Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften

(Dr. rer. nat)

Security and Performance Tradeoff Analysis of Offloading Policies in Mobile Cloud Computing

eingereicht
am Institut für Informatik
des Fachbereichs Mathematik und Informatik
der Freien Universität Berlin
von

Tianhui Meng

Berlin, 2017

Gutachter:

Prof. Dr. Katinka Wolter
Department of Computer Science
Freie Universität Berlin, Germany

Prof. Dr. -Ing. habil. Armin Zimmermann
Institute for Computer and Systems Engineering
Technische Universität Ilmenau, Germany

Tag der Disputation: 10. Juli 2017

Selbständigkeitserklärung

Ich versichere, dass ich die Doktorarbeit selbständig verfasst, und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit hat keiner anderen Prüfungsbehörde vorgelegen.

Tianhui Meng

Berlin, den 12. Juli 2017

Abstract

While the last decades witness great advances in hardware technology, new mobile applications have also become much more demanding. Hence, mobile devices still face the restrictions in resources, such as battery life, storage capacity, and processor performance. In Mobile Cloud Computing (MCC), offloading is a popular technique proposed to augment the capabilities of mobile systems by mitigating complex computation to resourceful cloud servers. While offloading may be beneficial from the performance and energy perspective, it certainly exhibits new challenges in terms of security due to increased data transmission over networks with potentially unknown threats.

Among possible security issues are timing attacks which are not prevented by traditional cryptographic security. Timing attacks belong to side-channel attacks in which the attacker attempts to compromise a system by analyzing the time it takes to respond to various queries. Offloading is particularly vulnerable to timing attacks because it often needs many times sending/receiving. So metrics on which offloading decisions are based must include security aspects in addition to performance and energy-efficiency. This thesis covers both the theoretical and practical aspects of offloading policies in MCC systems. Unlike previous work that only considers the performance and energy perspectives, this thesis presents and evaluates offloading policies based on security-performance tradeoff analysis to satisfy the security and performance requirements in offloading systems. Proposed stochastic models are applied and evaluated by numerical simulation and real world experiments. Specifically, the contributions of this thesis can be summarized as follows:

- Several stochastic model-based approaches to quantitatively assess the security and performance attributes of the mobile cloud offloading system are proposed.
- Methods to formulate metrics that include both, performance and security aspects and that optimise the tradeoff between the two are studied.
- A secure and cost-efficient offloading policy considering the specific threat of timing attacks against MCC systems is proposed and the offloading policy is evaluated with experiments.
- Two widely used secure containers for Android: Samsung Knox and IBM MaaS360, to en-

hance the client security in MCC systems are compared.

Acknowledgements

My study is under the financial support by China Scholarship Council (CSC) and has been carried out at Computer Systems & Telematics (CST) group at Freie Universität Berlin, Germany. I am very thankful to CSC and CST for providing me with such a valuable working opportunity. I would like to express my sincere gratitude to everyone who contributed to the completion of this thesis.

First and foremost, I want to thank my supervisor Prof. Dr. Katinka Wolter. I really appreciate her invaluable advice and encouragement throughout my four years research. Without her advice and patience, this thesis would have never been possible. I admire her work enthusiasm and professional knowledge deep in my heart. A lot of thanks to her for helping me fulfill my potential and leading me on my research road.

Many thanks to Professor Dr.-Ing. habil. Armin Zimmermann for his effort in reviewing this thesis and for his helpful comments throughout the process. Thanks to Prof. Dr.-Ing. Jochen Schiller. He is a very nice person and a good leader of CST group. I also would like to thank Prof. Dr. Marian Margraf, Prof. Matthias Wählisch, Stephanie Bahe, Dr. Philipp Reinecke, Dr. Norman Dziengel, Dr. Oliver Hahm, Dr. Emmanuel Baccelli, Dipl.-Inform. Martin Seiffert, Dipl.-Ing. Stephan Adler, M.Sc. Simon Schmitt, Hauke Petersen and all the members of CST for their kindly help and selfless support during the research. Thanks my colleagues for creating an ever nice and friendly working atmosphere in the institute. I enjoyed working with them very much. I wish to thank Dr. Yubin Zhao, Dr. Yuan Yang, Dr. Qiushi Wang, Dr. Huaming Wu and Dr. Yi Sun, who were an inexhaustible source of inspiration and helped whenever I was in need. Without their countless discussions I had certainly been stuck in blind alleys several times. Many thanks to my Chinese colleagues, Zhihao Shang and Han Wu for their help and company.

Last but not least, I am very grateful for the love and support of my parents. They dedicate everything they have to bring me up and teach me how to be a good man. They always patiently guide me when I am in darkness. I would like to thank my girlfriend Dr. Ting Ma. She is my strongest supporter whenever I encounter difficulties. She is the source of my strength and happiness. I cannot

make it without her love.

Thank you. Thank you all.

Contents

Part I

Introduction

Chapter 1

Basic Concepts and Problems

1.1 Problem Statement

Cloud computing has been widely accepted as computing infrastructure of the next generation, as it offers advantages by allowing users to exploit platforms and software provided by cloud providers (e.g., Google, Amazon and IBM) from anywhere on demand at low price [122].

Mobile devices, such as smartphones, tablets and smart watches, have become mandatory items in today's world. They are no longer used only for voice communication and short message service (SMS); instead, they are used for watching videos, gaming, web surfing, recording health data and social networking. Mobile devices are becoming progressively an important constituent part of every day's life as very convenient communication and business tools with a wide variety of software covering all aspects of life. Although the last decades witness great advances in hardware technology, new applications have also become much more demanding. Hence, mobile devices still face the restrictions in resources, such as battery life, network bandwidth, storage capacity, and processor performance.

Mobile cloud offloading is a promising solution to augment the mobile systems' capabilities by migrating computation via WiFi or 3G/LTE to more resourceful servers (i.e., Windows Azure and Amazon EC2 web services) [36]. The concept of computation offloading has been proposed with the objective to avoid a long application execution time on mobile devices, which results in large power consumption. This is different from the traditional client-server architecture, where a thin client always migrates computation to a server [82]. The offloading client has the capability to do the computation tasks locally on the mobile device. In mobile cloud offloading, applications are often divided into two parts: one of those parts that can be offloaded and the other that must be

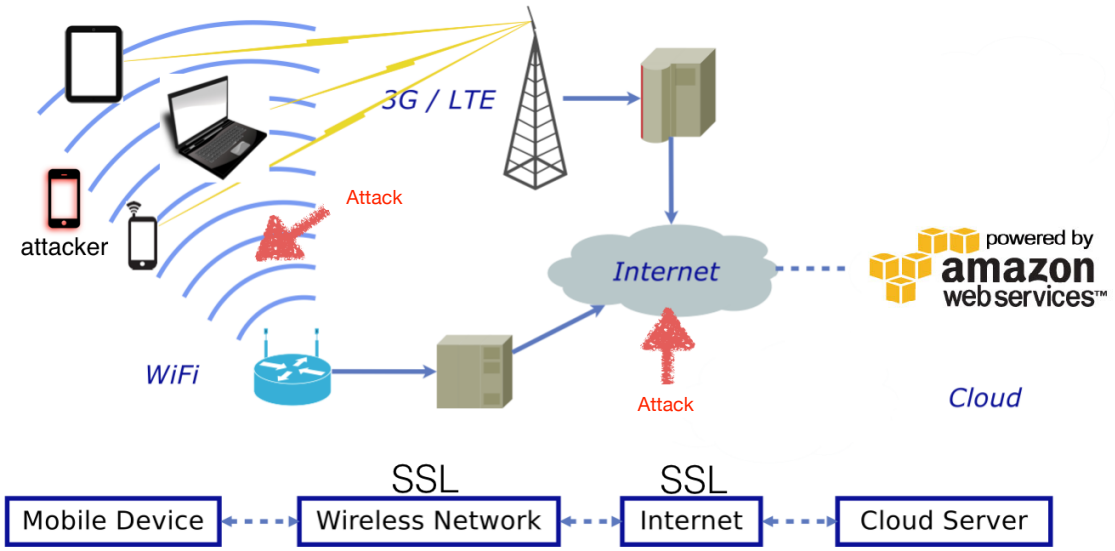


Figure 1.1: An illustration of Mobile Cloud Offloading system

executed on the mobile device, such as the user interface [136].

In many scenarios, the limited computing speeds of mobile systems can be enhanced by mobile cloud offloading. One example is a context-aware computing infrastructure proposed in [57] –where multiple streams of data from different sources like GPS, maps, accelerometers and temperature sensors need to be analyzed together in order to obtain real-time information about a user's context. Even though battery technology has been steadily improving, it has not been able to keep up with the rapid growth of power consumption of mobile systems. Offloading may extend battery life by migrating the energy-intensive parts of the computation to servers [83].

The smooth offloading of computation depends on a fast and stable network connection, which guarantees seamless communication. But in an unreliable network, task completion can be delayed or interrupted by congestion or packet loss, in which case offloading may not always benefit [130]. This involves making a decision regarding whether and what computation to migrate, which usually depends on many parameters such as the network bandwidth and the amounts of data exchanged through the networks.

Over the last years, research on computation offloading focused on how to offload and what to offload from mobile devices to cloud servers in order to reduce the execution time and power consumption of computation tasks [82]. Several offloading infrastructures have been developed for offloading at varying granularity, among which the MAUI offloading system, presented in 2010, not

only achieves significant reduction in energy consumption for some jobs on mobile devices, but also improves the performance of mobile applications (i.e., refresh rate of a game can increase from 6 to 13 frames per second) [36]. In addition, instead of offloading the full code, MAUI partitions the application code at runtime to maximize energy savings. However, several challenges still exist in the following three aspects of mobile offloading systems:

Time and energy consumption in data transition

Data transmission over wireless or cellular networks is of highly unpredictable quality. Wu [138] proposed metrics to express the energy response time tradeoff, the Energy-Response time Weighted Sum (ERWS) and Energy- Response time Product (ERP) [50] for mobile offloading systems which can be optimised using different offloading policies.

Lossy network

Low bandwidth or long delays are possible factors incurring connectivity problems. Consequently, when migrating computation to the cloud server, the execution of the offloading task may suffer from long delays or even failures by in the network. Limited battery capacity of the mobile device prohibits unpredictable waiting times, which may also be caused by a long recovery process. A dynamic scheme to determine whether and when to launch the local re-execution, instead of always waiting for network recovery to offload [130] may help to deal with this problem.

Security and data confidentiality

Along with the benefits of high performance, the offloading system witnesses potential security threats including compromised data due to the increased number of parties, devices and applications involved, that leads to an increase in the number of points of access. Security threats have become an obstacle in the rapid expansion of the mobile cloud computing paradigm. Significant efforts have been devoted to research organisations and academia to build secure mobile cloud computing environments and infrastructures [68]. However, works on modelling and quantifying the security attributes of mobile offloading system are rare.

Quantitative analyses of system dependability and reliability have received great attention for several decades. However quantification of security has only recently attracted more attention, and some initial conceptual work has been published already decades ago. A series of model-based evaluation of security mechanisms has been published only recently. But few studies have considered the quantitative evaluation of security and the tradeoff with performance [134].

1.1.1 Security Issues

Although the cloud based approach can dramatically extend the capability of mobile devices, the assignment of developing a secure and reliable mobile cloud offloading system remains challenging [68]. Protecting user privacy and data secrecy from an adversary is a key to establish and maintain consumers' trust, especially in Mobile Cloud Computing (MCC) systems. Thus metrics on which offloading decisions are based must include security aspects in addition to performance and energy-efficiency. In recent years, numerous works about security in mobile cloud offloading and cloud computing have been presented [37, 54, 67, 93].

Since offloading servers may lie in any corner of the world beyond the reach and control of users, there are many security and privacy challenges that need to be understood and taken care of in mobile offloading. Also, one can never ignore the possibility of a server breakdown that has been witnessed, rather quite often in the recent times [16].

There are several security challenges existing in the mobile cloud offloading scenario. Attacks can happen on both the client side and the server side as well as the communication between them (Figure 1.1). As reported by the Open Web Application Security Project (OWASP) [6], the "top 10" vulnerabilities in cloud-based or Software as a Service (SaaS) deployment models are listed: accessibility vulnerabilities which are intrinsic in TCP/IP such as denial of service (DoS) and distributed denial of service (DDoS); web application vulnerabilities like cross-site scripting and Structured Query Language (SQL) injection; authentication of respondent devices; data verification; physical access issues; privacy and data confidentiality including the increasing threat of data compromise in the cloud, due to the increasing number of devices and applications involved; and trust computations as the unencrypted data must reside in the memory of the host running the computation.

One important threat against the offloading system is the side-channel attack, which is not covered by traditional cryptographic security [22]. Among side-channel attacks are *timing attacks*, whose remote feasibility has been proven in [20]. A timing attacker does not require special equipment or physical access to the machine, which makes a practical threat against web services as well as mobile cloud offloading systems [19]. Protecting cryptographic implementations against timing attacks is one of the most important challenges in modern cryptography [15, 73]. Timing attacks are based on information gained from the service response time. In a timing attack, the attacker deduces information about a secret key from runtime measurements of successive requests.

Offloading is particularly vulnerable to timing attacks because it often needs many times sending/receiving. In a mobile cloud offloading system, an attacker can pretend to be a normal client and send offloading requests to the cloud server. Timing attacks enable the attacker to extract secrets maintained in the cloud server by observing the time it takes the server to respond to various

queries [21]. The process of a timing attack can be interrupted by frequently changing the key in the server [111] or mitigated by random padding in response time [73].

1.1.2 Tradeoff Analysis

A tradeoff is a situation that involves losing one quality or aspect of an object in return for gaining another quality or aspect. Tradeoff between security and throughput is always a major concern in wireless networks [64].

In mobile computing, security mechanisms such as security protocols or encryption algorithms bring a cost to the system in terms of processing effort and energy consumption. For instance, when Advanced Encryption Standard (AES) is implemented to protect the communication between mobile devices, the messages have to be encrypted before sending, which may degrade the performance. In this scenario, the tradeoff is that encryption with longer AES keys is slower because it requires more computing power to process. So the longer the encryption key the higher the encryption cost.

However, if the encryption key length is short, it is easier for an attacker to compromise the mobile computing system. A security incident has associated costs as well. So one has to choose an optimal key-length for the system as to keep encryption costs and security incident costs together as low as possible [134]. In the presence of timing attacks, we investigate how to the set system parameters to obtain the optimum tradeoff between security and performance in mobile computing systems.

1.2 Contributions

Performance and security are two major attributes that must be taken into consideration when designing mobile cloud offloading systems. As a system administrator, one may want to improve performance and security at the same time. Accordingly, we consider three objectives as follows:

- **Improving offloading performance**: by offloading heavy workload to resourceful cloud servers, mobile users want to reduce the job execution time on the mobile devices or increase the system throughput.
- **Quantitative security assessment**: protecting user privacy and data integrity are gradually attracting people's attention in recent years, especially in mobile computing systems where data resides over a set of networked resources. However, studies on assessing security quantitatively are still rare. In order to proceed to a quantitative treatment of the performance-security tradeoff of offloading systems, we first have to provide methods to assess security quantitatively.

- **Security and performance tradeoff**: security and performance are both critical design objectives of mobile cloud offloading systems. As discussed in Section 1.1.2, improving security of the system often brings an overhead that affects performance. One may look forward to improving security of offloading systems while maintaining performance within an acceptable range. So the tradeoff analysis of security and performance needs to be performed in the offloading scenario.

The main contributions of this thesis propose several model-based measures of assessing performance and security of mobile cloud offloading systems, and proceed to improve the tradeoff of both attributes. In the following we give a short overview of the major contributions. A chapter-by-chapter summary of this thesis is introduced in Section 1.3. The major contributions of this work are listed as follows:

1) We propose several stochastic model based approaches to quantitatively assess the security and performance attributes of the mobile cloud offloading system.
2) We show methods to formulate metrics that include both, performance and security aspects and that optimize the tradeoff between the two. By solving the proposed hybrid CTMC and queueing model, the optimal rekeying rate is determined for several system metrics. We found that with carefully selected parameters, we can configure the offloading system to achieve an optimal security and performance tradeoff.
3) By combining renewing the server key regularly with inserting random delays into the server processing time, we propose a secure and cost-efficient offloading scheme for MCC systems. Our experimental results show that the security performance tradeoff of offloading can be improved through our scheme.
4) We implement a system that allows us to compare the impact of different random padding strategies on the expected success of timing attacks. It is revealed that the variance of random delays is the decisive factor to mitigation effectiveness of a random padding and the extra number of measurements that an attacker has to make grows linearly with the standard deviation (SD) of the random padding.
5) With respect to client security issues, we perform an empirical comparison between two popular secure container solutions for Android: Samsung Knox and IBM MaaS360, and show several experimental results. Our experimental results shows that it is more secure for an application to run in the Knox container than in the MaaS360 container. We also found that mobile devices witness a notable deterioration in the performance of compute-intensive applications using the Knox container. However, for a memory-intensive application, performance does not degenerate much in Knox and MaaS360 containers comparing with running on the

device (outside the container).

