operation is known as *tagging*, where MACs are generated and attached to the corresponding messages. The third operation is known as *validating*, where a message is validated against a MAC attached to it.

MAC functions are powered by two types of cryptographic functions: 1) hash or 2) block cipher [21]. Hash functions, such as MD5, SHA1, and SHA2, are one-way compression algorithms that map variable-length large messages into short fixed-sized strings that are unique to their corresponding messages. Block cipher functions, such as AES, TWOFISH, SERPENT, and RC6, are encryption/decryption algorithms designed to work on fixed-sized portions of given messages called blocks. MACs are generated either by directly hashing combined messages with provided secrets using hash functions or by hashing message blocks encrypted with provided secrets when using block cipher functions.

In order to successfully evaluate the MAC functions for communication purposes within mobile systems, it is important to have a benchmarking setup that effectively describes the true nature of both mobile environments and communications. To achieve this, the benchmarking setup should incorporate the use of mobile production systems and production operating systems configured for everyday usage. For communications, the setup should emulate concurrent processing of message authentication and utilize realistic message lengths used in major mobile protocols.

In the benchmarking setup presented in this work, real architectures from three major well-known brands in mobile computing were evaluated. A customizable distribution of Linux operating system is used, in which kernel is powering many of today's mobile systems. The workload is made to apply message authentication processing in a simulated communication mode while controlling some operational factors such as cores involved, number of concurrent sessions, and session duration. The evaluation workflow is also designed to ensure that the both workloads and their parameters are appropriate for the studied systems. Finally, the evaluation metrics are selected to observe computational characteristics of the studied systems.

The following elaborates on the details of the benchmarking setup.

### A. Environment

The environment is designed to ensure the minimum influence from the operating system on benchmarking. All experiments in this study were conducted under the Linux operating system environment (Ubuntu 12.04 LTS). The chosen operating system running in Gnome desktop mode uses the "Completely Fair" scheduler for scheduling its processes.

A well-known cryptographic library Crypto++ (v. 5.6.2) is used for this study as the provider for the evaluated MAC functions. This selection is motivated by the library's popularity among academia for studying cryptographic performance and cryptanalysis [8]. It is also open-source and has cross-platform compatibility, making it suitable to run under various operating systems and computer architectures. The library further has both hardware-assisted and software-only implementations for some functions such as AES (using x86 AES-NI extension) and SHA-256/512 (using x86 SSE-2 extension), making it a good option to test the effect of different implementations using the same hardware. More significantly, the library has its own benchmarking tool that can be used as a validation tool for our benchmarking results.

Evaluation results were obtained for MAC functions running under the following architectures that represent most of the existing mobile systems in the market.

1) *x86-Based (32-bit) Laptops and Tablets:* Intel Core I3 M350 (32-bit mode; Dual-core with SMT), Intel Core I5 650 (32-bit mode; Dual-core with SMT), Intel Pentium 4 M 3.0 GHz (Single-core with SMT), Intel Atom D525 (Dual-core with SMT), and AMD Opteron 2354 (32-bit mode; Quad-core).
2) *x86-Based (64-bit) Laptops and Tablets:* Intel Core I5 650 (64-bit mode; Dual-core with SMT).
3) *ARM-Based (32-bit) Smart Phones and Tablets:* Texas Instruments' DM3730 ARM Cortex A8 (Single-core) and Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (Dual-core).

### B. Workload and Workflow

An in-house multithreaded benchmarking application is written to evaluate selected MAC functions via controlled dynamic mobile communication environments. The application uses multithreading to create workload instances in order to minimize the effect of memory switching on the measurement accuracy. The application was also equipped with a method for binding the execution of the workload instances into certain predefined cores in order to prevent process migration. Moreover, a resource–control module was integrated into the benchmarking application in order to control the following resource management features found in mobile systems.

1) *Frequency scaling:* Reduces or increases processor clock speed based on the applications' demands.
2) *Simultaneous hardware multithreading* (SMT)*:* Improves processor utilization (and reduces wasted energy from

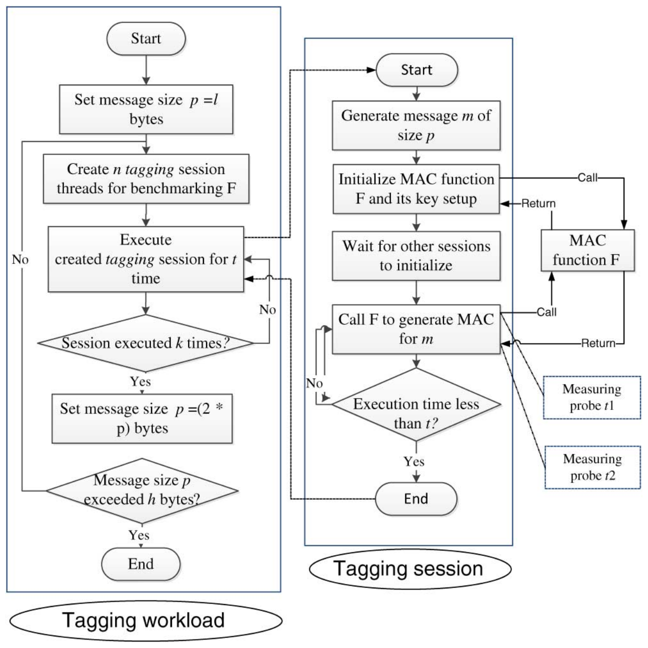

Start
Set message size p = l bytes
Create n tagging session threads for benchmarking F
Execute created tagging session for t time
No
Session executed k times?
No
Yes
Set message size p =(2 * p) bytes
Message size p exceeded h bytes?
Yes
End
Tagging workload
Start
Generate message m of size p
Initialize MAC function F and its key setup
Wait for other sessions to initialize
Call F to generate MAC for m
Execution time less than t?
No
Yes
End
Call
Return
MAC function F
Call
Return
Measuring probe t1
Measuring probe t2
Tagging session

Fig. 2. Benchmarking workload procedure for tagging operation.

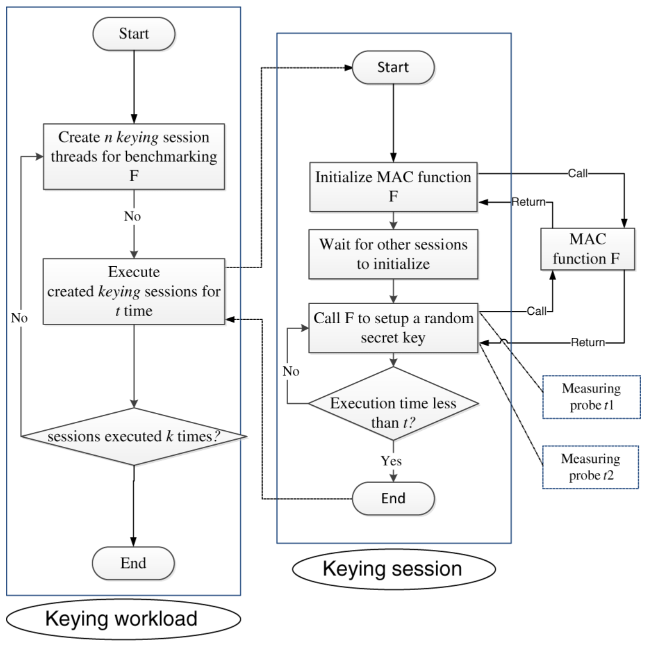

Start
Create n keying session threads for benchmarking F
No
Execute created keying sessions for t time
No
sessions executed k times?
End
Keying workload
Start
Initialize MAC function F
Wait for other sessions to initialize
Call F to setup a random secret key
Execution time less than t?
No
Yes
End
Call
Return
MAC function F
Call
Return
Measuring probe t1
Measuring probe t2
Keying session

Fig. 3. Benchmarking workload procedure for keying operation.

underutilization) through executing multiple hardware threads per core. With SMT enabled, it is expected that processor temperature will be higher as the processor will do more work in this case.

3) *Processor/core parking:* Parks (aka shutdowns or idles) some of the active processor/cores in case of low demands. Modern mobile systems also utilize this feature in order to reduce operating temperature.