1.3 Thesis Structure

This thesis consists of three parts. In the first part we generally introduce the background of this thesis and some related work.

First, in **Chapter 1** we present some elements of this thesis: where it happens, when it happens and how it develops. This thesis focuses on the field of the offloading technique in MCC. We introduce the basic concept of mobile cloud offloading and the security issues in offloading systems. The initial idea of tradeoff analysis is presented and the contributions of this thesis are summarized.

Chapter 2 first describes the basic characteristics of mobile cloud offloading. We discuss the tradeoff for offloading decisions and the existing offloading frameworks are compared in terms of their parameters, potential application and optimization strategy. We provide a brief introduction of the timing attack, which is the main security threat we consider in this thesis. Then we survey related work regarding existing mobile cloud offloading schemes, security assessment techniques and side-channel attacks.

The second part of the thesis illustrates the quantitative security assessment methods of the mobile cloud offloading system and the tradeoff analysis of its security and performance attributes.

In **Chapter 3** we first present an introduction to the stochastic modeling techniques used throughout this thesis. Then a state transition model is proposed for a general mobile cloud offloading system under the specific threat of timing attacks. Our model aims to quantitatively assess the security attributes of an offloading system. We show how to formulate metrics that include both, performance and security aspects and that optimize the tradeoff between the two. System metrics are evaluated by solving the steady-state probabilities of the proposed model.

Chapter 4 illustrates the issue of security and performance tradeoff analysis of mobile cloud offloading. We investigate how to quantitatively assess the security attributes and their impact on the performance. The security model proposed in the previous chapter is improved and extended with a performance model. By transforming the security model to a model with an absorbing state, the "mean time to security failure" (MTTSF) measure is computed.

In **Chapter 5** we propose a secure and cost-efficient offloading scheme for MCC by combining renewing the server key regularly with inserting random delays into the server processing time. A system is implemented to evaluate our scheme and to compare the impacts of different random padding strategies on the expected success of timing attacks.

The third part of the thesis concerns how to improve the client security of mobile cloud offloading systems.

The analysis presented in chapters 4 and 5 mainly considers the server security of offloading systems, while we address the client security in **Chapter 6**. We perform an empirical comparison between two popular secure containers for Android: Samsung Knox and IBM MaaS360. Benchmark tests are conducted to compare these two containers. Then in order to quantitatively assess the security property of these containers, we propose a measurement method based on a simulating attack. We arrive at guidelines how to select the secure containers in the actual application environment.

Chapter 7 concludes the thesis with a summary. The contribution of proposed offloading policies is emphasized. In the end, we provide the outlook of our future work.

Chapter 2

Background and Related Work

In MCC, compute-intensive applications suffer from the restricted resources of mobile devices, such as limited battery lifetime and memory. Mobile cloud offloading is a popular solution to mitigate these restrictions and enhance the capabilities of the mobile system by migrating large computation and complex processing from resource-limited devices to more resourceful computers (i.e., servers) [83]. In this chapter, we first provide a brief introduction of mobile cloud offloading. In particular, we are interested in different offloading architectures and frameworks. In the second part of this chapter, we introduce the procedure of a timing attack to mobile cloud offloading. The timing attack is the main security issue on which we focus our attention in this thesis. In addition, the existing countermeasures against this attack are surveyed. Last, we present the related work of this thesis. We give an overview of existing mobile cloud offloading schemes and model analysis methods for offloading and timing attack investigations. Our reviews are intended to serve as a guideline for a structured approach to building secure and cost-efficient offloading systems.

2.1 Mobile Cloud Offloading Overview

In this section we introduce the mobile cloud offloading architectures and describe a comparison of the existing offloading frameworks.

2.1.1 Generic architecture

The general architecture of mobile cloud offloading system can be depicted as in Figure 2.1. Offloading users with mobile devices are connected to powerful cloud servers in different forms. A simple way is through Wi-Fi networks where mobile devices connect to the Internet via Wi-Fi

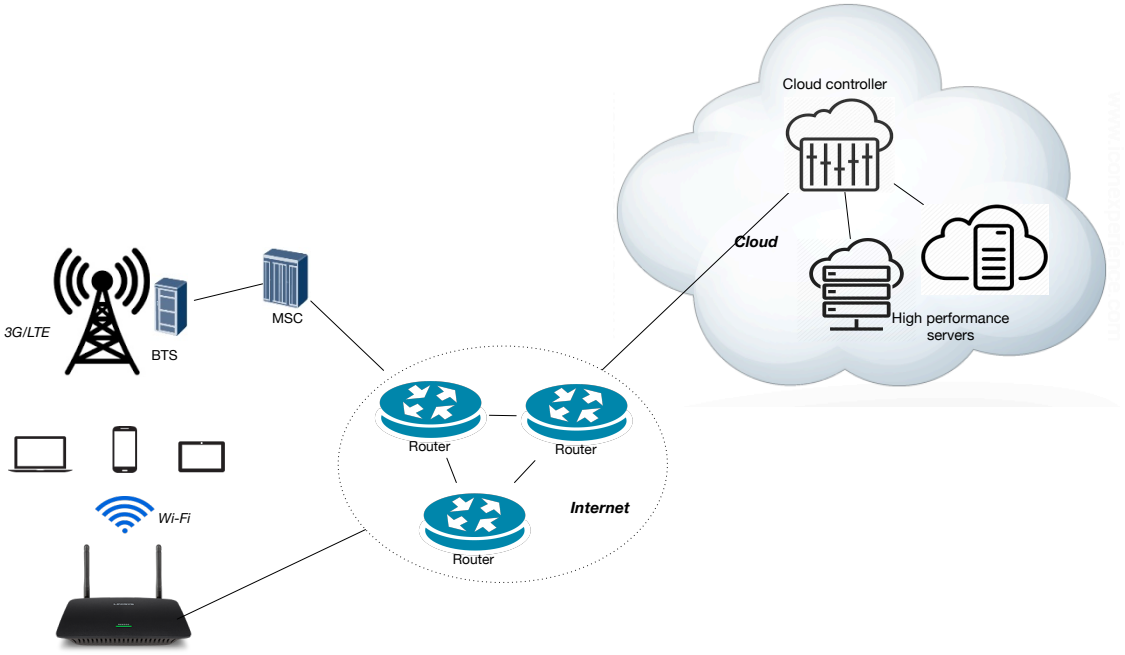


Figure 2.1: Mobile cloud offloading architecture [69]

access points (wireless routers). But this form has range limitations, e.g., eduroam [2] signal only covers the area near education buildings. A more flexible and long range connection is through cellular networks, where mobile users first connect to a 3G/LTE network through devices such as Base Transceiver Station (BTS) and Mobile Switching Center (MSC) to transmit data to the Internet. Base stations establish and control the connections (air links) and functional interfaces between the networks and mobile devices. This connection has much higher availability than Wi-Fi because it has high coverage. However the average data rate of cellular networks is often lower than Wi-Fi.

At runtime, mobile applications discover a cloud service and offload jobs to it. A mobile application generates the jobs of the offloading system which can either execute them locally on the mobile device or offload them to cloud servers. In the latter case the mobile clients offload jobs to a server and receive the computation results sent back by it. The offloadable jobs can be any computation-intensive or energy-intensive jobs such as Optical Character Recognition (OCR), real-time translation and chess algorithm computation, to name some examples. The offloading decision mechanisms on the mobile systems are managed by the system administrator by a mobile device management (MDM) system. The administrator makes offloading decisions based on performance and security criterion.

In the mobile cloud offloading system, mobile network operators can provide services to mobile users such as authentication, authorization, and accounting based on the home agent and subscribers' data stored in databases. After that, the mobile users' requests are delivered to the cloud service providers (e.g. Amazon Elastic Compute Cloud EC2 and Simple Storage Service S3) through the Internet. In the cloud, cloud controllers process the requests to provide mobile users with the corresponding cloud services. These services are developed with the concepts of utility computing, virtualization, and service-oriented architecture (e.g., web, application, and database servers) [43].

2.1.2 Tradeoff for offloading decisions

In mobile cloud offloading, mobile devices offload heavy workloads to cloud servers in order to reduce execution time and energy consumption. But offloading is not always beneficial since it brings an overhead caused by data transmission and the effort for executing offloading codes. At the same time, mobile users are confronted with dynamically changing network situations due to user mobility, that makes it hard to make offloading decisions in mobile situations [85]. So it is reasonable to make good offloading decisions based on several criteria to make sure that offloading benefits the user.

From the perspective of execution time, mobile cloud offloading may improve performance when the execution (including data transmission and computation) can be performed faster at the server than the mobile device [69]. Let T_s be the time to execute the offloading job on the cloud server, T_c be communication time involving establishing connection and data transmission and T_l be the local execution time on the mobile device. That is, the server execution time T_s together with the communication time T_c is shorter than the execution time T_l on the mobile device:

$$T_s + T_c < T_l \ . \tag{2.1}$$

In order to reduce the energy consumption on the mobile device, we let the E_l be the energy used by the mobile device when the application is locally executed. Let E_t be the energy required for transmitting offloading jobs and E_i be the energy used when the application is idle, waiting for the server to send back the result. Mobile cloud offloading is beneficial from the energy consumption perspective if it satisfies:

$$E_t + E_i < E_l \ . \tag{2.2}$$

While employing mobile cloud offloading may be beneficial from the performance and energy perspective, the system surely exhibits new security challenges because of increased data transmis-

sion over networks with potential threats. Thus, we have to take security into consideration when making offloading decisions in addition to performance and energy-efficiency. Metrics such as confidentiality and mean time to security failure (MTTSF) can serve as criterion to judge the security of mobile offloading systems. We let the C_o be the original confidentiality of the offloading system and C_s be the confidentiality when we develop a security strategy to protect the system. In order to improve the system security, the strategy has to satisfy:

$$C_s > C_o \, . \tag{2.3}$$

In the rest of this thesis, we propose tradeoff metrics to investigate the interrelation between security and performance perspectives of mobile offloading systems.

2.1.3 Existing frameworks

Table 2.1.3 describes a comparison of several existing mobile cloud offloading frameworks in terms of their applications, tradeoff parameters and optimization strategies.

Generally speaking, execution time, energy consumption and data transmission are the main tradeoff parameters that most of the offloading frameworks use. But the EECOF (Energy Efficient Computational Offloading Framework) particularly investigates the tradeoff between the size of data transmission and energy consumption cost in offloading different components of the application. The frameworks ThankAir, CloneCloud and DiET are proved to be effective for scientific computing applications, whereas the Cuckoo, MAUI and MACS are usable for applications which do a lot image processing work such as face detection. The EECOF framework is shown to work for a SOA (service-oriented architecture) prototype application and SociableSense is designed specifically for applications requiring social interaction measurement.

As for the strategies to optimize the offloading decision, most of the existing offloading frameworks utilize heuristics based on ILP (Integer Linear Programming), parallelizing model, and multi-criteria decision theory. As a practical implementation of mobile cloud offloading for Android, Cuckoo uses a very simple heuristics to always prefer offloading to remote servers.

Framework	Tradeoff parameters	Application	Optimization strategy
CloneCloud [28]	Execution time, energy consumption	Virus scanner, image searching	Integer Linear Programming
Cuckoo [66]	Execution time, energy consumption	Object recognition, PhotoShoot game	Always prefer remote execution
ThinkAir [76]	Execution time, energy consumption	Great Computer Language Shootout, N-queens	Parallelizing method based
MAUI [36]	Execution time, Energy consumption, offloaded data	Face-recognition, chess and video game	0-1 Integer Linear Programming
MACS [78]	Execution time, energy consumption, data transmission	Face detection and recognition	Cost function based
SociableSense [110]	Energy consumption, latency, data transmission	Sociability measurement application	Multi-ciriteria decision theory
EECOF [118]	Data transmission, energy consumption cost	SOA prototype application	Preconfigured service access
DiET [112]	Execution time, size of transmitted data, throughput	SciMark 2.0 (a scientific computing benchmark)	User configuration based

Table 2.1: The Comparison of the existing offloading frameworks

2.2 Timing Attacks

As more and more information on individuals and business are placed in the cloud, concerns today are serious about how safe an environment is. One should take security metrics into account when making mobile cloud offloading decisions, because security concerns become the main barrier to the development of MCC [68]. In this thesis, we mainly deal with the threat of the timing attack whose remote feasibility has been proved [20]. In the following, we introduce the procedure of timing attacks.

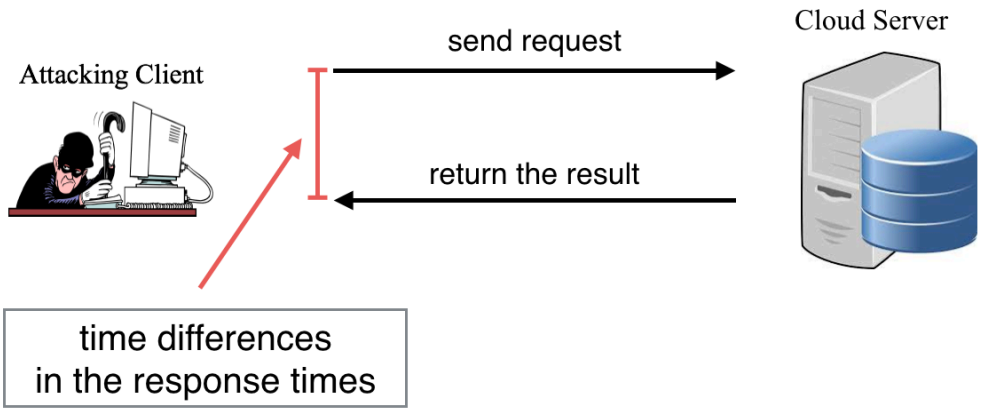


Figure 2.2: An illustration of A Timing Attack

Implementations of cryptographic algorithms often perform computations in non-constant time, due to performance optimization [42]. If such operations involve secret parameters, these timing variations in cryptographic computation can leak some information and a careful statistical analysis could even arrive at the total recovery of the secret keys [111].

Figure 2.2 shows an illustration of a timing attack. In the offloading system, a group of mobile devices offload computation to a cloud server supporting a web service. The server is an Apache web server with OpenSSL. We assume there is an attacking client conducting timing attacks to the cloud server. The cloud server can not discover the attacker because it sends only regular requests to the server. The attacking client sends requests to the server and the server returns back the result. What the attacker does additionally is that it records the response times.

The attacking client gains information about the server's secret by analysing the response time measurements. Simply, a timing attack is that an attacker repeatedly sends guesses about a secret value to the server, which rejects them as incorrect. However, if the first bit of the guess is correct, it takes slightly longer to return the error. With many measurements and some filtering, the attacker can distinguish this difference and then decrypt bit by bit the server's secret.

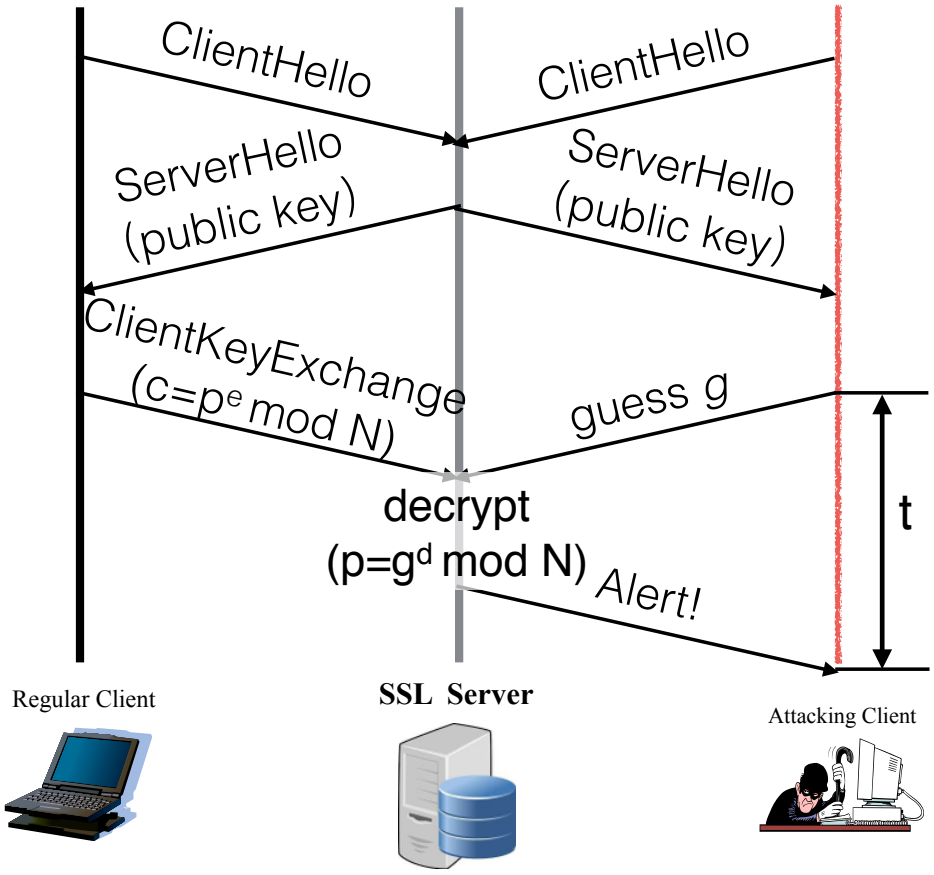


Figure 2.3: Brumley's remote timing attack

2.2.1 Brumley's attack

It was commonly believed that timing attacks can be directed only towards smart cards or affect inter-processing locally, but some studies reveal that remote timing attacks are also possible and should be taken into consideration [21, 111].

We take Brumley's attack [21] as an example and explain how remote timing attacks work. As shown in Figure 2.3, a regular client establish a connection with the cloud server use a three-way handshake. During a standard full SSL handshake the SSL server performs an RSA decryption using its private key. The SSL server decryption takes place after receiving the CLIENT-KEY-EXCHANGE message from the client. The CLIENT-KEY-EXCHANGE message is composed on the client by

encrypting PKCS 1 padded random bytes with the server's public key. The randomness encrypted by the client is used by the client and server to compute a shared master secret for end-to-end encryption.

Upon receiving a CLIENT-KEY-EXCHANGE message from the client, the server first decrypts the message with its private key and checks the resulting plain text for proper PKCS 1 formatting. If the decrypted message is properly formatted, the client and server can compute a shared master secret. If the decrypted message is not properly formatted, the server generates its own random bytes for computing a master secret and continues the SSL protocol. Note that an improperly formatted CLIENT-KEY-EXCHANGE message prevents the client and server from computing the same master secret, ultimately leading the server to send an ALERT message to the client indicating the SSL handshake has failed.

In Brumley's attack, the client substitutes a properly formatted CLIENT-KEY-EXCHANGE message with a guess value g. The server decrypts g as a normal CLIENT-KEY-EXCHANGE message, and then checks the resulting plain text for proper PKCS 1 padding. Since the decryption of g will not be properly formatted, the client will receive an ALERT message from the server. The attacking client computes the time difference from sending g as the CLIENT-KEY-EXCHANGE message to receiving the response message from the server as the time to decrypt g. The attacker repeats this process for many guess value of g. With carefully analysing the response time measurements, the attacker can guess the server's private key bit by bit.

Because timing attacks can be conducted remotely without the need of touching the hardware, mobile cloud offloading systems are surely vulnerable to such attacks. However this threat is not covered by traditional cryptographic security. Mobile cloud offloading requires access to resourceful servers for short duration through wireless networks. These servers may use virtualization techniques to provide services so that they can isolate and protect different programs and their data. However, in [133] it is shown that using a cache timing attack, an attacker can bypass the isolated environment provided by virtualization characteristics, where sensitive code is executed in isolation from untrustworthy applications. It is worth mentioning that a timing attack also poses a threat to other types of systems such as vehicular networks [123] and wireless sensor networks (WSNs) [95].

2.3 Related Work

The computation offloading is not a novel concept since it has evolved from many paradigms of distributed computing [12, 16, 43, 47, 52, 68, 79, 82, 122]. In this section we summarize the related work in the directions of mobile cloud offloading approaches, model analysis techniques and timing

attacks.

2.3.1 Offloading approaches

Due to the limitation of relatively low battery capacity of mobile devices as well as fragile mobile networks, a significant amount of studies have been performed on offloading computation to the cloud to achieve two main objectives: to extend battery lifetime [40, 83, 108, 137], and to shorten execution time of heavy applications [25, 109, 128–130]. In order to optimize the offloading gain, mobile cloud offloading involves making a decision about *whether*, *when* and *what* to offload. The latest works on mobile cloud offloading [25,40,109,135] focus on the offloading decision problems.

In order to achieve efficient computation offloading, Chen [25] propose a game theoretic approach for offloading decision making problem. Researchers in [40] consider a mobile computation offloading problem where multiple mobile services in workflows can be invoked to fulfill their complex requirements. To address unstable connectivity of mobile networks, a novel offloading system is proposed to design robust offloading decisions for mobile services. It is also worth mentioning that in [14], a precise evaluation of the feasibility and costs of offloading processes in terms of bandwidth and energy consumption are presented. The researchers conduct measurements on a testbed of 11 Android devices and the same number of software clones running on the Amazon EC2 cloud.

A vast body of research categorizes mobile cloud offloading approaches into static and dynamic resting with regard to when the offloading decisions are made. As shown in Figure 2.4(a), the static offloading utilizes performance prediction models and offline profiling based on measured data to estimate the offloading gain [87, 104, 131, 139]. The application is partitioned into client and server part to be executed [69]. The static partition approaches have the advantage of low execution overhead, but the parameters are very dependent on the accurate prediction of system performance.

In contrast, the dynamic offloading approach (Figure 2.4(b)) can adapt to variable conditions. The dynamic strategies initially perform static analysis of the application code and instrumentation in order to prepare for dynamic profiling [24, 36, 41, 58, 59]. During the execution, the offloading system keeps updating its configuration based on the information obtained from dynamic profiling. Meanwhile, dynamic decisions are often subject to high overhead caused by the run-time situation monitoring and profiling. For ease of reference, Table 2.2 summarizes all the related work in terms of their offloading decision approach, parameters and core component.

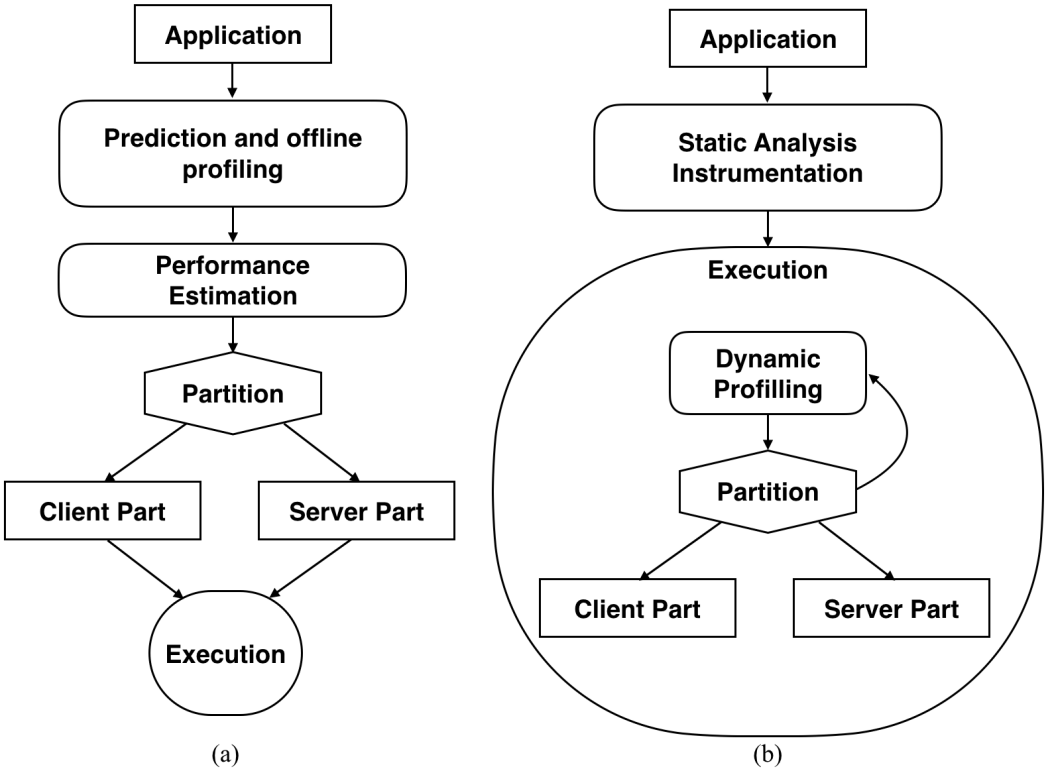


Figure 2.4: Flowchart of static and dynamic offloading process

- **Static approaches**

The first offloading approach we consider is suggested in [87]. It generates a cost graph for the application which takes into account the computation time and the data to be transmitted. The sum of both these parameters is minimized by a branch-and-bound algorithm and a pruning heuristic that reduces the search space to provide a near-optimal result.

Wang and Li [126] present a computation offloading scheme using program abstraction to partition an ordinary program into client-server distributed subprograms to be run on a device or a server. A polynomial time algorithm is suggested to achieve optimal partitioning of programs for a given set of inputs. Their scheme is designed to guarantee the right distributed execution under different contexts.

An adaptive method presented in [139] performs computation offloading to save energy on a mobile client which uses an initial profile and timeout obtained by executing the program. It does not need to estimate the execution time of each computation instance. With the reduced energy

consumption, an improvement in the performance is achieved for image processing benchmarks.

In [131], the researchers propose a graphic rendering adaptation technique which can adapt the game rendering parameters to satisfy cloud mobile gaming processing and communication constraints, such that the overall mobile gaming user experience is maximized. First, a static analysis is performed to select optimal settings for game rendering. While executing, the rendering settings are modified based on the communication and computation costs. Their experimental results indicate that the Game Mean Opinion Score (GMOS) is improved corresponding to the user experience.

CloneCloud [28] is an offloading framework that makes an effort to allow the execution of a mobile application on the cloud. CloneCloud uses static analysis and dynamic profiling corresponding to different inputs to partition applications while optimizing execution time and energy use for a target computation and communication environment. It constructs a profile tree to represent the execution traces of the application. At runtime, partitioning is done by migrating a thread from the mobile device at a specified point to the clone in the cloud, executing in the cloud for the remainder of the partition, and reintegrating the migrated thread back to the mobile device.

An offloading approach aiming at improving the execution performance is presented in [104] using the branch-and-bound and min-cut based strategies for partitioning applications. It first performs a static analysis and profiling, and then generates a weighted object relation graph (WORG), which represents the objects and relations between them. The proposed partitioning scheme takes the bandwidth as a variable to improve static partitioning and reduce the cost of dynamic partitioning.

- **Dynamic approaches**

As for the dynamic offloading approaches, the first one we consider is suggested in [24], It offloads computation to powerful servers to save energy in a wireless Java environment. It is suggested to perform compression and decompression operations simultaneously during computation offloading in order to reduce the data transmission cost. The offloading decisions are made dynamically based on the required computation and communication energy for invoking various applications. Their strategy is based on object serialization features of a Java framework where the overall optimum solution is investigated instead of obtaining the optimum for each method.

In [140], Yang *et al*. propose an architecture which supports offloading part of a mobile application seamlessly from mobile handsets to powerful servers. Their approach first gathers the resource information, and then the application is partitioned using a multi-cost graph. The graph is constructed via a profiling procedure which is a part of the offloading process. They solve the graph partitioning problem using a designed partitioning algorithm to provide a close approximation to an optimal solution.

Table 2.2: The Comparison of offloading approaches

Work	Year	Offloading decision	Profiling parameters	Core component
[87]	2001	Static	computation time and data sharing	Cost graph
[126]	2004	Static	execution cost, communication cost and run-time bookkeeping cost	Control flow graph
[139]	2007	Static	execution time and energy cost	Execution profile
[131]	2010	Static	computation and communication	Adaptive graphic rending
[28]	2011	Static	computation and migration cost	Execution profile
[104]	2014	Static	execution cost, data transmission and bandwidth	Objective relation graph
[24]	2004	Dynamic	computation and communication energy	Java serialization feature
[140]	2008	Dynamic	class weight and communication cost	Multi-cost graph
[58]	2008	Dynamic	class usage and use frequency	Execution profile
[29]	2010	Dynamic	application structuring and security	Formulation of partition
[36]	2010	Dynamic	energy, bandwidth and latency	Application profile
[41]	2015	Dynamic	execution time of service workflow and mobility of mobile devices	Genetic algorithm based
[59]	2015	Dynamic	energy consumption, expected delay and communication cost	M/G/1-queue based

An adaptable offloading approach based on execution behavior of the mobile application is proposed in [58]. Their mechanism records the consumed resources and the history of the execution pattern of the application, and later offloading decisions are made corresponding to the records. The static offloading strategy migrates most used classes to the servers, while it offloads only the invoked classes in the dynamic offloading. It is shown that their offloading scheme reduces the application execution time and the profiling overhead against simple runtime offloading.

B. Chun *et al*. [29] investigated dynamic and seamless partitioning of applications between weak device and clouds to better support execution in different environments. They formulate the dynamic partitioning problem of an application as a collection of processing modules interacting with each

other. Then the system support for dynamic partitioning is discussed including structuring applications, partition choosing and partition with security constraints. However, they do not provide any solution in their work.

The MAUI framework [36] provides automatically fine-grained code offloading to cloud servers. It creates two versions of the mobile application, one of which runs locally on the mobile device and the other runs remotely on the server, to enable software portability. The MAUI architecture includes decision engine, proxy and profiler on both the client side and server side. It makes offloading decisions dynamically at runtime to minimize the energy consumption considering various latency constraints. MAUI regards the application partitioning as a 0-1 ILP problem. However, it does not support the tradeoff analysis between energy consumption and execution time.

The work [41] proposes a dynamic offloading approach by investigating computation offloading for service workflows where multiple services are composed together. They consider mobility of mobile devices and aim to find a strategy by optimizing the execution time of the service workflow and the energy consumption of the mobile device.

The researchers in [59] consider a dynamic offloading problem in the context of MCC where the cloud servers are accessed via two channels: through a (costly) cellular connection or through intermittently available WLAN hotspots. They first build M/G/1 queueing models to represent different offloading options, local execution or offloading to a cloud. Then the dynamic decision problem is solved in the framework of Markov decision processes (MDPs), considering the availability of WLAN hotspots, energy consumption, communication costs and the expected delays.

Z. Hao *et al*. [54] presents a mobile cloud computing platform which allows users to choose to run their applications either in the cloud (for high security guarantees), or on their local mobile device (for better user experience). A scheme is proposed in [37] to enable a secure and efficient cloud-assisted image sharing architecture for mobile devices, by directly utilizing outsourced correlated images to reproduce the image of interest inside the cloud for immediate dissemination. In order to deal with the limitation of the existing group key management (GKM) protocols, Mapoka et al [93] propose the slot based multiple group key management (SMGKM) scheme for multiple multicast groups, which supports the movement of single and multiple members across a homogeneous or heterogeneous wireless network while participating in multiple group services with minimized rekeying transmission overheads.

Another work worth addressing is context-aware computing infrastructure [57] – where multiple streams of data from different sources like GPS, maps, accelerometers and temperature sensors need to be analyzed together in order to obtain real-time information about a user's context.

2.3.2 Security assessment techniques

Model analysis techniques are widely used to evaluate system performance. Ou *et al*. [105, 106] analyze the performance of offloading systems in failure-prone and fault-tolerance environments by proposing an analytical model based on application execution state transition in offloading. Their model considers the surrogate unreachability, the failure recovery time, and total execution time of applications and shows that in the areas covered by surrogates, offloading may result in speedup in the performance. In the scenario of secure group communication systems (GCSs) [27], a mathematical model based on stochastic Petri net (SPN) is proposed to investigate the performance characteristics. The model takes intrusion detection system (IDS) detection interval and rekeying interval into account to quantify the tradeoff between performance and security attributes of GCSs.

In [49] an analytical Markov model is presented to investigate the performance of service component migration from a mobile client to the infrastructure-based cloud. The proposed model utilizes a two-phased method to give a reconfiguration scheme and the gain through reconfiguration can be computed from the model. A multi-queue based stochastic model for offloading problems in MCC has been developed in [59] using various performance metrics. The costs in terms of bandwidth (the communicate overhead) and energy (computation and use of network interfaces of mobile devices) are researched in [14] by giving an evaluation model of the feasibility and cost of mobile cloud offloading.

Using quantitative methods to analyze system dependability and reliability has received great research interests for several decades. In 1993, Littelwood [90] first introduced the idea to evaluate the system security attributes using analytical methods of system reliability. Then, Nicol et al. [103] surveyed the model-based techniques for evaluating system dependability, and summarized how they can be extended to evaluate system security. However quantification of security has only recently attracted more attention. Some initial conceptual works have been published already decades ago, and a series of studies about model-based evaluation of security mechanisms have been published only recently.

In [92] the researchers attempt to quantify the system security attributes of intrusion tolerant systems through a semi-Markov process (SMP) model. They treat various system failures as absorbing states in their model and the MTTSF is computed as the time or effort to reach such absorbing states. The authors in [143] show how a key distribution center can be modeled and analyzed, and how to find an optimal key refresh rate for such a system. Previous work on the security of computing and information systems has been mostly assessed from a level point of view. A system is assigned a given security level with respect to the presence or absence of certain functional characteristics and the use of certain development techniques.