The benchmarking workload procedures for tagging and keying are, respectively, illustrated in Figs. 2 and 3. Since typical communication sessions usually run concurrently with different durations and message lengths, the workload

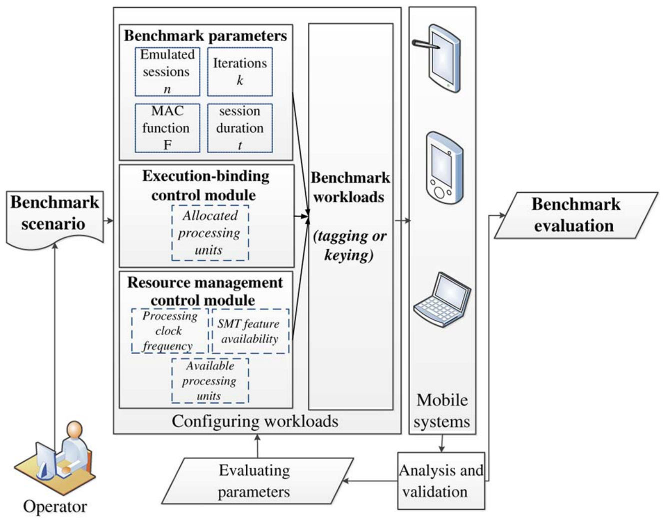


Fig. 4. Benchmarking workflow procedure.

is implemented to simulate such conditions while assuming a fixed selection of evaluated function and fixed session duration per run to simplify the evaluation. In addition, the workload executes the evaluated function as if it is in a typical production session. Under each of studied architectures, the workload is applied with no other foreground applications running except for the Gnome desktop environment in order to reduce the effect on the measurement accuracy.

The workload creates $n$ session threads to emulate the existence of $n$ communication sessions transmitting messages ranging from $l$ bytes to $h$ bytes. For each thread, there are $n-1$ background threads representing $n-1$ concurrent communication sessions. Within each session, measurements are taken using both the processor clock and the system's wall clock. The period of taking measurements is defined by interval from $t_1$ to $t_2$, where $t_1$ and $t_2$ are predefined time values between 0 and the workload session duration $t$, with $t_2 > t_1$.

Fig. 4 describes the benchmarking workflow procedure. This procedure is essential to ensure workloads being applied to suit benchmarked systems, and measurements obtained are accurate and informative. First, a benchmarking scenario is defined to include the number of connections, the function to evaluate, and the available/allocated resources. Through the process, obtained measurements are analyzed and validated. For example, the accuracy can be affected by resources overutilization, insufficient session duration for reaching the steady state, inappropriate measurement period, or insufficient collected result samples. In this case, their corresponding parameters, which are the number of connections $n$, the session duration $t$, the measurement period $[t_1, t_2]$, and the number of iterations $k$, will be adjusted accordingly for the next workload.

## C. Metrics

Cryptography-based security functions are known to be computationally demanding, but many of them do not pose extensive memory and I/O demands. This makes the processing power the most reasonable evaluation metric for these

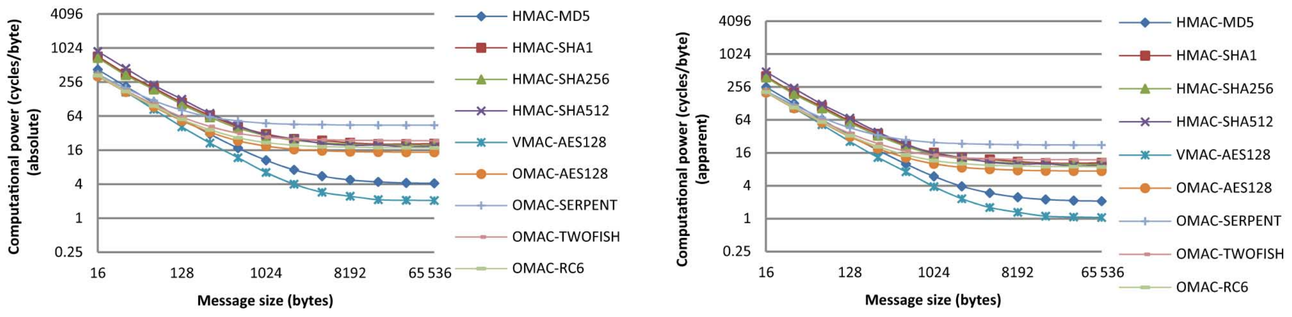


Fig. 5. Processing power of tagging operations under Intel Core I3 M350 (32-bit), using hardware-optimized Crypto++ Library.

functions. In this paper, two types of the processing power metrics are considered.

1) *Absolute processing power:* Defined as the actual number of cycles that a CPU uses to execute a function.
2) *Apparent processing power:* Defined as the estimated number of cycles that a CPU appears to use for executing a function, from the calling application perspective.

The above metrics are evaluated in two modes.

1) *Task mode*, where the metrics are evaluated for individual function calls.
2) *Average effective mode*, where the metrics are evaluated for all involved function calls during an evaluation session and averaged over the number of those function calls.

In this study, only "keying" and "tagging" operations are considered for the processing power evaluation, as the "validating" operates similarly to the "tagging" operation.

### *D. Evaluated MAC Functions*

*1) Hash-Based MAC (HMAC) [22]:* A hash-based algorithm used for message authentication in various popular Internet protocols (e.g., IPSec and TLS [1], [23]). Evaluated hash functions under this group are MD5, SHA1, SHA2-256 (simply SHA256), and SHA2-512 (simply SHA512).

*2) One-Key MAC One (OMAC1 or Simply OMAC):* Also known as CMAC; a block cipher-based algorithm that was introduced to resolve security flaw of its predecessor, CBC-MAC when generating MACs for variable-length messages [24]. In 2005, NIST recommended using OMAC for operating block cipher-based authentication [25]. Evaluated block cipher functions under this group are RIJNDAEL, TWOFISH, SERPENT, and RC6, all of which were the finalist candidates in advanced encryption standard (AES) selection process with Rijndael becoming the official AES [4].

*3) VMAC:* A block cipher-based algorithm designed to offer high-performance message authentication service [26]. Optimized for 64-bit computing architectures, VMAC utilizes block cipher functions via a "universal hashing" algorithm and secret key to generate MACs for a given message. Evaluated block cipher function under this group is AES, which is the only implementation available for VMAC at the time of study.

### *E. Measuring Assumptions*

The measuring overhead is taken into consideration when setting up the workload in order to minimize its effect on the measurement accuracy, especially for evaluating the processing power in average effective mode. Meanwhile, the processing memory utilization is assumed to be reflected on the processing power to some degree due to the increased I/O activity resulted from the process/thread transfer into memory for sleeping. Finally, voltage scaling, which saves power through reducing the operating core voltage and which is not manually adjustable, is assumed not to affect the processing power.

## V. Benchmark Analysis

Three scenarios are investigated in this benchmark study. The first observes the generic computational characteristics of the evaluated functions across various mobile computing architectures. The effect of the mobile resource management on the computational characteristics is the focus of the second scenario. The third scenario is concerned with observing the role of the process scheduling on the computational characteristics of the evaluated functions.

### *A. Observing the Generic Computational Characteristics*

In this benchmark scenario, the workloads run for $k = 10$ iterations with sessions running for $t = 20$ s. The measuring start ($t_1 = 5$ s) is set to give enough time for the sessions to reach steady-state stage. The measuring end ($t_1 = 15$ s) is set lower than $t$ to prevent the premature ending of sessions from affecting the measurement quality. The resource management control module is disabled since as the focus is on the generic computational trends of the evaluated functions. However, the execution-binding control module is used to bind the sessions to their allocated resources, so session migrations do not affect the measuring accuracy.

Figs. 5–8 show the processing power for the evaluated functions under different architectures and implementations, with 100 simultaneous sessions per workload and message lengths ranging from 16 bytes to 64 kbytes.

*1) General Computational Trend for Architectures Using the Same Function Implementation:* The computational trend, using the same implementation for Crypto++ library is mostly similar across evaluated architectures. However, there are non-deterministic differences in the performance despite having the same software implementation running across the evaluated architectures. This appears more clearly with the absolute demanded resources in each of the evaluated architectures.