In 2013, Zhang [142] proposed an approach to evaluate the network security situation objectively using Network Security Index System (NSIS). Only a few studies have considered the quantitative evaluation of security. The authors in [86] make an effort to examine the security vulnerabilities of operating systems of routers within the cloud carrier by assessing the risk based on the National Vulnerability Database (NVD) and give quantifiable security metrics for cloud carrier, which is very useful in the Service Level Agreement (SLA) negotiation between a cloud consumer and a cloud provider. More recently a number of model-based evaluations of security mechanisms have been published [96, 101, 127].

However in the existing literatures, the question of quantitatively assessing the system security attribute is still open. In this thesis, we develop several methods based on stochastic models to quantitatively assess the security attributes of the mobile cloud offloading system. Compared to the previous work that only considers the performance and energy perspectives, the proposed approaches goes beyond the existing ones by considering the security and performance tradeoff and informed offloading decisions are made based on quantitatively assessments of system attributes.

2.3.3 Side-channel attacks

The timing attacks discussed in this thesis belong to side channel attacks. Rather than brute-force or theoretical weaknesses in the encryption algorithms, side channel attacks make use of information leakage by the physical implementation of a cryptosystem. Power channel, timing channel and electromagnetic channel are examples of side channels.

An early example of the side channel attack is the password authentication weakness discovered in the Tenex operating system [84]. Kocher was the first who observed that side channel attacks could generally be applied against various common cryptographic algorithms. Following researches extend the analysis of side channel attacks based on his foundation works on response times [71] and power consumption [70].

Side channels exist where a computer may leak its information through Radio frequency (RF) emissions [81]. Surprisingly, methods as effortless as watching the diffuse reflections of cathode ray tube (CRT) display against nearby walls may allow an observer to see the content on the screen remotely [80]. Side channels have also been used to detect passwords over Secure Shell (SSH) [120], where the researchers predict the key sequences from the inter-keystroke timings by developing a Hidden Markov Model (HMM) and a key sequence prediction algorithm.

Every logical operation in a device takes time to execute, and the time can differ depending on the inputs. So with precise analysis of each operation time, an attacker may guess the secret input successfully. This is the information leakage that a timing attacker takes advantage of. The idea of

timing attack was first suggested by Kocher in 1996 [71]. The approach proposed in [42] improves Kocher's ideas and conducts a practical implementation of timing attack against a smartcard which stores a RSA private key. Schindler [115] presented a timing attack against the implementation of RSA exponentiation which employs Chinese Remainder Theorem (CRT) theorem. He also modeled and optimized timing attacks against RSA in [116].

OpenSSL is a widely used open source crypto library which is generally seen on Apache Web Servers to provide SSL service. In [21], Brumley and Boneh demonstrate that timing attacks can reveal RSA private keys from an OpenSSL-based web server over a local network. In 2005, the researchers in [9] improved Brumley and Boneh's approach and proposed an efficient attack on RSA implementations that use CRT with the Montgomery Multiplication (MM) algorithm. They also gave suggestions to improve the decision strategy. In 2007, the researchers in [8] showed a local implementation of the timing attack on RSA that exploits branch mis-prediction delays in order to determine the secret. Bortz *et al* [18] present that timing difference may leak information such as the existence of an account or shopping cart size in web applications.

2.3.4 Defense against timing attacks

A popular strategy for defending against timing attacks is to adjust the system implementation so that the timing and cache access patterns of hardware instructions are independent of the inputs. However, this solution is architecture-specific, brittle, and difficult to get authorized [19]. So people also try to hide or mask the timing information. A series of works [13, 32, 53, 141] have proposed techniques to pad the execution to certain predetermined thresholds and formal models are also presented to describe the bound on timing channel leakage [45, 72].

To counter cache-based side-channel attacks, researchers in [44,45] present a tool called CacheAudit for the automatic, static exploration of the interactions of a program with the cache and show an approach that relies on game theory for framing and solving the decision problem, which offers the advantage of a clean interface between the security guarantees and algorithm challenges.

Interposition of random delays in the cryptographic algorithm execution flow is a simple but quite effective countermeasure against side-channel and fault attacks by mitigating the information leakage [34].

Even though Kocher [71] observed that this technique can be rendered useless by increasing the number of timing measurements, inserting random delays has the advantage of easy implementation and leading to lower system cost. It requires the attacker to take more samples in more time, which gives the system administrator more opportunities to prevent this attack. Random delays are easily deployed even if the source code of the application is not at hand. They are widely used

for protection of cryptographic implementations in embedded devices. The first detailed analysis of this kind of countermeasure showed in [31] where the number of samples for a successful differential power analysis (DPA) attack grow linearly with the standard deviation of the delay. Since then, several papers have presented implementations of random delay countermeasures in various systems [56, 77, 91].

The researchers in [91] implement random delays on FPGA and obtain the optimal parameters for delay generators. To date, based on random delay insertion, a processor architecture resistant to side-channel attacks was proposed in [56] using a combination of randomized scheduling, randomized instruction insertion and randomized pipeline-delay. Researchers in [77] present a design and hardware implementation of asynchronous AES with random noise injection for improved side-channel attack resistance.

Blinding technique is another widely deployed countermeasure against timing attacks on cryptosystems. The idea was first introduced by Kocher [71] to adapt the techniques used for blinding signatures to prevent timing attacks. It was improved by Adi Shamir [117] by choosing a new random secret for public key schemes with only negligible overhead. Although blinding techniques are often the preferred solution in practice, they are not a general remedy for timing leaks. First, blinding relies on the algebraic properties of the computed function. It works for securing RSA, but is more difficult to apply to algorithms for computing functions with a less obvious algebraic structure, such as AES or examples outside the realm of cryptography [75]. Second, potential new side-channels are introduced during the blinding (and unblinding) operations. From this perspective, blinding relocates the problem of side-channels to a different part of the implementation. Although we are not aware of a documented exploit, timing differences in the blinding steps can, in principle, be exploited by side-channel attacks.

Another countermeasure is to ensure that the implementation exhibits a constant execution time. As one can see, this countermeasure yields the provable absence of timing leaks. Its drawbacks are that it is difficult to achieve constant running times on many platforms and that the performance of a constant-time implementation may be unacceptable to pad all execution time to WCET (Worst-case Execution Time) [74]. A less restrictive way of dealing with timing leaks is to ensure that only an acceptable amount of secret information is revealed through them. We will follow this line of thought and explore the proper countermeasure against timing attacks in mobile offloading systems [71].

Researchers in [15] reason about tradeoffs between security and performance in mitigating side-channel attacks, where they concentrate on the decision between the masking countermeasure and leakage-resilient constructions. Unlike our work, they investigate which implementation has the

best performance for a fixed level of security. They consider power analysis attacks and use the best known attacks as a security benchmark, whereas we consider timing attacks and look for the optimal rekeying rate for the best security and performance tradeoff. In the area of Internet traffic masking, which obfuscate the information leaked by packet traffic features, the tradeoff between traffic privacy protection and masking cost, namely required amount of overhead and realization complexity has been studied [60]. They first propose a general model of an application flow and then optimize the metrics based on measured Internet traffic traces.

The approach proposed in this thesis goes beyond existing approaches by combining a rekeying mechanism with random padding of the processing time and considering a quantitative treatment of security problem. We aim to find a secure and cost-efficient offloading strategy for a given system configuration by optimizing the security and performance tradeoff.

2.3.5 Secure containers

A secure container is an authenticated, encrypted area of a user's mobile device designed to separate, isolate and protect enterprise data from attackers. It is a promising solution to enhance the client security in offloading scenario.

There are several container products on the market.BlackBerry's Secure Work Space [17] is an option for providing extra security for work data on iOS and Android devices. Using containerization and application wrapping, Secure Work Space separates personal information from work information by creating a personal space and a work space on devices. The Samsung company has made an effort to improve the security of its mobile devices for several years, through its Knox technology [7]. Knox is Samsung's defense-grade security platform to empower corporations to secure, manage and customize the business's mobile devices. Samsung has gotten a big boost by having its Knox-enabled devices approved by the U.S. Department of Defense (DoD) for use in DoD networks [113]. At the same time, IBM also provides a product for securing enterprise mobility. Within its MaaS360 Enterprise Mobility Management (EMM) suite, IBM MaaS360 Mobile Application Security enables an application container solution for the enterprise and third-party applications, providing operational and security management for Android, iOS and other mobile devices.

Apple does not use a separate workspace for business applications and content, but instead keeps each application in a separate sandbox and highly restricts what data can be moved between sandboxes. The APIs introduced in iOS 7 allow MDM tools to control the permissions for that data. Other iOS policies enable IT staff to manage application deployment and VPN usage on a per-application basis [5].

Andrus proposes the idea to run multiple "virtual phones" (VPs) on a mobile device [10], running each VP under its own namespace to isolate the applications and data. The idea of secure containers is put forward after that. Android for work [51] is the secure container designed by Google which utilizes the "multiple user" mechanism added in Android 5.0 to create a "work user" environment. This provides some isolation between the work and the user environment but it still allows data sharing. Knox container differs from other container solutions in that it is both software and hardware (root of trust by ARM TrustZone) based.

It is worth mentioning that Chin examines Android inter-application communication and identify security risks in the application components in [26]. [65] makes a case study of Samsung Knox container (mainly focuses on Knox 1.0) and presents a systematic assessment of security critical areas in design and implementation of the secure container. It reveals several design weaknesses of Knox 1.0. However, we consider the runtime security of the containers in this work and make an empirical approach for the quantitative evaluation of the security and performance attributes of secure container solutions.

2.4 Summary

Many offloading approaches are proposed to extend battery lifetime and to shorten execution time on the mobile device. Several researchers have worked on optimizing the tradeoff between the energy consumption and response time in mobile offloading. However, few work considers how to evaluate the secure attribute and the security performance tradeoff of offloading systems.

Part II

Security Performance Tradeoff

Chapter 3

Modelling Formalisms and Security Analysis

Mobile cloud offloading has been proposed to migrate complex computations from mobile devices to powerful servers. While this may be beneficial from the performance and energy perspective, it certainly exhibits new challenges in terms of security due to increased data transmission over networks with potentially unknown threats. Among possible security issues are timing attacks which are not prevented by traditional cryptographic security. Metrics on which offloading decisions are based must include security aspects in addition to performance and energy-efficiency. This chapter aims at quantifying the security attributes of mobile cloud offloading systems.

In this chapter, we propose a state transition model of a general mobile offloading system under the specific threat of timing attacks. Our model is aimed to deal with an offloading system with a master secret stored on the server side. In a timing attack the attacker deduces information about a secret key from runtime measurements of successive requests. From the security quantification point of view, since the sojourn time distribution function in different system states may not always be exponential, the underlying stochastic model needs to be formulated as a Semi-Markov Process (SMP).

First, we present an introduction to the stochastic modeling techniques we use. We define the terms and notation of these techniques. Then a model based method is presented for the security quantification analysis. Computing the system security and cost metric, we investigate the cost for a given security requirement. Our results will give security metrics on which offloading decisions are based.

3.1 Markov Chains and Queueing Models

In this section, we give an introduction to the modeling techniques: Markov chains and queueing models. The notation that will be used throughout this thesis is defined and important results are summarized on their properties.

3.1.1 Stochastic processes and Markov chains

A *Markov process* is a special type of *stochastic process* [121]. A *stochastic process* is a collection of random variables $\{X(t)|t \in T\}$, defined on a probability space, and indexed by a parameter t (usually assumed to be time) which can take values in a set T [55]. T is called the *index set* or *parameter space*. If the index set is discrete, then the process is called a *discrete-time stochastic process*; otherwise, if T is continuous, the process is a *continuous-time stochastic process*.

The values assumed by random variables $X(t)$ are called *states*. The set of all possible states forms the *state space* of the process and this may be discrete or continuous. If the state space is discrete, the process is referred to as a *chain* and the states are usually identified with the set or a subset of natural numbers. Without loss of generality, we assume the state space can be denoted $I = \{0, 1, 2, ...\}$.

A *Markov chain* is a Markov process with a discrete state space. A *Markov process* is a stochastic process whose conditional probability distribution function satisfies the following property: Given a stochastic process $\{X(t)|t \in T\}$, for any $t_0 < ... < t_n < t_{n+1}$ the distribution of $X(t_{n+1})$ only depends on $X(t_n)$, not on the values $X(t_0), ..., X(t_{n-1})$, i.e.,

$$Pr\{X(t_{n+1}) \leq x_{n+1} \mid X(t_n) = x_n, ..., X(t_0) = x_0\} = Pr\{X(t_{n+1}) \leq x_{n+1} \mid X(t_n) = x_n\}. \tag{3.1}$$

As can be observed, the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it. Eq. 3.1 is generally denoted as the *Markov* or *memoryless* property. Then we give the definitions of *discrete-time Markov chains* and *continuous-time Markov chains*.

Definition 3.1.1. Discrete-time Markov chain A discrete-time Markov chain (DTMC) is a discrete-time process $\{X_n|n = 0, 1, 2, ...\}$ that satisfies the Markov property: For all natural numbers n and all states x_n,

$$Pr\{X_{n+1} = x_{n+1} \mid X_n = x_n, ..., X_0 = x_0\} = Pr\{X_{n+1} = x_{n+1} \mid X_n = x_n\}.$$

Thus, the fact that the system is in state x_0 at time step 0, in state x_1 at time step 1, and so on, up

to the fact that it is in state x_{n-1} at time step $n-1$ is completely irrelevant for the next step. The state in which the system finds itself at time step $n+1$ depends only on where it is at time step n. To simplify the notation, rather than using x_i to represent the states of a Markov chain, henceforth we shall use single letters, such as i, j, and k.

The conditional probabilities $Pr\{X_{n+1} = j | X_n = i\}$ are called *transition probabilities* and denoted by $p_{ij}(n)$. Then we have the *transition probability matrix*

$$\mathbf{P}(n) = \begin{pmatrix} p_{00}(n) & p_{01}(n) & p_{02}(n) & \cdots & p_{0j}(n) & \cdots \\ p_{10}(n) & p_{11}(n) & p_{12}(n) & \cdots & p_{1j}(n) & \cdots \\ p_{20}(n) & p_{21}(n) & p_{22}(n) & \cdots & p_{2j}(n) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{i0}(n) & p_{i1}(n) & p_{i2}(n) & \cdots & p_{ij}(n) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.2}$$

P is a stochastic matrix, i.e., for all states i and j,

$$p_{ij}(n) \geq 0, \sum_j p_{ij}(n) = 1 \, .$$

A Markov chain is said to be *homogeneous* if for all states i and j

$$Pr\{X_{n+1} = j | X_n = i\} = Pr\{X_{n+m+1} = j | X_{n+m} = i\}$$

for n = 0, 1, 2, ...and $m \geq 0$.

If the parameter space T is continuous, a Markov process is called a continuous-time Markov chain (CTMC). The formal definition of CTMC is:

Definition 3.1.2. Continuous-time Markov chain We say that a stochastic process $\{X(t) | t \geq 0\}$ is a continuous- time Markov chain if for all integers (states) n, and for any sequence $t_0, t_1, ..., t_n, t_{n+1}$ such that $t_0 < t_1 < ... < t_n < t_{n+1}$,
$Pr\{X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n, ..., X(t_0) = x_0\} = Pr\{X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n\}$.

The state residence times in CTMCs are exponentially distributed [55]. Thus, we can associate with every state i in the CTMC a parameter μ_i describing the rate of the exponential distribution,

that is, we have as residence distribution in state i:

$$F_i(t) = 1 - e^{-\mu_i t}, t \geq 0. \tag{3.3}$$

Thus the vector $\mu = (\mu_1, ..., \mu_i, ...)$ describes the state residence time distributions in the CTMC. The interactions in a continuous-time Markov chain are usually specified in terms of the *rates* at which transitions occur. A continuous-time Markov chain in some state *i* at time t will move to some other state j at rate $q_{ij}(t)$ per unit time. The matrix $\mathbf{Q}(t)$, formed by placing $q_{ij}(t)$ in row i and column j, for all i and j, is called the *transition-rate matrix*. We have

$$\mathbf{Q}(t) = \begin{pmatrix} q_{00}(t) & q_{01}(t) & q_{02}(t) & \cdots & q_{0j}(t) & \cdots \\ q_{10}(t) & q_{11}(t) & q_{12}(t) & \cdots & q_{1j}(t) & \cdots \\ q_{20}(t) & q_{21}(t) & q_{22}(t) & \cdots & q_{2j}(t) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ q_{i0}(t) & q_{i1}(t) & q_{i2}(t) & \cdots & q_{ij}(t) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.4}$$

Notice that the elements of the matrix $\mathbf{Q}(t)$ satisfy the following properties:

$$q_{ij}(t) \geq 0, i \neq j,$$

and

$$-q_{ii}(t) = \sum_{j \neq i} q_{ij}(t) = \mu_i \ .$$

In this manner, a continuous-time Markov chain is represented by its matrix of transition rates, $Q(t)$, at time t.

Transient distribution analysis of a CTMC Let the probability that the system is in state i at time t be $\pi_i(t)$, i.e.,

$$\pi_i(t) = Pr\{X(t) = i\}, \tag{3.5}$$

and $\pi(t) = \{\pi_i(t), i \in I\}$. The transient distribution of a CTMC can be computed by solving

$$\frac{\mathrm{d}\pi(t)}{\mathrm{d}t} = \pi(t)\mathbf{Q}(t). \tag{3.6}$$

Steady-state distribution analysis of a CTMC If the CTMC arrives a point in time at which the rate of *change* of the probability distribution vector $\pi(t)$ is zero, then the left-hand side of Eq. 3.6 is identically equal to zero. In this case, we call the system is in a steady-state. The steady-state distribution is written simply as $\pi = \{\pi_i, i \in\}$ in order to show that it no longer depends on time t. π can be computed by solving the system of linear equations

$$\pi \mathbf{Q} = 0. \tag{3.7}$$

Semi-Markov Processes (SMPs) are generalizations of Markov chains in that the future evolution of the process is independent of the sequence of states visited prior to the current state and independent of the time spent in each of the previously visited states, as is the case for discrete- and continuous-time Markov chains. Semi-Markov processes differ from Markov chains in that the probability distribution of the remaining time in any state can depend on the length of time the process has already spent in that state.

A semi-Markov process consists of two components: (i) a discrete-time Markov chain $\{X_n | n = 0, 1, 2, ...\}$ with transition probability matrix **P** which describes the sequence of states visited by the process, and (ii) $H_{ij}(t)$, the conditional distribution function of a random variable T_{ij} which describes the time spent in state i from the moment the process last entered that state

$$H_{ij}(t) = Pr\{T_{ij} \leq t\} = Pr\{\tau_{n+1} - \tau_n \leq t | X_{n+1} = j, X_n = i\}, t \geq 0.$$

The random variable T_{ij} is the sojourn time in state i per visit to state i prior to jumping to state j. The evolution of a semi-Markov process is as follows:

1) The moment the semi-Markov process enters any state i, it randomly selects the next state to visit j according to **P**, its transition probability matrix.
2) If state j is selected, then the time the process remains in state i before moving to state j is a random variable T_{ij} with probability distribution $H_{ij}(t)$.

Thus the next state to visit is chosen first and the time to be spent in state i chosen second, which allows the sojourn time per visit to depend on the destination state as well as the source state. The **steady-state probabilities** $\{\pi_i, i \in I\}$ of the SMP states can be computed in terms of the embedded DTMC steady-state probabilities v_i and the mean sojourn times h_i [124]:

$$\pi_i = \frac{v_i h_i}{\sum_j v_j h_j} \quad i, j \in X_s. \tag{3.8}$$

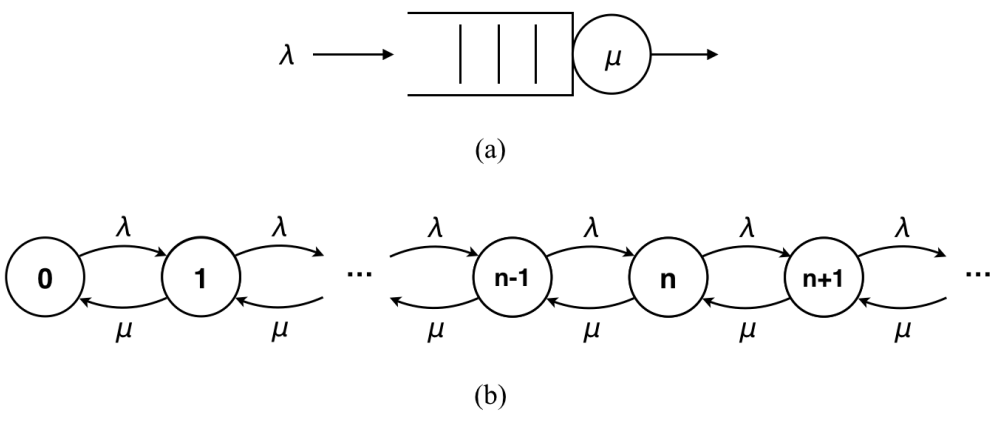


Figure 3.1: (a) The $M/M/1$ queue and (b) its state transition diagram

3.1.2 Elementary queueing theory

The simplest of all queueing systems is the $M/M/1$ queue, that is a single server queue with first-come, first-served (FCFS) scheduling discipline, a Poisson arrival process and service time that is exponentially distributed. As shown in Figure 3.1(a), λ is the parameter of the Poisson arrival process and μ is the exponential service rate. The mean time between arrivals (which is exponentially distributed) is $1/\lambda$ and the mean service time is $1/\mu$.

The underlying Markov chain of the $M/M/1$ queue is a birth-death process with the state transition diagram shown in Figure 3.1(b). *Birth–death processes* are continuous-time Markov chains with a special structure. If the states of the Markov chain are indexed by the integers $0, 1, 2, ...$, then transitions are permitted only from state $i > 0$ to its nearest neighbors, namely, states $i-1$ and $i+1$. As for state $i = 0$, on exiting this state, the Markov chain must enter state 1. Here we introduce some basic performance measures of the $M/M/1$ queue.

Utilization In a $M/M/1$ queue, the *utilization* ρ is defined as the fraction of time that the server is busy and we have $\rho = \lambda/\mu$.

Number of customers Let N be the random variable that describes the number of customers in the system and p_n the probability that there are n customers in the system, $p_n = Pr\{N = n\}$. Then the average number of customers in the system is

$$E[N] = \sum_{i=0}^{\infty} np_n. \tag{3.9}$$

Let $E[N_s]$ be the average numbers of jobs in the server and $E[N_q]$ be the average numbers of jobs in the queue. For a overall queueing system, the average number of jobs in the queueing system $E[N]$ must be equal to the sum of the average numbers of jobs in the components of the queueing system, i.e., $E[N] = E[N_q] + E[N_s]$. Given the system utilization of a $M/M/1$ queue, we have

$$E[N] = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}, \tag{3.10}$$

and the average number of customers in the queue

$$E[N_q] = \frac{\rho^2}{1-\rho} = \rho E[N]. \tag{3.11}$$

System time and queueing time The time that a customer spends in the system, from the instant of its arrival to the queue to the instant of its departure from the server, is called the *response time* or *sojourn time*. We shall denote the random variable that describes response time by R, and its mean value by $E[R]$. The response time is composed of the time that the customer spends waiting in the queue, called the *waiting time*, plus the time the customer spends receiving service, called the *service time*. We shall let W_q be the random variable that describes the time the customer spends waiting in the queue and its mean will be denoted by $E[W_q]$. Given the system utilization of a $M/M/1$ queue, we have

$$E[R] = \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda}. \tag{3.12}$$

System Throughput The throughput of a queueing system is equal to its departure rate, i.e., the average number of customers that are processed per unit time. It is denoted by X. In a stable queueing system in which all customers that arrive are eventually served and leave the system, the throughput is equal to the arrival rate, λ. This is not the case in queueing systems with finite buffer, since arrivals may be lost before receiving service.

In the end of this section, we introduce the *most general* law in model-based performance evaluation: *Little's law*, named after the author who first proved it [89]. Little's law relates the average number of jobs in a queue to the average number of arrivals per time unit and the average time a job spends in a queue.

Little's law: The average number of customers in the system is equal to the average arrival rate of customer to the system multiplied by the average system time per customer,

$$E[N] = \lambda E[R]. \tag{3.13}$$

3.2 Metrics

Before proposing models for the mobile cloud offloading system, we establish the metrics we want to investigate. We present security and performance metrics respectively in this section. In addition to analyzing the metrics independently, the tradeoff between different metrics is also addressed.

3.2.1 Security Metrics

The security metrics are defined in this work as confidentiality Λ and system (security) cost C. In information security, confidentiality is defined as the property that information is not made available or disclosed to unauthorized individuals, entities, or processes (Excerpt ISO27000 [62]). If a timing attack to the offloading system is successful, the attacker will obtains the server's private key. It can browse unauthorized files thereafter and do more harm. This denote the loss of confidentiality. So the confidentiality metric Λ is defined as the probability that the sensitive information of the offloading system is not disclosed.

$$\Lambda = Pr\{\text{information is not disclosed}\} \ . \tag{3.14}$$

In our scenario, the offloading system suffers from cost in two cases: the system loses sensitive information in the compromised state, and cost is also incurred when the system deploys a rekeying process regularly. So the cost metric C is defined as the sum of these two costs.

$$C = C_{rekeying} + C_{disclosure} \ . \tag{3.15}$$

3.2.2 Performance Metrics

The performance metrics we are interested in describe the system in terms of its throughput, completion times, or response times, as defined e.g. in queueing theory or networking. Here we use the throughput as the performance metric for the offloading system. By Little's Law, the throughput (denoted X) is defined as:

$$X = \frac{E[N]}{E[R]} \ . \tag{3.16}$$

The throughput equals the average number of jobs in the queue ($E[N]$) divided by the average time a job spends in the queue ($E[R]$).

3.2.3 Tradeoff Metric

In order to investigate how system security will interact with performance, we also define a tradeoff metric. The scurity-performance tradeoff metric we propose is an objective function formed from the product of the security attribute confidentiality and the system throughput. This can be seen as **Security per time** metric. As a system designer, one may look forward to maintaining the confidentiality of sensitive information with higher throughput, as for the tradeoff measure, the larger the better.

$$\Phi = \Lambda \times X \,. \tag{3.17}$$

The security and performance metrics defined here will be used to evaluate the system attributes in the rest of this thesis.

3.3 Model based Security Analysis

A mobile cloud offloading system is a solution to enhance the capabilities of the mobile system by migrating computation to more resourceful computers (i.e., servers). To quantitatively analyse the security attributes of a system under the threat of timing attacks, we have to incorporate the actions of an attacker who is trying to capture sensitive information in conjunction with the protective actions taken by the offloading system.

Therefore, we have to develop a composite security model that takes into account the behaviour of both actors. First, we propose a SMP model for the security attribute of the offloading system. Semi-Markov Processes are generalizations of Markov chains where the sojourn times in the states need not be exponentially distributed [88].

3.3.1 Offloading under timing attacks

We specify the behaviour of the mobile cloud offloading system under timing attacks and the attacker in our scenario in this section.

We consider an environment where there is a remote cloud server for executing mobile application jobs. An offloading approach is implemented in this environment to perform computation and data offloading [119]. The computation- or energy-intensive jobs generated by the mobile applications can be either executed locally on the mobile device or offloaded to the cloud servers to save execution time and energy on the mobile devices. Real-time multimedia applications, especially real-time strategy game, fitness application are some applications that can benefit from this approach. For

instance, in mobile chess games, the searching job for the next best move needs a lot of computation resources, so this job is offloaded to cloud servers. The mobile device only needs to send the current arrangement of all figures on the board, which is a small amount of data, and receives the search results after the jobs are executed in the cloud.

In the mobile cloud offloading system we consider, mobile clients are connected to a mobile network via base stations and WiFi access points that establish and control the connections (air links) and functional interfaces between the networks and mobile devices. The mobile users' requests are delivered to the cloud service providers (e.g. Amazon Elastic Compute Cloud EC2 and Simple Storage Service S3) through the Internet. In the cloud, cloud controllers process the requests to provide mobile users with the corresponding cloud services [43].

A master key stored in the cloud server is used for the encryption and decryption operations of all user data. The offloading system is assumed to be vulnerable to timing at tacks in which the attacker in the worst case will eventually decrypt the system's private key saved in the server. In timing attacks to the offloading system, an attacker continues to send jobs to the cloud server. In addition the attacker records each response time for a certain service and tries to find clues to the master secret of the server by comparing time differences from several request queues. If the attacker successfully breaks the secret information from the timing results, he may enter the system, read and even modify other users' information without further authorization. In order to improve security, the server regularly or irregularly changes the master key, which is called the rekeying process.

3.3.2 System lifetime analysis

After initialization, the system starts to operate properly in the good state. A normal client and the cloud server use a three-way handshake to establish a connection (Figure 3.2). After a full-duplex communication is established, the offloading jobs generated by the mobile applications are offloaded to the cloud server. Then the mobile device receives the computation results sent back by the server.

The mobile offloading system is under the specific threat of timing attacks conducted by random attackers. When an attacker starts to conduct timing attacks to the cloud server, the system is in danger (Figure 3.3). We assume the attacker conducts a Brumley's attack discussed in Section 2.2.1. In this state, the attacker is still conducting the timing attack to guess the server key and it is not yet able to access confidential information.

If the attacker succeeds to determine the encryption key through time measurements, confidential data will be disclosed which is assumed to incur a high cost. This can only happen if the system is in the compromised state (Figure 3.4) and we call the incident of entering the compromised state a *security failure*.

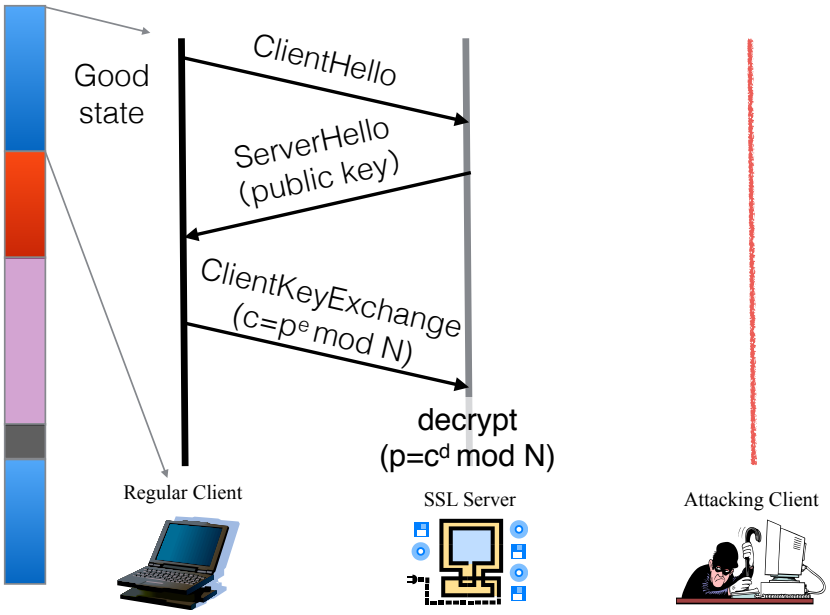


Figure 3.2: Illustration of the offloading system after initialization

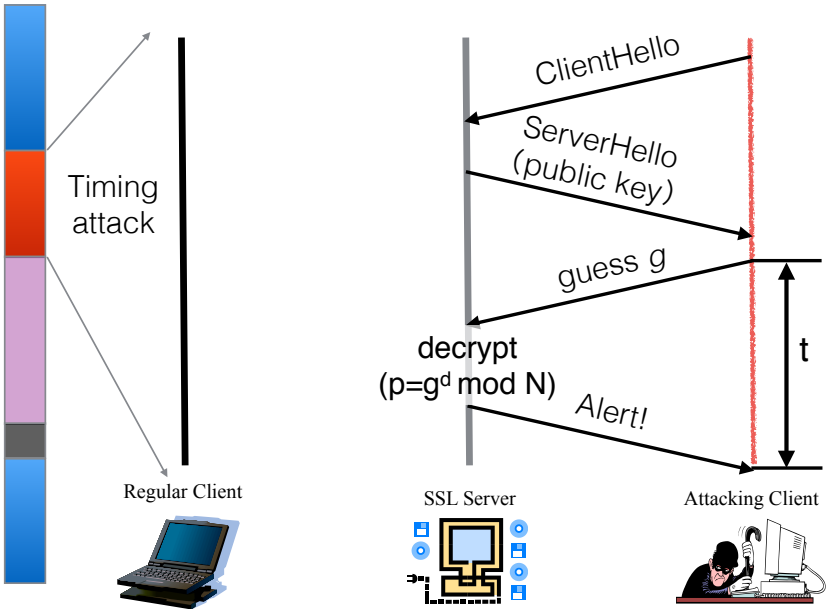


Figure 3.3: Illustration of the offloading system when an attacker is conducting timing attacks