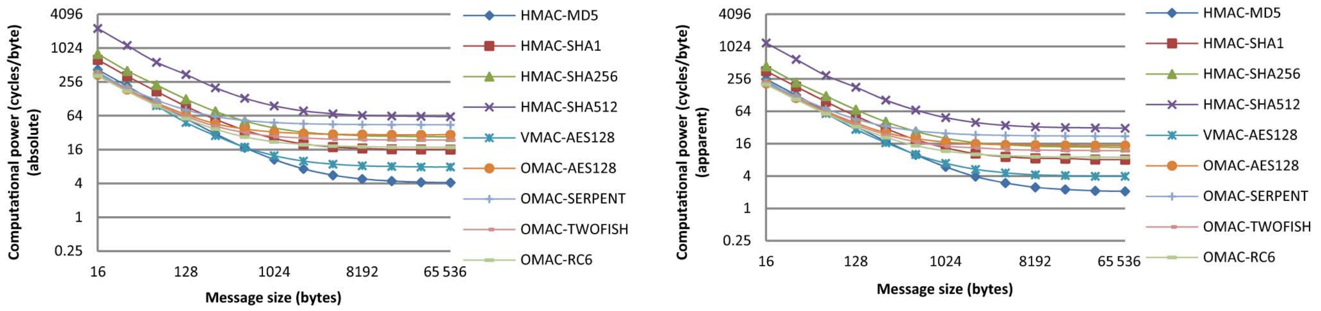


Fig. 6. Processing power of tagging operations under Intel Core I3 M350 (32-bit), using software-only Crypto++ Library.

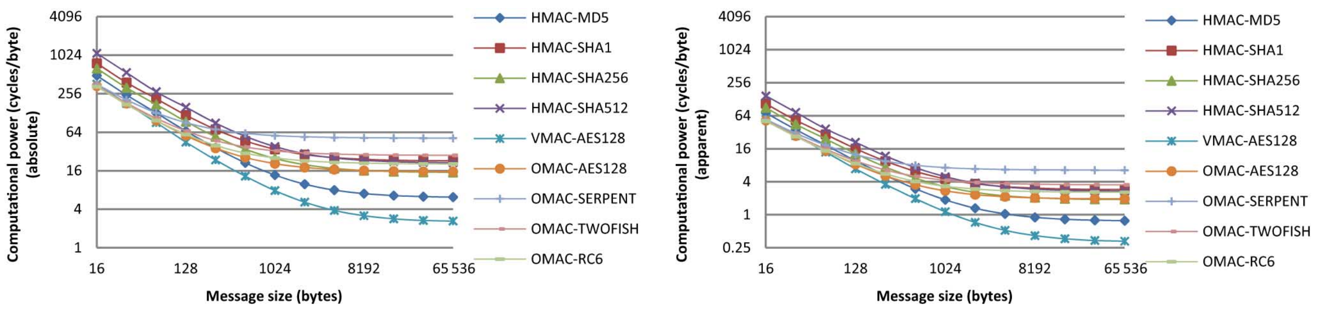


Fig. 7. Processing power of tagging operations under AMD Opteron 2354 (Dual Quad-core Processors—32-bit), using hardware-optimized Crypto++ Library.

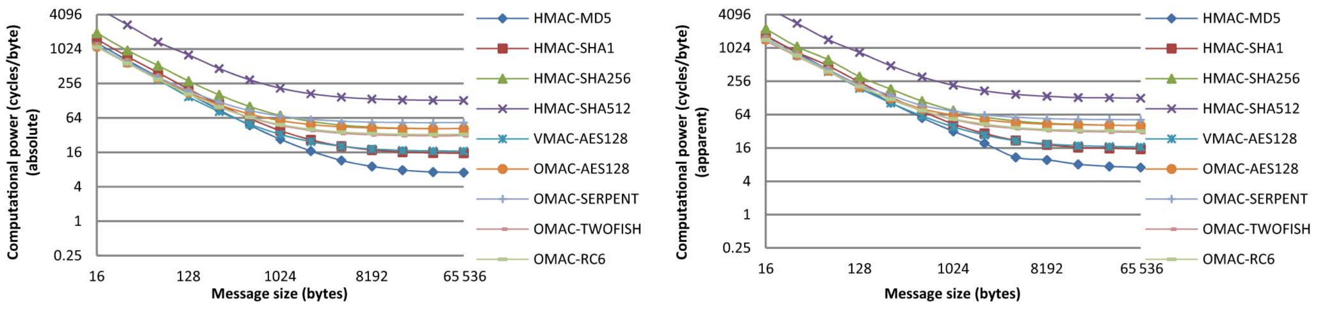


Fig. 8. Processing power of tagging operations under Texas Instruments' DM3730 ARM Cortex A8 (32-bit), using software-only Crypto++ Library.

Such nondeterministic behaviors are due to how different architectures are designed to handle execution of programs.

We remark from Figs. 5–8 that the apparent processing power, which reflects the processing latency as seen by the communication sessions, differs from the absolute one under the same evaluated architectures. In multicore computing environments, such as in Figs. 5–7, the processing latency, from the session perspective, appears to be better than the absolute demanded resources, and it is somehow related to the number of available processing cores. We note, however, that the apparent processing power is also subject to the contention effect with the background processes and I/O demands, as with Fig. 7, where there are two quad-core processors, but the apparent processing power appears to be up to six times better than the absolute one.

*2) Computational Trend Versus Message Size:* The computational trend changes with message sizes under different architectures. Systems and services can, therefore, optimize the selection of the best performing function based on the size of the transmitted messages, provided that other factors such as security strength and key setup time are also taken into consideration.

*3) Effect of a Function's Implementation on Computational Trend:* In the hardware-assisted implementation of Crypto++, as shown in Figs. 5 and 7, functions such as AES and SHA take advantage of speed boosting, while in Figs. 6 and 8, both AES and SHA lose their advantage. The computational trend has changed with HMAC-MD5 being the fastest processing 64-kbyte messages, and HMAC-SHA512 being the slowest (as opposed to VMAC-AES128 for fastest and OMAC-SERPENT for slowest in the hardware-assisted implementation). Such change in the trend is due to the fact that software-only implementations usually require higher I/O and memory operations than the hardware-optimized one. This is reflected mostly with the HMAC-SHA512, which operates on larger 1024-bit message blocks as opposed to the 512-bit message blocks in the other evaluated functions, and so requires higher I/O operations.

Similarly, we note that the implementations built for 64-bit architectures have different trends from the ones built for 32-bit architectures. In the 64-bit mode, HMAC-SHA512 benefits from the larger 64-bit register space, which reflected positively on its I/O operations and so its performance. Same thing is observed with AES-based MAC functions, such as

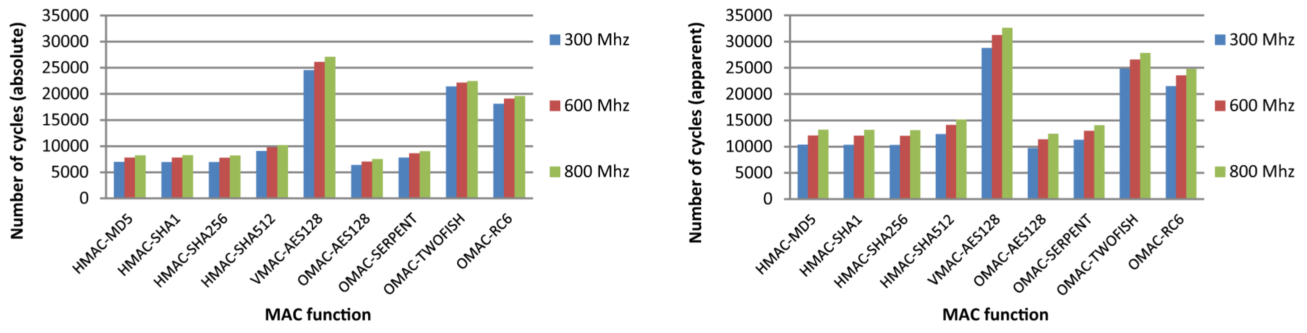


Fig. 9. Processing power of keying operations verses CPU frequency under Texas Instruments' DM3730 ARM Cortex A8 (32-bit).