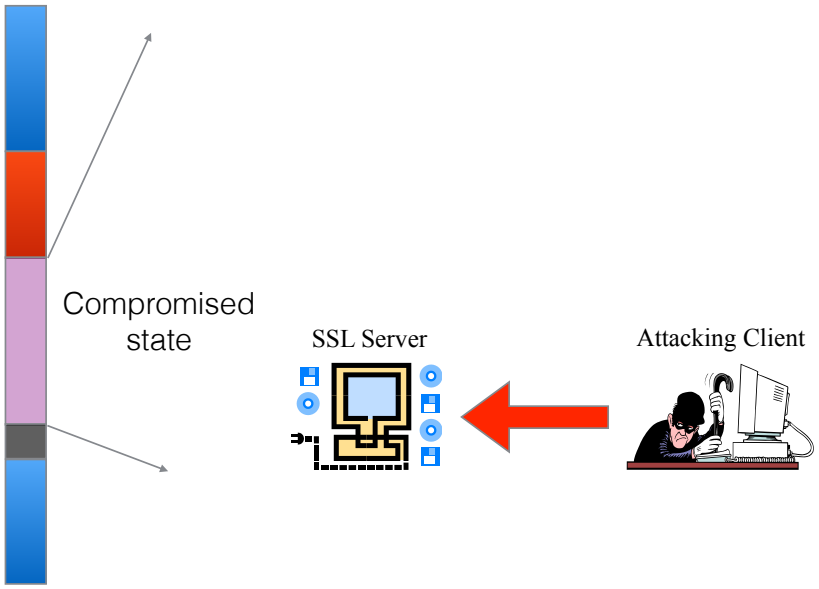


Figure 3.4: Illustration of the offloading system in the compromised state

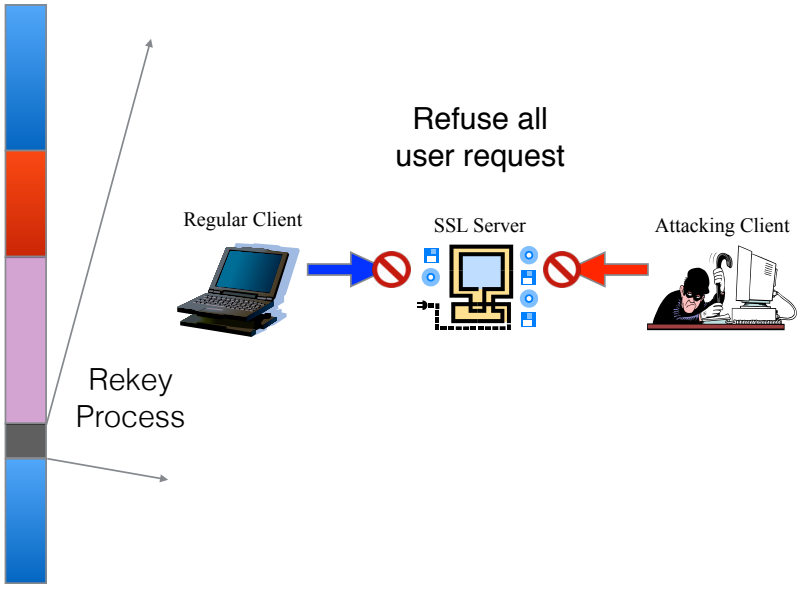


Figure 3.5: Illustration of the offloading system in the rekeying state R

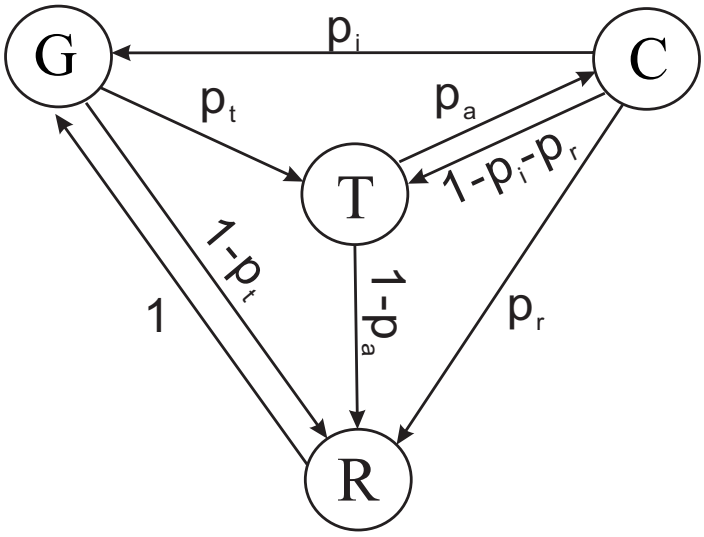


Figure 3.6: State transition diagram for a generic offloading system

We assume that the security policy of the server system requires to launch a rekeying process regularly to avoid timing attacks. Renewing the server encryption key can prevent or interrupt a timing attack. However this rekeying process is not free. It brings cost to the system. When the system such as Cisco MDS Family Storage Media Encryption is used in the server side, tape volume groups should be rekeyed periodically to ensure better security and also when the key security has been compromised [30]. The system has to process all user-files with both the new and the old master key. In this process, the system does not accept any other user commands. When user data is very large, this process will take long. Therefore, it is reasonable to recommend an optimal rekeying interval for the master key replacement cycle (or an optimal rekeying rate), and select a suitable time, when there is a low number of user accesses (e.g. at night). After the rekeying process, the system is secure and it is brought back to the initial state. The life cycle of the system starts again.

3.3.3 Security Model

From the real system life time, we abstract four operational states for the mobile cloud offloading system. Figure 3.6 depicts the SMP model we propose for describing the dynamic behaviour of a generic mobile cloud offloading system. This system is under the specific threat of timing attacks. The state transition model represents the system behaviour for a specific attack and given system configuration that depends on the actual security requirements. We describe the events that trigger transitions among states in terms of probabilities and cumulative distribution functions, which will

be shown later.

The states and parameters of the SMP model are summarized here:

- G Good state in which the offloading system works properly
- T Timing attack state in which an attacker is conducting timing attacks
- C Compromised state after the attacker knows the secret of the system
- R Rekeying state in which the system renews its master secret
- p_t probability that an attacker beginS to conduct a timing attack to the system
- p_a probability of system confidentiality lost
- p_i probability that the system returns to initial state by manual intervention
- p_r probability that the attack is terminated due to rekeying operation

After initialisation, the system is in the good state G. The sojourn time in state G is the life time of the system before an attacker starts a timing attack or the system renews its key. We assume there is only one attacker in the system at one time. If an attack happens, the system is brought to state T, in which the timing attack takes place and the attacker decyphers the encryption key by making time observations. So while the system is in state T, the attacker is not yet able to access confidential information.

It takes a certain amount of time to perform the timing attack after which the attacker will know the encryption key and the system moves to the compromised state C. Renewing the encryption key can prevent or interrupt a timing attack. During rekeying the system is in state R. The challenge is to find an optimal value for the rekey interval. The rekeying should certainly happen before or soon after the system enters the compromised state. Rekeying will bring the system back to the initial state G.

If the attacker succeeds to determine the encryption key through time measurements, confidential data will be disclosed which is assumed to incur a high cost. In this state, one possibility is that one attacker stops the attack and another attacker comes for a new timing attack. So the system is brought from compromised state C to another timing attack state T. The attack can also be stopped by manual intervention, i.e. closure of the session when finding the intrusion behaviour. The system may also trigger the rekeying process regularly, this can happen either in the attack state T or in the compromised state C, both transitioning the system to the rekey state R from which it will finally return to the initial state.

3.4 Semi-Markov Process analysis

For the offloading system, we have described the system's dynamic behaviour by a SMP model in Figure. 3.6 with four states $\{G, T, C, R\}$ and the transition between these states. A system response to a security attack is fairly automated and could be quite similar to how it may respond to accidental faults. Let $\{X(t) : t \geq 0\}$ be the underlying stochastic process with a discrete state space $X_s = \{G, T, C, R\}$. To obtain a complete description of this SMP model, two sets of parameters must be known: the mean sojourn time h_i in each state and the transition probabilities p_{ij} between different states, where $i, j \in X_s$, which we have depicted in the previous Section. The mean sojourn time in each state are summarized here:

- h_G the mean time the system spends before an attacker conducts a timing attack or rekey itself
- h_T the mean time before the attacker break the master secret of the server by timing attack
- h_C the mean time the system is in the compromised state
- h_R the mean time for rekeying process

In order to carry out the security quantification analysis, we need to analyse the SMP model of the system that was described by its state transition diagram. As described in Eq. 3.8, the steady-state probabilities $\{\pi_i, i \in X_s\}$ of the SMP states are computed in terms of the embedded DTMC steady-state probabilities v_i and the mean sojourn times h_i.

3.4.1 DTMC steady-state analysis

In order to distinguish from the steady-state probabilities of the SMP states, we let $\overrightarrow{v} = \{v_i, i \in X_s\}$ be the steady-state probability vector of the underlying DTMC. Assuming the existence of the steady-state in the underlying DTMC, it can be computed as

$$\overrightarrow{v} = \overrightarrow{v} \cdot \mathbf{P} \quad i \in X_s. \tag{3.18}$$

where $\overrightarrow{v} = [v_G, v_T, v_C, v_R]$ and **P** is the DTMC transition probability matrix which can be written as:

$$\mathbf{P} = \begin{array}{c} \\ G \\ T \\ C \\ R \end{array} \begin{array}{c} \begin{array}{cccc} G & T & C & R \end{array} \\ \begin{pmatrix} 0 & p_t & 0 & 1-p_t \\ 0 & 0 & p_a & 1-p_a \\ p_i & 1-p_i-p_r & 0 & p_r \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{3.19}$$

In addition, we have the total probability relationship:

$$\sum_i v_i = 1 \quad i \in X_s. \tag{3.20}$$

The transition probability matrix **P** describes the DTMC state transition probabilities between the DTMC states as shown in Figure 3.6. The first step towards evaluating security attributes is to find the steady-state probability vector $\overrightarrow{v}$ of the DTMC states by solving Eqs. 3.18 and 3.20. We can get solutions:

$$\begin{aligned} v_G &= \frac{p_i p_a + 1 - p_a + p_a p_r}{\phi}, \\ v_T &= \frac{p_t}{\phi},\ v_C = \frac{p_t p_a}{\phi},\ v_R = v_G - \frac{p_i p_t p_a}{\phi} \end{aligned} \tag{3.21}$$

For the sake of brevity, we assume: $\phi = 2 + 2p_i p_a + p_t + p_t p_a - 2p_a + p_a p_r - p_i p_t p_a$.

In the next subsection, the DTMC steady-state probabilities are used to compute the SMP steady-state probabilities.

3.4.2 Semi-Markov model analysis

The mean sojourn time h_i in a particular state $i \in X_s$ is the other quantity that is needed to compute the SMP steady-state probabilities. It is determined by the random time that a process spends in a particular state.

The parameters h_T, h_C, p_t, p_a depend on the attackers' behavior which we model as random processes. The analysis in this chapter only takes into account the mean value of these processes. More complex study will consider a quantitative analysis of attacker behavior based on empirical data. However, this chapter is limited to dealing with an SMP model only.

Clearly, for the model to be accurate, it is important to estimate accurately the model parameters. Some parameters we will get from experiments. The measurements we are in process of taking are

based on an offloading server under timing attacks. We have built a timing attack demonstrator and metric the mean time for a successful attack which will be used as h_G. Some parameters, e.g. the probability that an attacker begins to conduct a timing attack and attacks system confidentiality after a successful timing attack will be assumed as an attacker. Other parameters used in our system can be tuned by the system administrator, like the rekeying probability p_r and the mean sojourn time in the initial state h_G. In this work, however, our focus is primarily on developing a quantitative analysis methodology for the security attributes of an offloading system. So, in the absence of exact values of model parameters, we assume it will also be meaningful to evaluate the sensitivity of security attributes to variations in model parameters.

In Section 3.5 we present a case study with numerical results to show how one can use our quantitative analysis of system security and the influences of changes in the various model parameters. Here, we can compute the steady-state probabilities $\{\pi_i, i \in X_s\}$ of the SMP states by using Eqs. 3.8 and 3.21. Again, for the sake of brevity, we assume:
$\Phi = (p_i p_a + 1 - p_a + p_a p_r)h_G + p_t h_T + p_t p_a h_C + (p_i p_a + 1 - p_a + p_a p_r - p_i p_t p_a)h_R$. The solutions are presented as

$$\pi_G = \frac{p_i p_a + 1 - p_a + p_a p_r}{\Phi} h_G \tag{3.22}$$

$$\pi_T = \frac{p_t}{\Phi} h_T \tag{3.23}$$

$$\pi_C = \frac{p_t p_a}{\Phi} h_C \tag{3.24}$$

$$\pi_R = \frac{h_R}{h_G}\pi_G - \frac{p_i p_t p_a}{\Phi} h_R \tag{3.25}$$

3.4.3 Computing Security

We have defined the security metrics confidentiality Λ and system (security) cost C in Section 3.2.1. We compute the security metrics as functions of the state probabilities of the SMP model in this section.

From the system lifetime analysis, one can see that the offloading system's confidential data will be disclosed only in the compromised state C. Therefore, the steady-state confidentiality metric can then be computed as

$$\Lambda = 1 - \pi_C \ . \tag{3.26}$$

The offloading system suffers from cost in two states, the compromised state C and the rekeying

state R. The system loses sensitive information in the compromised state, and cost is also incurred when the system deploys a rekeying process regularly. The steady-state probabilities π_i may be interpreted as the proportion of time that the SMP spends in the state i. In the SMP model, the rekeying effort cost $C_{rekeying}$ and the data disclosure cost $C_{disclosure}$ are both interpreted as the proportion of system life time, that is, the steady-state probability of the SMP. In order to share relative importance between the loss of sensitive information and the effort needed to rekey regularly, we define two weights w and its complement $1 - w$ for the two kinds of cost. We use normalization weights for simplicity. The system cost can be computed as:

$$\begin{aligned} C &= C_{rekeying} + C_{disclosure} \\ &= (1-w)\pi_R + w\pi_C \ , \end{aligned} \tag{3.27}$$

where $\pi_i, i \in \{C, R\}$ denotes the steady-state probability that the SMP is in state i. $0 \leq w \leq 1$ is the weighting parameter.

In order to investigate how system security will interact with the cost, we also compute a *Security per dollar* metric. An objective function formed from the division of the security attribute confidentiality and system cost is created to demonstrate the relationship between the cost the system has to pay and the corresponding security system gain. This metric shows the how much security per cost one can obtain. As a system designer, one may look forward to maintaining the confidentiality of sensitive information with lower system cost, as for the metric T, the larger the better.

$$T = \frac{\Lambda}{C} \ . \tag{3.28}$$

Given the steady-state probabilities, the system throughput can be written as:

$$\begin{aligned} X = &\frac{\lambda\lambda_4\lambda_5(\lambda_1+\lambda_2+\lambda_3+\lambda_5)}{\phi} + \frac{\lambda'\lambda_4[(\lambda_1+\lambda_2+\lambda_3+\lambda_6)\lambda_5+\lambda_2\lambda_3]}{\phi} \\ &+ \frac{\mu'[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi} \ . \end{aligned} \tag{3.29}$$

3.4.4 Sensitivity analysis

The main aim of parametric sensitivity analysis is to predict the effect of variations in inputs and parameters on outputs (metrics), hoping to find performance or reliability bottlenecks, and guiding an optimisation process [48]. It is a useful procedure for offloading system optimisation in the early design phase. Since some model parameters are difficult to ascertain in the design phase,

sensitivity analysis can predict the influence on the quantitative analysis results from changes in different parameters.

$Metric \in \{C, \Lambda, T\}$ is a metric. $x \in \{p_i, p_t, p_a, p_r, h_G, h_T, h_C, h_R\}$ is a variable in our model. The sensitivity analysis is conducted by calculating the derivative of the metric with respect to a certain input parameter.

$$\frac{d(Metric)}{dx} \tag{3.30}$$

Eq. 3.30 is the sensitivity formula for metric prediction in the SMP model. The numerical results in the format of graphs will be shown in the next section, from which we can see intuitively the impact of parameter changes on different metrics.

3.5 Numerical Study

In this section we give numerical results as examples to show how one can evaluate security attributes of the SMP model defined in the previous sections using different metrics.

We use the system parameters we propose in [96]. We assume that the probability of a timing attack coming to the offloading system is equal to the one that the system will trigger its rekeying process, i.e., p_t = 0.5. The mean time the system spends before an attacker conducts a timing attack or it rekeys is h_G = 10 time units. Further, the probability that the attacker successfully cracks the system secret using a timing attack is p_a = 0.6 and the probability of an unsuccessful attack $1 - p_a$ = 0.4. The time taken by a successful timing attack is assumed to be h_T = 5 time units. Besides, suppose that the probability that the system returns to the initial state by manual intervention is p_i = 0.2 and probability of the attack is terminated due to rekeying operation is p_r = 0.5. Hence, the probability that the current attack stops and another timing attack affects the system is $1 - p_i - p_r$ = 0.3 . We also assume the duration for a specific attack is supposed to be h_C = 3 time unites and rekeying time is h_R = 1 time unite respectively.

Using the values given above as the model input parameters and Eqs. 3.22 - 3.25, we obtain the steady-state probabilities of the Semi-Markov process as:

π_G = 0.6634, π_T = 0.2023, π_C = 0.0728, π_R = 0.0615.

The steady-state probabilities π_i may be interpreted as the proportion of time that the SMP spends in state i. For the assumed values of the input parameters, the proportion of time that the offloading system spends in the initial state G is approximately 66% of the whole system life time.