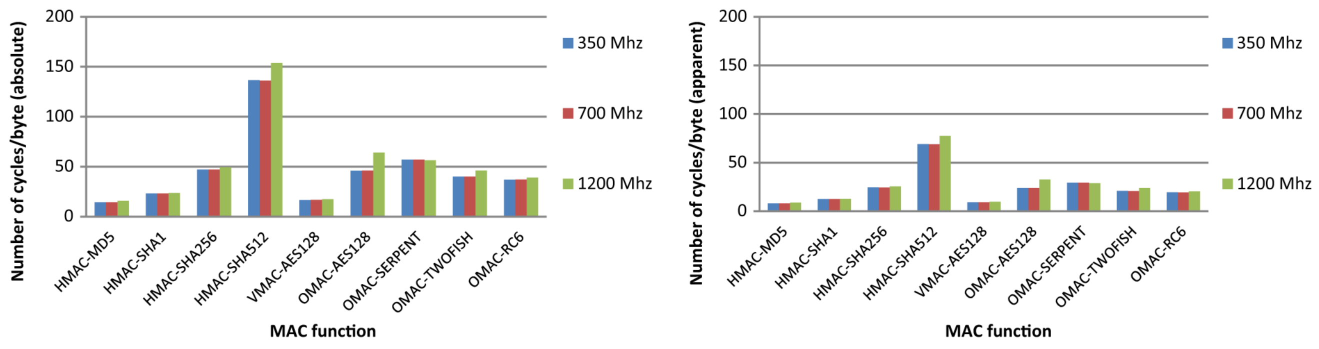


Fig. 10. Processing power of tagging operations (2048-byte messages) versus CPU frequency under Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (32-bit).

VMAC-AES128 and OMAC-AES128, since AES in 32-bit mode requires higher memory access [27].

## *B. Observing the Effect of the Mobiles' Dynamic Resource Management on the Computational Characteristics*

This benchmark scenario is constructed to provide a proper understanding of how the dynamic environment of mobile systems can affect the computational characteristics of the evaluated functions. In this scenario, we utilize the resource management control module to observe the environment dynamicity in a controlled matter. Other evaluation parameters are same as with the benchmark scenario in Section V-A, except that the message size is selected to be 2048 bytes, which represents the lowest value with order magnitude of 2 that is bigger than the Ethernet packet size of 1500 bytes. Moreover, we do not use the execution-binding module in order to observe the effect of session migrations on the measurements.

*1) Observing the Effect of Frequency Scaling:* The results for frequency scaling are shown in Figs. 9 and 10. Since frequencies are usually available in several steps, we only focus on three frequency levels for each processor: 1) the highest; 2) the middle; and 3) the lowest. It is intuitive to expect that the absolute processing power should remain the same regardless of the changes made in frequency scaling (The actual processing time, however, is naturally expected to change.). This is because absolute processing power is measured in cycles and not in seconds. Practically, this is not the case for all/most processors evaluated.

The only evaluated system that follows the aforementioned intuition is the Intel Core I5 650, which exhibits only slight irregularities in the performance of keying operations. These irregularities are mainly due to the resource contention with the OS and background processes, especially with the keying operations being of the light-processing type.

In architectures with considerably slow memory and I/O access compared to processor clock speed (especially with entry-level systems such as ARM architectures with high-speed CPU clocks), the increasing processor clock frequency leads to an increased number of wasted cycles. This appears noticeably with light-processing (Fig. 9) and/or I/O-intensive (Fig. 10) operations, such as with HMAC-SHA512 in 32-bit mode.

Generally speaking, applying the frequency scaling has resulted into nondeterministic behavior of the processing power for the benchmarked tagging/keying operations. Some tagging operation, such as in OMAC-SERPENT, had mostly showed some performance improvement with the increased clock speed across most of the benchmarked architectures, with no determination found between the operating frequency and processing power. HMAC-MD5 tagging operations, on the other hand, had shown slight performance degradation with the increase clock speed across the evaluated architectures.

*2) Observing the Effect of Core Parking and Enabling/Disabling SMT:* Fig. 11 shows the effect of core parking on the apparent processing power of selected architectures, with the percentage axis reflecting the additional cycles that the evaluated functions require after enforcing the parking.

We observe that enabling SMT generally improves the performance of tagging/keying operations; up to approximately 33% as shown in Fig. 11, since the resources of the SMT-enabled processors, by design, are utilized more effectively.

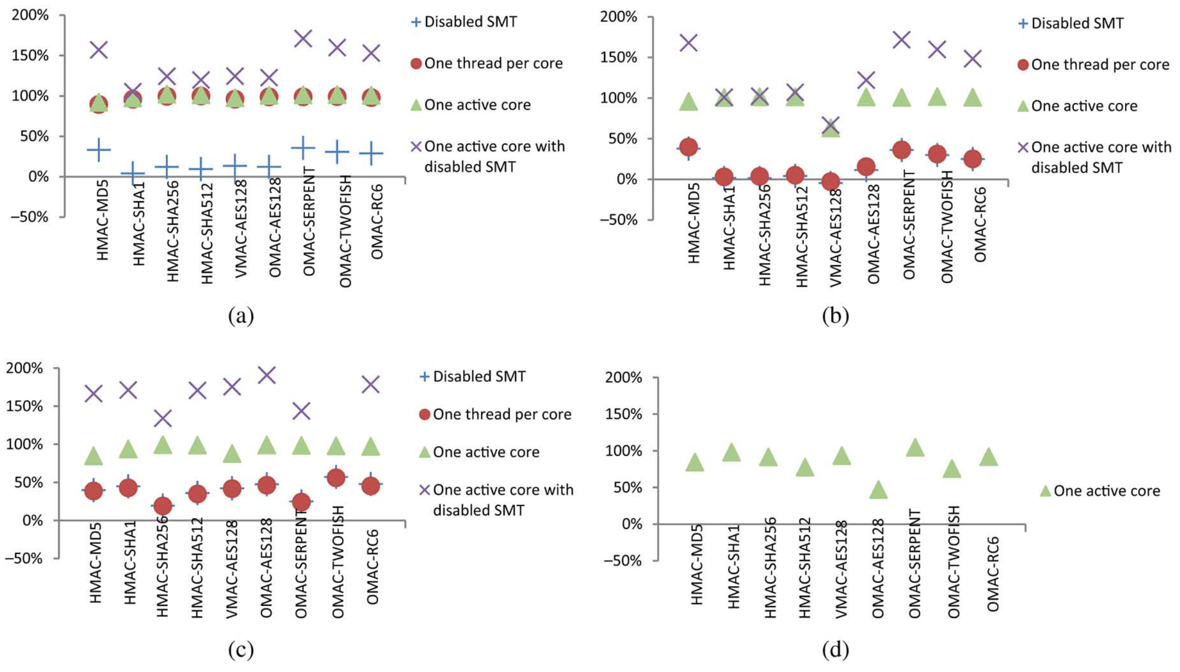


Fig. 11. Effect of parking and SMT availability on the performance degradation of the apparent tagging processing power (2048-byte messages) of selected architectures. Intel Core I5 650: (a) 32-bit, (b) 64-bit. (c) Intel Atom D525 (32-bit). (d) Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (32-bit).

However, some evaluated functions under certain systems, such VMAC-AES-128 under Intel Core I5 650 (64-bit mode), show performance degradation among all evaluated MAC functions. This can be due to the lack of sufficient instruction caching space, causing excessive cache misses that reflects on the SMT poor performance [28].

It is also noted that the trend of performance degradation caused by core parking is subject to the architecture of the evaluated system. For example, OMAC-AES128 had shown 1.5x performance degradation on an ARM-based architecture having only one active core, while the same function had shown 2x performance degradation on x86 architectures having only one active core.

## C. Observing the Effect of the Process Scheduling on the Computational Characteristics

In this scenario, we focus on how the process scheduling and the contention of background process can affect the computational characteristics of the evaluated functions. We conduct the evaluation in aforementioned modes: task, and average effective, where previous scenarios are conducted in task mode only. Since the focus is on the effect caused by resource contentions, the resource management module is taken out of the setup. All other parameters are the same as in the Section V-B setup. However, as the results have been normalized, measurement overhead was observed to rise by up to 25%.

*1) Observing the Effect of the Switching Overhead:* Figs. 12 and 13 show the processing power for some selected mobile architectures, evaluated in both "task" and "average effective" modes. It is observed that "Completely fair scheduler" is designed to watch solely for CPU demands. The scheduler works on allocating CPU resources for processes as a whole without considering the switching overhead due to the memory and I/O demands of the involved processes. Therefore, it is generally expected that the overall processing power for a communication session to be greater than the processing power of an individual tagging/keying operation. This appears clearly in the evaluation of light-weight operations such as keying (Fig. 12), where the allocated process memory plays a more significant role in affecting the processing power. Moreover, the process switching has high impact on the absolute CPU demands, indicating significant wasted processor cycles due to the process switching. Such impact is less visible with the apparent performance as it reflects memory and I/O demands for individual operations, which is the same reason why the apparent processing power usually appears higher than the absolute one under single-core computing architectures.

*2) Observing the Effect of Contention with Background Processes:* In order to observe how the processing power is affected due to the contention with the background processes, we focus on evaluating multicore architectures. We use the execution-binding module to bind the evaluated functions to a single processing core, leaving the background processes and the OS to execute freely on all cores. We observe that with the lightweight operations, such as keying and small message tagging [Fig. 14(a) and (b)], the processing power increases if all cores are used. This is because the demands for the slower memory and I/O accesses dominate the demand for the faster processing power in these operations. Therefore, the contention on all cores will drive the processes to use additional processing power to handle memory and I/O access bottlenecks. On the other hand, we note that there are cases where evaluation on all

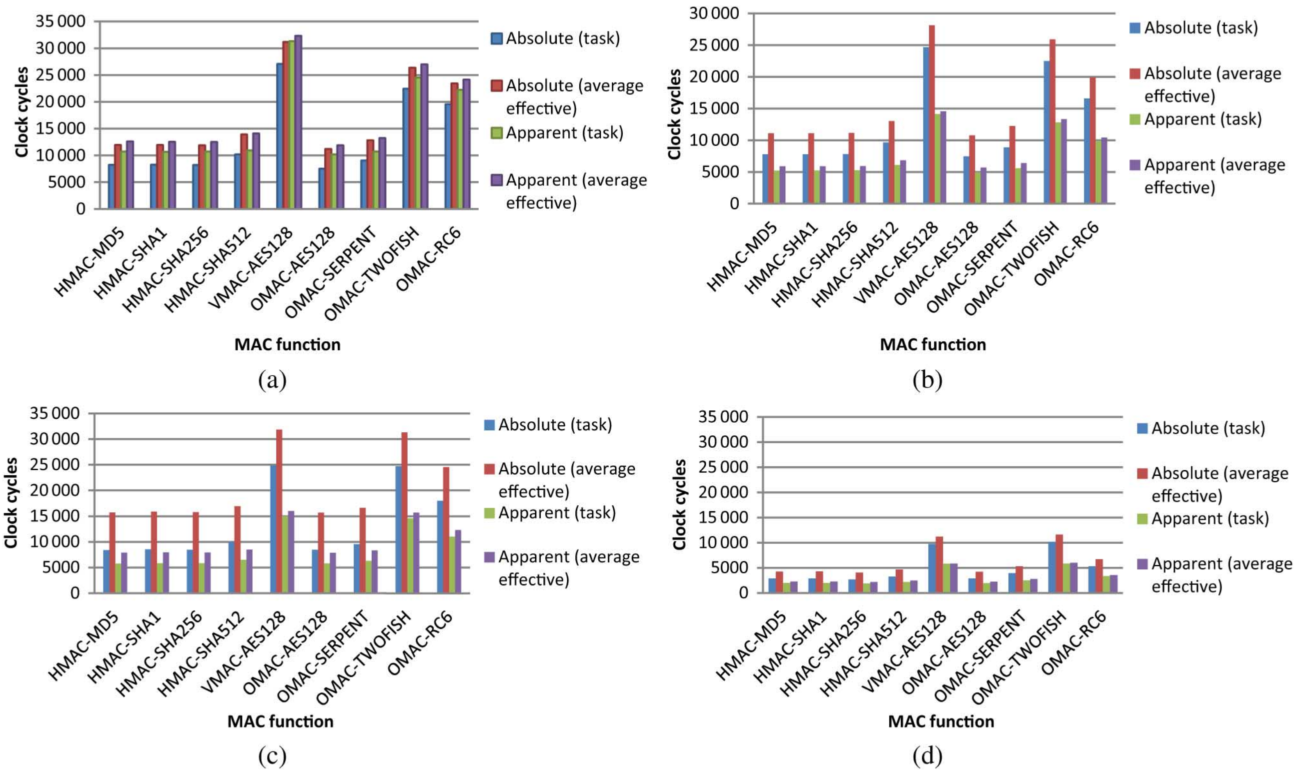


Fig. 12. Keying processing power of selected architectures in the two evaluation modes. (a) Texas Instruments' DM3730 ARM Cortex A8 (32-bit). (b) Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (32-bit). (c) Intel Atom D525 Dual-core (32-bit). (d) Intel Core I5 650 Dual-core (32-bit).

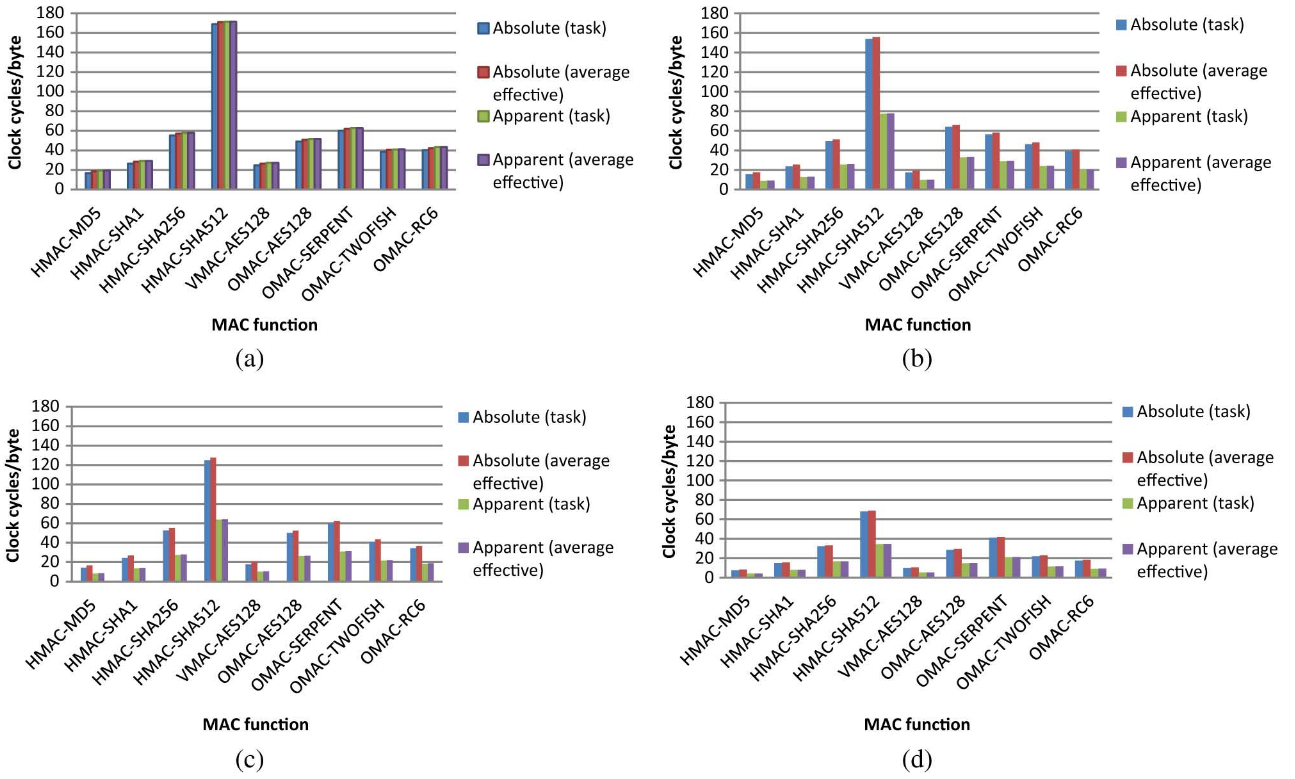


Fig. 13. Tagging processing power (2048-byte messages) of selected architectures in the two evaluation modes. (a) Texas Instruments' DM3730 ARM Cortex A8 (32-bit). (b) Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (32-bit). (c) Intel Atom D525 Dual-core (32-bit). (d) Intel Core I5 650 Dual-core (32-bit).