Figure 3.7 shows the metric of system cost C, confidentiality Λ and *Security per dollar* metric T changing with different weighting parameters w. To better scale for the figure, the *Security per dollar* metric T is divided by 15 and the cost metric C is multiplied by 10. From Figure 3.7, it can

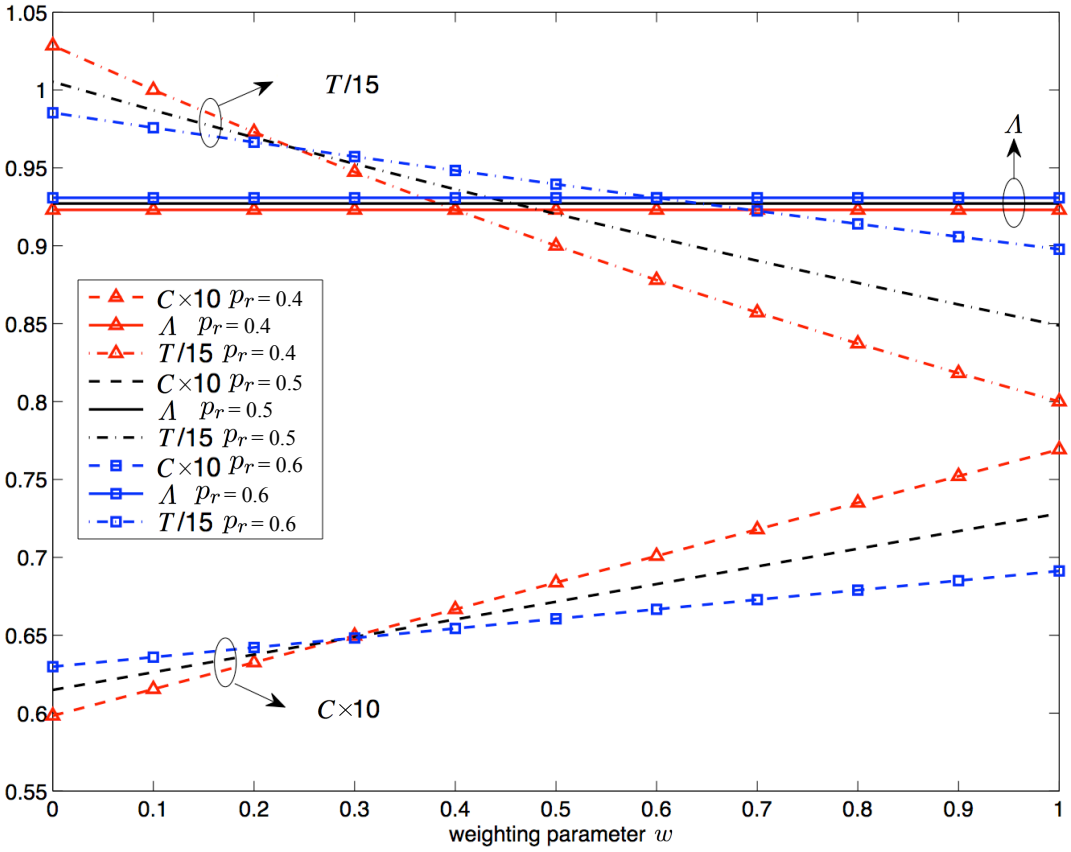


Figure 3.7: System metrics changing with weighting parameter w under different p_r

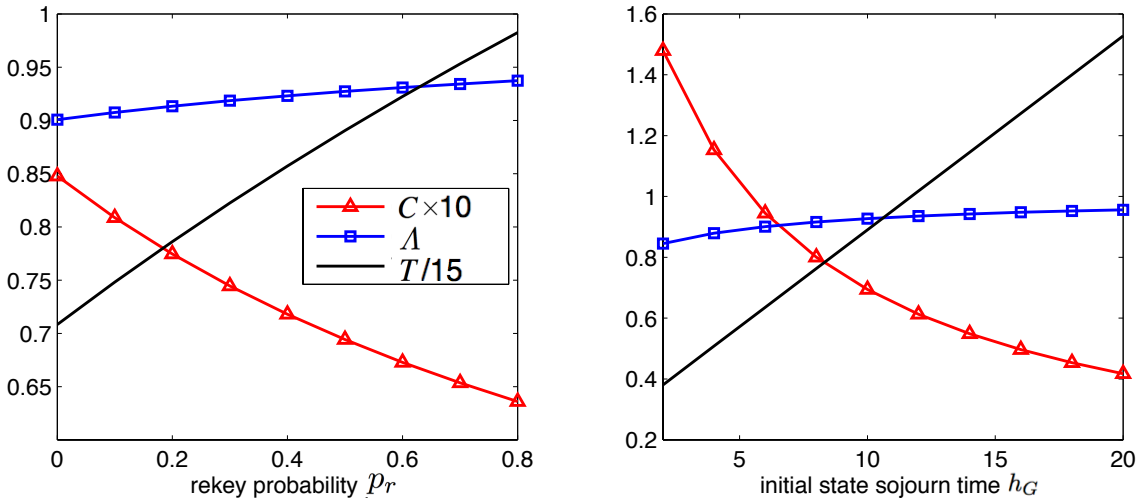
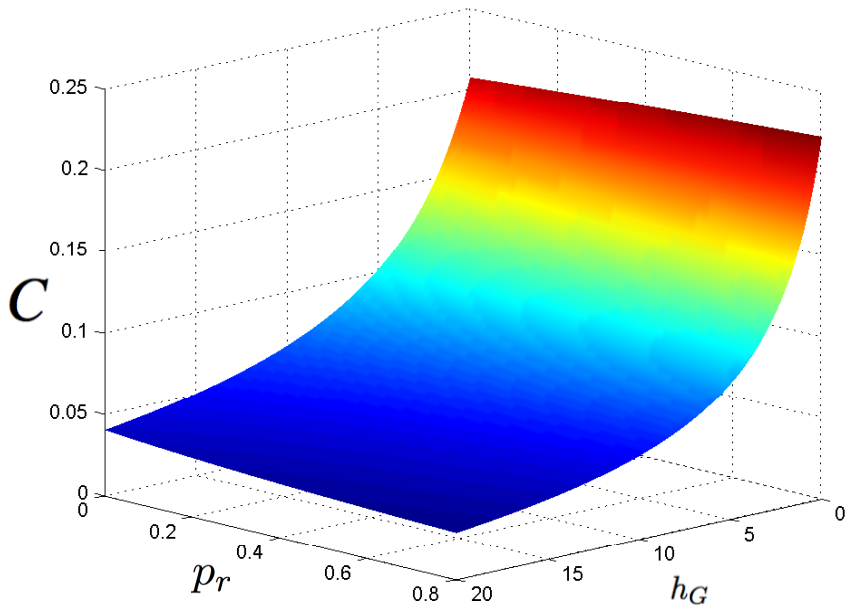


Figure 3.8: System metric comparison under p_r and h_G

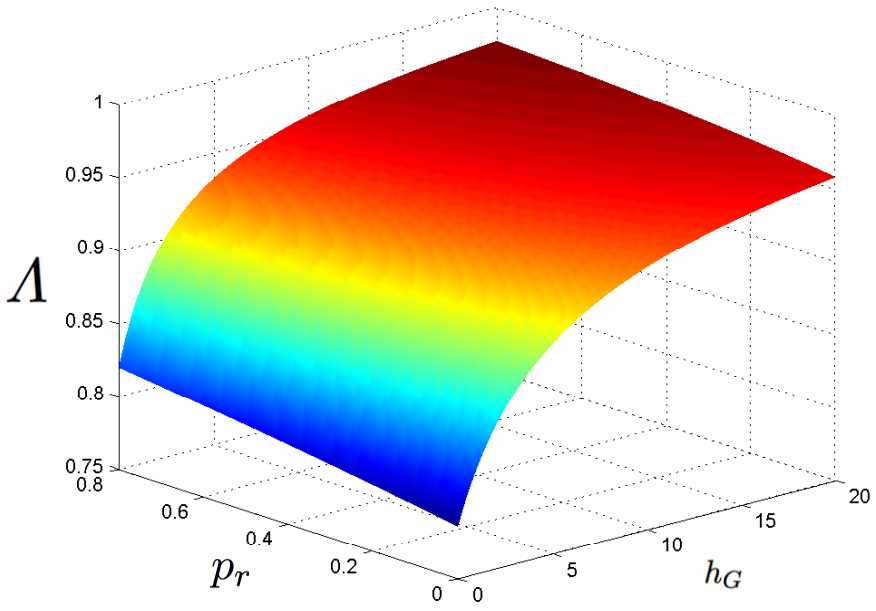
be seen that the system cost increases with rekeying probability p_r when the weighting parameter is small, as we put more weight on the rekeying effort cost. At high values of w, the information loss cost component becomes the decisive factor. We see the decrease in system cost C as p_r increases. While one can see that the cost increases with the rekeying probability p_r, the confidentiality Λ stays constant with changing weighting parameters w as Λ is independent of the weighting. The *Security per dollar* metric increases as the rekeying probability p_r increases when the parameter w is small. We found that as we tune the weighting parameter w, the effect of increasing the rekeying probability p_r is different for different metrics.

Figure 3.8 shows how different metrics behave with changes in rekeying probability parameter p_r and the time in the initial state h_G. It can be seen that the system cost C decreases with both p_r and h_G. At the same time, the *Security per dollar* metric T increase with growing p_r and h_G. The larger p_r and h_G are, the better the offloading system performs. So in the offloading system, one can increase the rekeying probability and the sojourn time in the initial state to improve the security attribute. But in the SMP model, we cannot represent the rekeying rate with a single model parameter. We will improve the model in the next Chapter.

As discussed in Section 3.3.3, p_r is the probability that the attack is terminated due to rekeying operation, so it depends on the system configuration. The rekeying process can bring the system back to the initial state before an attack, which will affect h_G. Since we can tune the trigger probability of the rekeying process as system administrators, we conduct sensitivity analysis of system behaviour on the effect from changes in the rekeying probability p_r and the mean sojourn time in the initial



(a) C as a function of h_G and p_r



(b) Λ as a function of h_G and p_r

Figure 3.9: Sensitivity analysis to C and Λ

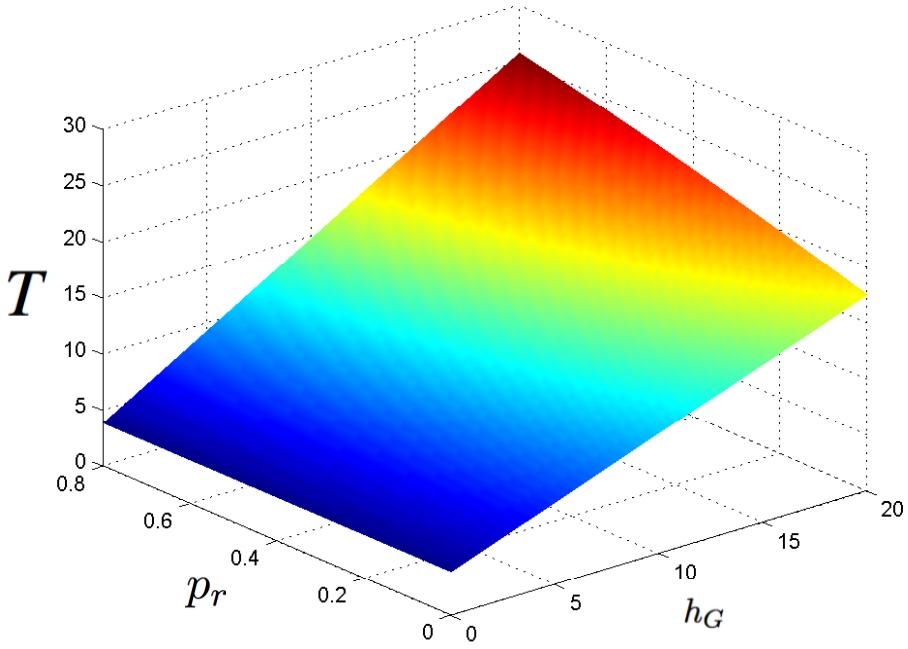


Figure 3.10: *Security per dollar* metric as a function of h_G and p_r

state h_G. Figure 3.9(a) shows the system cost C as a function of h_G and p_r. Interestingly, when the mean time in the initial state h_G is short, the system cost increases as the rekeying probability p_r increases. However, we see a decrease in system cost as p_r increases, when h_G is very long. Also we can see, the system cost is more sensitive to h_G than to p_r. In Figure 3.9(b), we conduct sensitivity analysis to the system confidentiality metric Λ. It increases dramatically with the sojourn time h_G in state G when the system is rekeying more frequently. However, it does not interact that strikingly with model input parameter p_r.

The *Security per dollar* metric as a function of h_G and p_r is depicted in Figure 3.10. As expected, the metric T monotonically increases as p_r and h_G increase. That is because the system more often rekeys and it spends more time in good state. When the time in the initial state is short, the metric T hardly changes with the parameter p_r. However, the rekeying probability has a significant effect on the system as h_G is very large.

3.6 Summary

In this chapter, we have presented an introduction to the modeling techniques and an approach for quantitative assessment of security attributes for an offloading system under the specific threat of timing attacks. A state transition model that describes the dynamic behavior of this system is used as the basics for developing a stochastic model. We have solved for steady-state probabilities of the Semi-Markov Process model as the foundation of security attribute analysis. These include system cost and a *Security per dollar* metric. Also, the model analysis is illustrated in a numerical example. The sensitivity of the influencing factors in the quantitative analysis is also discussed.

Chapter 4

Security and Performance Tradeoff Analysis

With growing interest in the combined analysis of performance and security, stochastic models have been widely applied to conduct quantitative evaluations. In this chapter, we improve our proposed security model to represent the system rekeying rate with a transition rate of a CTMC model (the *Security Model*). In oder to take the performance property into account, we extend the model with an open queueing model (the *Performance Model*), which is proposed to exhibit the offloading decision and job processing operation. A job is either processed locally by the mobile device or offloaded and served by cloud servers.

In the presence of timing attacks, we investigate how to set system parameters to obtain the optimum tradeoff between security and performance in mobile cloud computing systems. This chapter aims at quantifying the security attributes and their impact on the performance of mobile offloading systems. The offloading system is modeled as a hybrid CTMC and queueing model. The proposed model focuses on state transition and state-based control. The quantification analysis is carried out for steady-state behavior of the CTMC model as to optimize the tradeoff metric. By transforming the security model to a model with absorbing state, we compute the "mean time to security failure" (MTTSF) measure.

4.1 Tradeoff analysis of Mobile Cloud Offloading

In order to proceed to a quantitative treatment of the performance-security tradeoff of mobile cloud offloading systems we propose a hybrid CTMC and queueing model which treats the security

and performance attributes respectively.

We show how to formulate measures that include both, performance and security aspects and that optimize the tradeoff between the two. Our model deals with a general offloading system with a private key (e.g. a Tape volume key) stored on the server side, where random timing attacks try to guess the key. Frequently renewing the server private key can prevent or interrupt a timing attack. However this rekeying process brings cost to the system as it requires processing effort. So the system performance will be affected. We try to investigate the tradeoff between the security we gain and the performance degradation. By solving the proposed model, we propose different metrics on which offloading decisions can be based.

4.2 The Hybrid Model

We develop a hybrid CTMC and queueing model that takes into account the behavior of both the system and the attacker, to proceed to a quantitative assessment of the performance and security attributes of the mobile cloud offloading system under the threat of timing attacks.

Figure 4.1 depicts the diagram of a performance and security model formulated of a CTMC and an open queueing model for describing dynamic behavior of a mobile offloading system. As compared to the model proposed in the last chapter, which only considers the security attributes of offloading systems, the proposed hybrid CTMC and queueing model here takes the performance properties of a generic offloading system into account. In the SMP model, we cannot represent the system rekeying rate with a single model parameter. Here, we improve our proposed security model and represent the system rekeying rate with a transition rate in the CTMC model.

In the upper part of Figure 4.1 is a CTMC model (the *Security Model*) proposed to depict the security attributes of an offloading system. It is a state transition model represents the system behavior under a specific attack and given system configuration that depends on the actual security requirements. We assume that the server is configured as to renew its key regularly to prevent or handle timing attacks. Whereas the lower part of Figure 4.1 is the open queueing model (the *Performance Model*), which is proposed to exhibit the offloading decision and job processing operation.

In our scenario, the offloading system is under timing attacks. If the attacker successfully compromises the system through time analysis, all jobs dispatched to offload are not secure any more, therefore they must be repeated and do not contribute to the throughput. That means only jobs processed locally contribute to the system throughput.