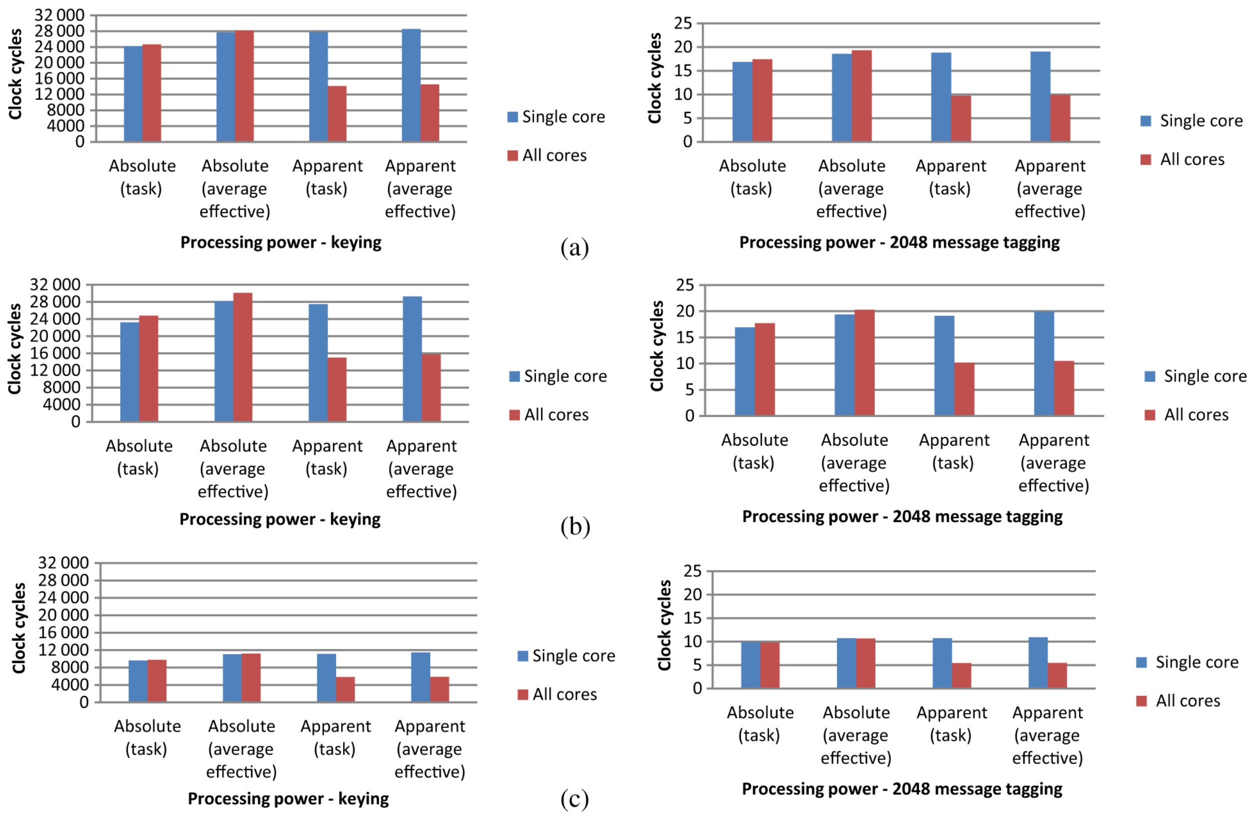


Fig. 14. Processing power for VMAC-AES128 under selected architectures in the two evaluation modes. (a) Texas Instruments' OMAP 4460 ARM Dual-core Cortex A9 (32-bit). (b) Intel Atom D525 Dual-core (32-bit). (c) Intel Core I5 650 Dual-core (32-bit).

cores improves the processing performance. Such cases include the domination of the processing power over memory demands [Fig. 14(c)], and the existence of a less significant contention from the background processes. In either case, the impact on the apparent performance is less visible than with the absolute one, which is expected.

*3) Observing the Effect of Processor Design:* We observe that processor design characteristics, especially with regards to cache memory, have a significant effect on which resource demands are dominating. For example, in Fig. 14, the keying operation in VMAC-AES128 appears to be affected more by the all-cores resource contention under Intel Atom D525 than under the Intel Core I5 650. We can relate such finding to the fact that the Intel Core I5 650 has a larger cache space and higher memory access throughput. Therefore, we conclude that whatever the effect of resource contention on the processing power is subject on the design characteristics of the evaluated architecture.

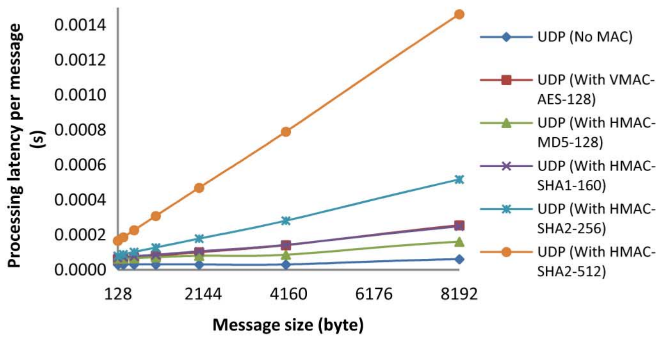
## *D. Observing Overhead of Applying MAC to Communication Sessions*

In this section, we discuss the impact of cryptography-based security measures on the performance of communication sessions. We focus on observing the significant impact of applying MAC on the UDP communication performance, especially when it comes to latency and jitter. We consider that the strength of a MAC function is typically correlated to the rate between the message size and its MAC size, where a lower rate translates to a stronger security.

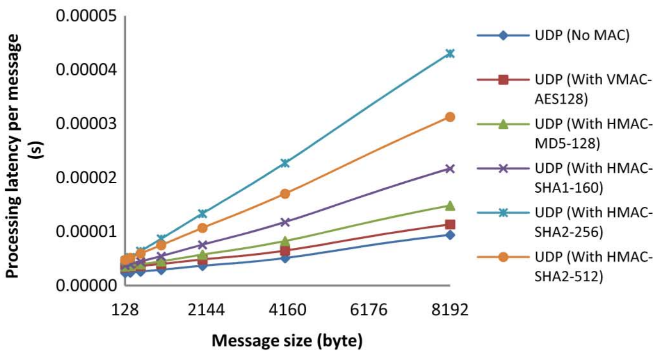
It is observed in Fig. 15 that the overhead in processing latency generally increases with: 1) the increased UDP message size; 2) the increased resulting MAC size; or 3) the increased rate of key changes. It is also noted that the overhead is more significant when it comes to ARM-based systems, since they usually lack the computational power or hardware assistance available to their X86 counterparts. This is also noticeable in the throughput performance illustrated in Fig. 16. Such variance in overhead makes ARM-based systems more vulnerable to denial-of-service attacks. This is a critical observation, especially given the current widespread of ARM-based systems in mobile IoT networks [29].

In Fig. 17, we observe the throughput performance of a UDP simplex communication, where all MAC functions generate 512-bit MACs for given message sizes. In MAC functions with MAC size less than 512-bit, we assume only two substitute techniques of generating a 512-bit MAC.

1) *Packetization*, where the message is divided into a number of parts equal to the rate between the desired MAC size and the function's MAC size. Each part is tagged individually, and the resulting MACs are combined to form the 512-bit MAC.
2) *Salting*, where the message is tagged $N$ times, where $N$ equals to the rate between the desired MAC size and the function's MAC size. For each process, the message is padded with a 1-byte random padding (aka salt). The resulting MACs are combined to form the 512-bit MAC.

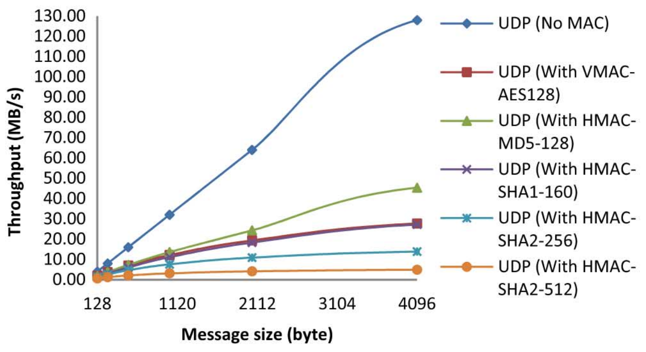


(a)

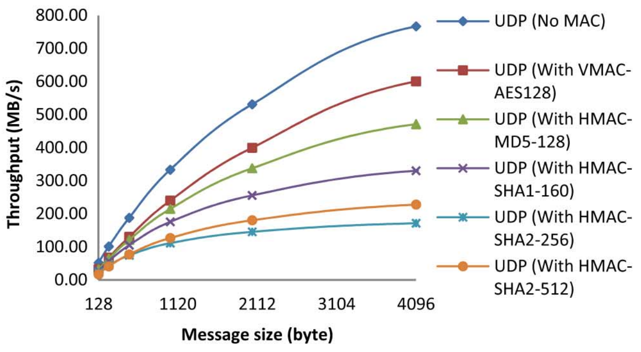


(b)

Fig. 15. UDP apparent processing latency under two selected architectures. (a) Texas Instruments' DM3730 ARM Cortex A8 (32-bit). (b) Intel Core I5 650 Dual-core (64-bit).



(a)



(b)

Fig. 16. Apparent throughput for a simplex UDP session under selected architectures. (a) Texas Instruments' DM3730 ARM Cortex A8 (32-bit). (b) Intel Core I5 650 Dual-core (64-bit).

We understand that MACs generated with aforementioned substitute techniques may not represent the true strength equivalence to native 512-bit MACs. However, generating a MAC of an equivalent strength will require substantially higher processing powers. We therefore, assume that the aforementioned substitutes generate strength-equivalent MAC for the purpose of this study in order to simplify the analysis.

It appears that using the packetization substitute to generate MACs of same size has less overhead compared with the salting. This is expected since salting involves computation on larger message sizes, which in turn requires higher processing power. The overhead variance between the packetization and the salting is less with smaller message sizes, where the processing power for preparing the MAC function has significant share of the total processing power required for the tagging operation.

It also appears that weaker MAC functions are not always faster when generating stronger MACs, especially when the message size is small. For example, HMAC-SHA2-512 has an advantage over all other functions (except for VMAC-AES128) generating MACs for 1024-byte messages under Intel Core I5 650, while HMAC-SHA1 has a comparable performance to VMAC-AES128 operating on 1024-byte messages under TI DM3730.

Given the previous findings, it is clear that the addition of security measures will lead to a significant degradation of UDP communication performance (which is also applicable to other communication protocols). In high-bandwidth and/or real-time services, such degradation can result in overutilization of local resources for communicating entities, especially the low-end systems, while having the network largely underutilized. The application of security measures will further lead to an increased gap in the communication performance between high-end and low-end computing systems. With the diversity implementation and performance trends of the available security measures under various computing architectures, it is not feasible to adopt a specific function (e.g., HMAC-SHA512) to offer data integrity measure for communications in all entities. In such scenarios, a security measure is needed that adapts to the context of the communication environments.

## VI. Further Considerations for Repeatability

Fellow researchers interested in replicating this work for validation and further expansion should keep in mind the descriptions offered in Sections III–V, in addition to the following considerations. Ignoring these considerations can affect the accuracy of measuring both absolute and apparent processing powers or render them useless as evaluation metrics.

In application development, interfacing with a function or procedure can be accomplished in several ways. Examples include calling a function in a separate library and calling a

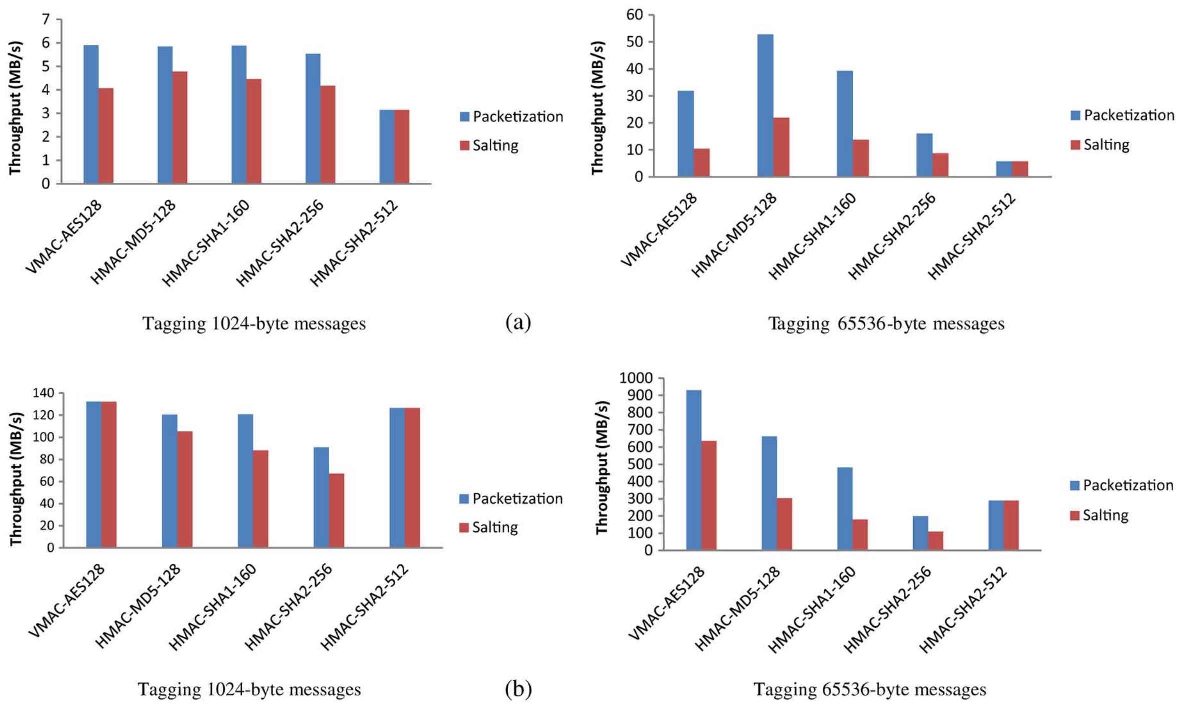


Fig. 17. Apparent throughput for a simplex UDP session under selected architectures, where messages are tagged with 512-bit MAC. (a) Texas Instruments' DM3730 ARM Cortex A8 (32-bit). (b) Intel Core I5 650 Dual-core (64-bit).

procedure from a separate process through pipes and sockets. As with any computing operation, the interfacing demands processor and I/O resources that can affect the performance of the evaluated security function in front of its calling application. Therefore, it is important to consider how the interfacing with a security function contributes to the evaluation. It is also important not to exclude the interfacing demands.

Meanwhile, some modern mobile and high-speed systems may offload security functions outside their main processor. Examples of places where security functions are offloaded into include *general purpose graphic processing unit* (GPGPU) and cryptographic processing units. While the performance can be evaluated from the offload unit, it is important not to ignore the main processor in the evaluation, since it is still handling communication sessions that call the offloaded security function. In such cases, it will be more logical to benchmark for the apparent processing performance, as if the main processor is executing that function, although it will not point to the absolute performance of the evaluated security function at the offload unit.

In a multiple processors/cores system, a thread can also migrate from one processor/core to another. If such a system does not maintain a synchronized clock across all the processors/cores, it is not possible to get the absolute performance for the evaluated security function. It will still be possible, however, to benchmark for the apparent performance. An estimate for absolute performance can then be made knowing the system's parallelism factor [30].

The physical context of a modern mobile system may limit the ability to conduct a controlled performance evaluation. Modern mobile system can have built-in hardware profiles that are internally applied based on the operating characteristics. For example, a system can adapt the available processing resources for different energy/battery levels. In such a scenario where the evaluator cannot control the profile dynamicity, the benchmarking should be conducted with the awareness of the system's ongoing operating context. Therefore, obtained performance measures should be mapped to the continuously probed operating characteristics at the time of measurement, resulting in additional overhead that increase challenges for obtaining a realistic evaluation in resource-limited mobile systems.

Finally, it is important to be aware of the scheduling algorithm used by the evaluated operating system. Some operating systems might schedule evaluation sessions unevenly, which will be mainly reflected in the measured apparent performance.

## VII. Conclusion

Current mobile computing architectures widely vary in terms of processing power, memory capabilities, I/O interfacing, and other hardware-specific optimizations and adaptabilities. Evaluating the impact of security functions on mobile systems analytically or based on nominal time and space requirements is not reflective of their performance in active systems, especially from a communication perspective. Existing benchmarking works also rely on absolute resource metrics, which are not delay-based and do not reflect actual communication demands nor many of today's mobile computing sophisticated operation and features. Addressing this dynamic nature, we proposed a carefully tailored benchmarking procedure for evaluating security functions in mobile systems. We also introduced the *apparent performance* metric, which is purposefully based on showing processing delays over actual processing performance. We observed various nonintuitive performance behaviors that necessitate revising how security functions are evaluated and selected in mobile communications, especially when it comes to time-based guarantees. Meanwhile, the suggested metric was shown to be more informative in the communications context, and a reasonable candidate metric in future protocol design.

## REFERENCES

[1] Internet Official Protocol Standards. (2008, May). *RFC 5000 IETF* [Online]. Available: http://www.ietf.org/rfc/rfc5000.txt

[2] M. Sokol, S. Gajewski, M. Gajewska, and L. Staszkiewicz, "Security and performance analysis of IPsec-based VPNs in RSMAD," in *Proc. 1st Int. Conf. Adv. Commun. Comput. (INFOCOMP'11)*, 2011, pp. 70–74.

[3] C. Shen, E. M. Nahum, H. Schulzrinne, and C. Wright, "The impact of TLS on SIP server performance," in *Proc. IPTComm'10*, 2010, pp. 59–70.

[4] B. Schneier and D. Whiting, "A performance comparison of the five AES finalists," in *Proc. 3rd AES Conf. (AES3)*, 2001, pp. 123–135.

[5] O. Hyncica, P. Kucera, P. Honzik, and P. Fiedler, "Performance evaluation of symmetric cryptography in embedded systems," in *Proc. IEEE 6th Int. Conf. Intell. Data Acquis. Adv. Comput. Syst. (IDAACS)*, 2011, pp. 277–282.

[6] D. J. Bernstein and P. Schwabe, "New AES software speed records," *INDOCRYPT*, vol. 5365, pp. 322–336, 2008.

[7] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software AES encryption," *Fast Software Encryption (FSE)*, vol. 6147. Berlin, Germany: Springer-Verlag, 2010, pp. 75–93.

[8] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Trans. Sens. Netw. (TOSN)*, vol. 2, pp. 65–93, Feb. 2006.

[9] J. Kaps and B. Sunar, "Energy comparison of AES and SHA-1 for ubiquitous computing," in *Proc. Embedded Ubiq. Comput.*, 2006, pp. 372–381.

[10] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Des. Test Comput.*, vol. 24, no. 6, pp. 522–533, Nov./Dec. 2007.

[11] I. Branovic, R. Giorgi, and E. Martinelli, "Memory performance of public-key cryptography methods in mobile environments," in *Proc. ACM SIGARCH Workshop Memory Perform. Dealing Appl. (MEDEA-03)*, New Orleans, LA, USA, 2003, pp. 24–31.

[12] H. Rifà-Pous and J. Herrera-Joancomarti, "Computational and energy costs of cryptographic algorithms on handheld devices," *Future Internet*, vol. 3, no. 1, pp. 31–48, 2011.

[13] D. S. Abd Elminaam, H. M. Abdual Kader, and M. M. Hadhoud, "Evaluating the performance of symmetric encryption algorithms," *Int. J. Netw. Secur.*, vol. 10, no. 3, pp. 216–222, 2010.

[14] Y. Yue and C. Lin, "Npcryptbench: A cryptographic benchmark suite for network processors," *SIGARCH Comput. Archit. News*, vol. 34, pp. 49–56, 2006.

[15] D. Bernstein and T. Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems* [Online]. Available: http://bench.cr.yp.to, accessed on Jul. 2014.

[16] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Process. Mag.*, vol. 26, no. 6, pp. 26–37, Nov. 2009.

[17] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors–A survey," *Proc. IEEE*, vol. 94, no. 2, pp. 357–369, Feb. 2006.

[18] A. M. Rashwan, A. M. Taha, and H. S. Hassanein, "Benchmarking message authentication code functions for mobile computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Anaheim, CA, USA, 2012, pp. 2585–2590.

[19] A. M. Rashwan, A. M. Taha, and H. S. Hassanein, "Characterizing the impact of dynamic resource management on message authentication in mobiles," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, N.S.W., Australia, 2014, pp. 1813–1818.

[20] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proc. 15th Euromicro Conf. Real-Time Syst.*, 2003, pp. 151–158.

[21] C. Paar, J. Pelzl, and B. Preneel, *Understanding Cryptography: A Textbook for Students and Practitioners*. New York, NY, USA: Springer, 2010.

[22] H. Krawczyk, M. Bellare, and R. Canetti. (1997, Feb.). *HMAC: Keyed-Hashing for Message Authentication. RFC 2104 IETF* [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[23] T. Dierks and E. Rescorla. (2008). *The Transport Layer Security (TLS) Protocol Version 1.2* [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[24] T. Iwata and K. Kurosawa, "OMAC: One-key CBC MAC," in *Fast Software Encryption (FSE'03)*, Lund, Sweden, Feb. 2003. LNCS vol. 2887, Springer-Verlag, Berlin, Germany, 2003, pp. 129–153.

[25] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, Gaithersburg, MD, USA: NIST Special Pub. 800-38B, May 2005.

[26] T. Krovetz, "Message authentication on 64-bit architectures," in *Proc. 13th Int. Conf. Sel. Areas Cryptogr.*, 2006, pp. 327–341.

[27] S. Tillich and J. Großschädl, "Instruction set extensions for efficient AES implementation on 32-bit processors," in *Proc. Cryptogr. Hardware Embedded Syst. (CHES'06)*, 2006, pp. 270–284.

[28] Intel Corporation. (2003, Jan.). *Intel® Hyper-Threading Technology Technical User's Guide* [Online]. Available: http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf

[29] K Karimi, "What the Internet of Things (IoT) needs to become a reality," INTOTHNGSWP REV 2, Freescale/ARM, White Paper, 2014.

[30] E. Biham, *Fast Software Encryption: 4th Int. Workshop (FSE"97)*. Berlin, Germany: Springer-Verlag, 1997.

**Abdulmonem M. Rashwan** (GSM'13–M'14) received the B.Sc. (Hons.) and M.Sc. degrees in computer engineering from Kuwait University, Kuwait, in 2001 and 2005, respectively, and is currently working toward the Ph.D. degree at Queen's University, Kingston, ON, Canada.

He is a Research Assistant with the Telecommunications Research Laboratory (TRL), School of Computing, Queen's University.

**Abd-Elhamid M. Taha** (S'03–GSM'06–M'07–SM'12) received the B.Sc. (Hons.) and M.Sc. degrees in electrical engineering from Kuwait University, Kuwait, in 1999 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Queen's University, Kingston, ON, Canada, in 2007.

He is currently an Assistant Professor of electrical engineering with Alfaisal University, Riyadh, Saudi Arabia.

**Hossam S. Hassanein** (S'86–M'90–SM'05) is a leading authority in the areas of broadband, wireless and mobile networks architecture, protocols, control, and performance evaluation. His record spans more than 500 publications in journals, conferences, and book chapters, in addition to numerous keynotes and plenary talks in flagship venues. He is the Founder and Director of the Telecommunications Research Laboratory (TRL), Queen's University School of Computing, Kingston, ON, Canada, with extensive international academic and industrial collaborations.