The aim of an attacker is to hack the master secret stored in the server. The attacker records each response time for a certain query and tries to guess the master secret of the server by comparing time

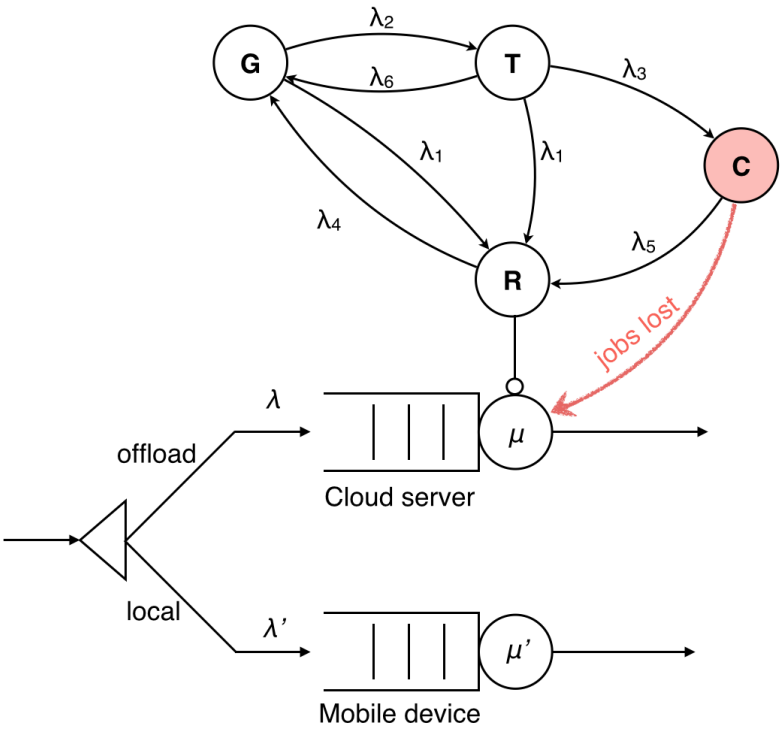


Figure 4.1: The hybrid CTMC and queueing model

differences from several request queues. Obviously, this requires the attacker to spend effort, where we use time to represent the attacking effort. We use exponential distribution to model the attacker arrival time and the time a timing attack takes.

The upper part of Figure 4.1 shows the CTMC model representing the states of the mobile cloud offloading system. The operational states of an offloading system are abstracted from the system lifetime analysis discussed in Section 3.3.2. The CTMC states are the same as the SMP model listed on page 42. We summarize the parameters of the CTMC model here:

- λ_1 rate at which the system launches the rekeying process in state G and state T
- λ_2 rate at which an attacker triggers a timing attack to the system
- λ_3 rate at which a timing attack succeeds to break the system secret
- λ_4 rate at which the system is brought back to the good state by the rekeying process
- λ_5 rate at which the system launches the rekeying process in state C
- λ_6 rate at which the attacker successfully breaks the key, while fails at accessing the data or he just fails to conduct a successful timing attack

We describe the events that trigger transitions among states in terms of transition rates. We assume that the time the system spent in each state is exponentially distributed. We also assume that there is only one attacker in the system at a time. If an attacker starts a timing attack to the cloud server, the system is brought to the timing attack state T at rate λ_2. The attacker has to make an effort before he successfully cracks the system secret by a timing attack, and the system moves to the compromised state C at rate λ_3. Consequently the mean time a timing attack takes is represented by λ_3^{-1}. If the attacker fails to conduct a successful timing attack, the system will go back to the good state G by the arc λ_6.

The rekeying rate is the parameter one can tune as a system administrator. It indicates how often the system launches the rekeying process. The rate λ_1 is the rekeying rate when the system is in the good state G or in the timing attack state T. The considered mobile cloud offloading system has intrusion detection mechanisms running on it that can find indicators of compromised behavior, in which case the system will trigger the rekeying process more frequently. The intrusion detection mechanism does not trigger the rekeying immediately because of the rekeying cost to the service performance. So in the compromised state C, we assume the rekeying process is triggered at a different rate, $\lambda_5 = n\lambda_1, n > 1$. The parameter n represents the relationship between the rekeying rate (or rekeying frequency) in good state and the rekeying rate in compromised state. The rekeying process will bring the system back to the initial state G at rate λ_4.

The challenge is to find an optimal value for the rekeying interval. The rekeying rate should be high to reduce the security lost cost in the state C, but triggering the rekeying process too often will lead to high system effort cost. We optimize this tradeoff in Section 5.5.

4.2.1 Performance Analysis

When jobs are generated by a mobile application, they are either offloaded to the cloud or executed locally. In the rekeying state R, the system refuses all user requests and all jobs are processed locally on the mobile device. Consequently all the jobs are dispatched to the Mobile device queue and some jobs will be lost. As a result, the system throughput is degraded. So the rekeying period should be as short as possible. When the system is in the compromised state C, which means the attacker successfully compromises the system through a timing attack, all jobs dispatched to offload are not secure any more. Hence they must be repeated and do not contribute to the throughput. The lost jobs are represented by the red arc in Figure 4.1. In this state, only jobs processed locally on the mobile device contribute to system throughput.

The lower part of Figure 4.1 shows the queueing model proposed to exhibit the performance attribute of the system. The two queues express the job processing by the cloud server and the

mobile device respectively. The parameters λ and λ' indicate the arrival rates for the two queues. A job dispatched to offload comes to the upper queue and is processed by the server with service rate μ, which also includes the data transmission time. For jobs dispatched to execute locally, the service rate is μ' which is assumed to be lower than μ.

4.3 Model Analysis

In this section, we derive and evaluate the security and performance attributes of the offloading system using methods for quantitative assessment of dependability, known as the dependability attributes, e.g. reliability, availability, and safety which have been well established quantitatively. We will evaluate the security and performance metrics by computing the steady-state probability of the CTMC model and solving the queueing model.

4.3.1 CTMC Steady-State Probability

For the system security attributes, we have described the system's dynamic behavior by a CTMC model with the state space $X_s = \{R, G, T, C\}$. The state space is the same as the SMP model, but the transitions between the states are represented by transition rates $\{q_{ij}, i \in X_s\}$. In order to carry out the security quantification analysis, we need to determine the stationary distribution of the CTMC model.

As in Section 3.1.1, the steady-state probabilities $\{\pi_i, i \in X_s\}$ of the CTMC can be computed by solving the system of linear equations

$$\pi \mathbf{Q} = 0, \tag{4.1}$$

where $\pi = [\pi_R, \pi_G, \pi_T, \pi_C]$ and $\mathbf{Q}$ is the infinitesimal generator (or transition-rate matrix) which can be written from Figure 4.1 as:

$$\mathbf{Q} = \begin{array}{c} \\ R \\ G \\ T \\ C \end{array} \begin{array}{c} \begin{array}{cccc} R & G & T & C \end{array} \\ \begin{pmatrix} -\lambda_4 & \lambda_4 & 0 & 0 \\ \lambda_1 & -\lambda_1 - \lambda_2 & \lambda_2 & 0 \\ \lambda_1 & \lambda_6 & -\lambda_1 - \lambda_3 - \lambda_6 & \lambda_3 \\ \lambda_5 & 0 & 0 & -\lambda_5 \end{pmatrix} \end{array} \tag{4.2}$$

In addition, we have the total probability relationship:

$$\sum_i \pi_i = 1 \quad i \in X_s. \tag{4.3}$$

The transition-rate matrix **Q** describes the dynamic behavior of the security model as shown in Figure 4.1. The first step towards quantitatively evaluating security attributes is to find the steady-state probability vector π of the CTMC states by solving Eqs. 4.1 and 4.3. We use Mathematica to solve the linear equations and get the solutions:

$$\begin{aligned} \pi_R &= \frac{[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi}, \\ \pi_G &= \frac{(\lambda_1+\lambda_3+\lambda_6)\lambda_4\lambda_5}{\phi},\ \pi_T = \frac{\lambda_2\lambda_4\lambda_5}{\phi},\ \pi_C = \frac{\lambda_2\lambda_3\lambda_4}{\phi}\ . \end{aligned} \tag{4.4}$$

For the sake of brevity, we assume:
$\phi = (\lambda_1+\lambda_4)(\lambda_1+\lambda_3+\lambda_6)\lambda_5 + [(\lambda_1+\lambda_4)\lambda_5 + (\lambda_4+\lambda_5)\lambda_3]\lambda_2$.

4.3.2 CTMC with absorbing state - MTTSF analysis

For quantifying the reliability of a software system, *mean time to failure* (MTTF) is a widely used reliability measure. MTTF provides the mean time it takes for the system to reach one of the designated failure states, given that the system starts in a good state. In reliability analysis, the failed states are made absorbing states. Once the system reaches one of the absorbing states, the probability of moving out of this state is 0, i.e., there are no outgoing arcs from such states. In this section, we use *mean time to security failure* (MTTSF) as the measure for quantifying the security of our offloading system. MTTF or MTTSF can be evaluated by making the compromised state of the CTMC an absorbing state, as shown in Figure 4.2.

Given a Continuous-Time Markov Chain (CTMC) with one absorbing state, we may enter this chain at some state i with probability α_i. For each state that is visited, the time before going to the next state follows an exponential distribution. Thus, the time required to reach the absorbing state from an initial state i is a sum of samples from exponential distributions. For a given CTMC, a phase-type distribution is defined as the distribution of the time to absorption that can be observed along the paths in a CTMC with one absorbing state [102].

In our scenario, the MTTSF is the mean time it takes for the system to reach the security failure state C. The first moment of a PH-distribution exactly expresses the mean time to absorption in an

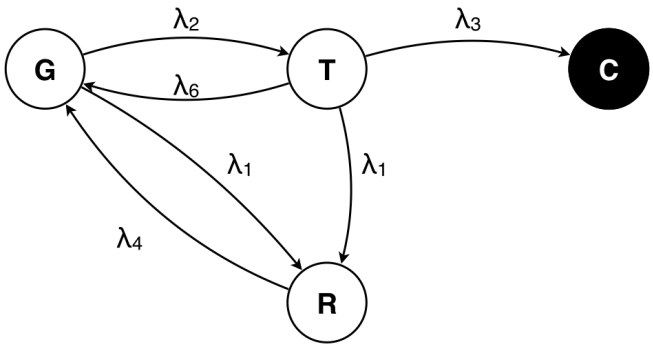


Figure 4.2: CTMC model with an absorbing state

absorbing CTMC. So for our model

$$MTTSF = E[X] = -\underline{\alpha}\mathbf{T}^{-1}\underline{1}. \tag{4.5}$$

The parameters are

$$\mathbf{T} = \begin{pmatrix} -\lambda_4 & \lambda_4 & 0 \\ \lambda_1 & -\lambda_1 - \lambda_2 & \lambda_2 \\ \lambda_1 & \lambda_6 & -\lambda_1 - \lambda_3 - \lambda_6 \end{pmatrix}, \tag{4.6}$$

and the initial probability vector is

$$\underline{\alpha} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}. \tag{4.7}$$

Substituting into Eq. 4.5, we get

$$MTTSF = \frac{(\lambda_1 + \lambda_4)(\lambda_1 + \lambda_2 + \lambda_3 + \lambda_6)}{\lambda_2\lambda_3\lambda_4}\ . \tag{4.8}$$

4.3.3 Computing Security and Throughput

Given the steady-state probabilities of CTMC model, we can compute the security and performance metrics we defined in Section 3.2. The security metrics have can be computed by Eqs. 3.26 and 3.27:

$$\Lambda = 1 - \frac{\lambda_2\lambda_3\lambda_4}{\phi}\ , \tag{4.9}$$

$$C = (1-w)\frac{[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi} + w\frac{\lambda_2\lambda_3\lambda_4}{\phi} \ . \tag{4.10}$$

Then we compute the throughput metric of the offloading system. As shown in the lower part of Figure 4.1, for each queue, the throughput equals the average number of jobs in the queue divided by the average time a job spends in the queue. The system throughput equals the sum of the throughput of the two queues.

We assume the total system life time is T. In the good state G and timing attack state T, the number of jobs served by the system should be $\lambda(\pi_G+\pi_T)T + \lambda'(\pi_G+\pi_T)T$, given the queues are stable. While in the rekeying state R, the server refuses all the users' requests and all jobs must be executed locally. Assuming $\mu' < \lambda + \lambda'$, the number of jobs served then is $\mu'\pi_R T$. In the compromised state C, all the jobs dispatched to offload are not secure, so they do not contribute to the throughput. In this state, the system throughput only covers the jobs executed locally $\lambda'\pi_C T$. Therefore, we get the system throughput as

$$\begin{aligned} X =& \frac{\lambda(\pi_G+\pi_T)T + \lambda'(\pi_G+\pi_T)T + \mu'\pi_R T + \lambda'\pi_C T}{T} \\ =& (\pi_G+\pi_T)\lambda + (1-\pi_R)\lambda' + \pi_R\mu' \ . \end{aligned} \tag{4.11}$$

4.4 Numerical results

In this section, we evaluate the proposed metrics using the model analysis results. First we specify the parameters for the security model. We use the system parameters we propose in [98]. It is assumed that the attack rate to the system is $\lambda_2 = 1$. Because a timing attack is considered to consume more time than the time between attacks [20], the rate at which a timing attack succeeds to break the system secret is assumed to be smaller than the attack rate, i.e. $\lambda_3 = 0.3$. We also assume it takes an average of 0.5 time units for the server to carry out a rekeying process and the rate at which the system is brought back to the good state by the rekeying process is $\lambda_4 = 2$. The rate at which the attacker successfully breaks the key, while it fails at accessing the data or it just fails to conduct a successful timing attack is assumed to be $\lambda_6 = 1$.

For the system parameters we use experimental data from an offloading engine and OCR (Optical Character Recognition) implementation [130] here. The mean local execution time for an OCR job on the mobile device was 2377 ms. We set $\mu' = 1/2.377 \approx 0.42$. The mean offloading time including the data transition time is 1191 ms. Then $\mu = 1/1.191 \approx 0.84$. For the queues to be

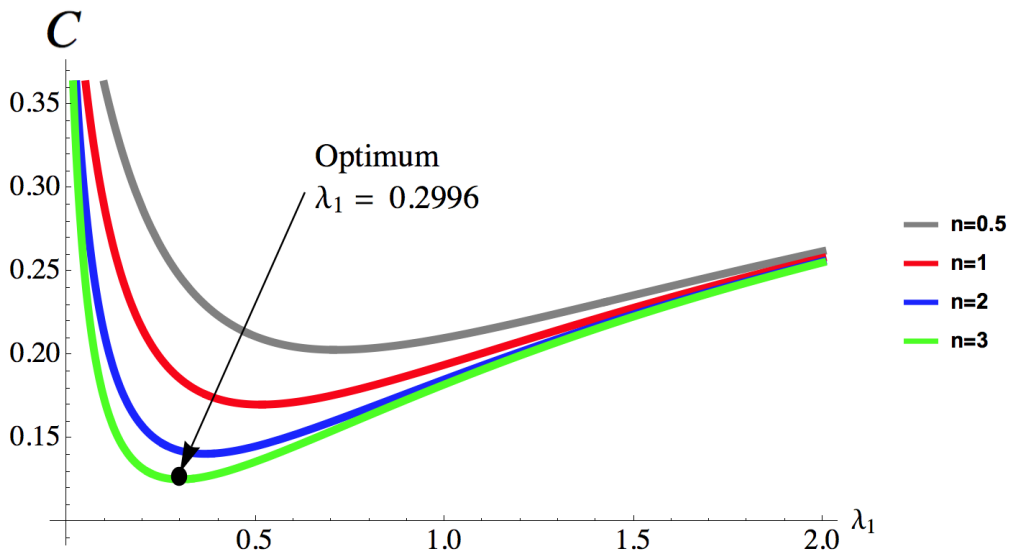


Figure 4.3: System cost C as a function of the rekeying rate λ_1

stable, we assume $\lambda = 0.8$ and $\lambda' = 0.4$.

- **System cost analysis**

Figure 4.3 shows the system cost metric C (defined in Eq. 3.15) changing with the rekeying rate λ_1. Here we set the weighting parameter $w = 0.5$ to put equal importance to the loss of sensitive information cost and the effort needed to rekey regularly. The parameter n in this figure is the coefficient of rekeying in the compromised state, i.e. $\lambda_5 = n\lambda_1$. In this chapter, we consider 4 rekeying options, that is $n = 0.5, 1, 2, 3$ respectively. The rekeying rate λ_1 indicates how often the system launches the rekeying process. The higher the rekeying rate, the more often the rekeying process is triggered. When the rekeying rate is low, the system cost is very large due to the high probability of an insecure state. We find the optimum rekeying rate $\lambda_1 = 0.2996$ for the lowest system cost when $n = 3$. After the lowest value, because of the increasing effort to perform rekeying process, the cost is also getting larger at high rekeying rate. We further see that the system cost decreases with increasing coefficient n. This is because for all rekeying rates, the mean time in the compromised state decreases as we rekey more frequently in this situation.

We study the effect of the weighting parameter w on the system cost in Figure 4.4. We look at the marginal values first. It can be seen from the figure that the cost decreases monotonically with the rekeying rate λ_1 when $w = 0$, where we only consider the costs of losing sensitive information

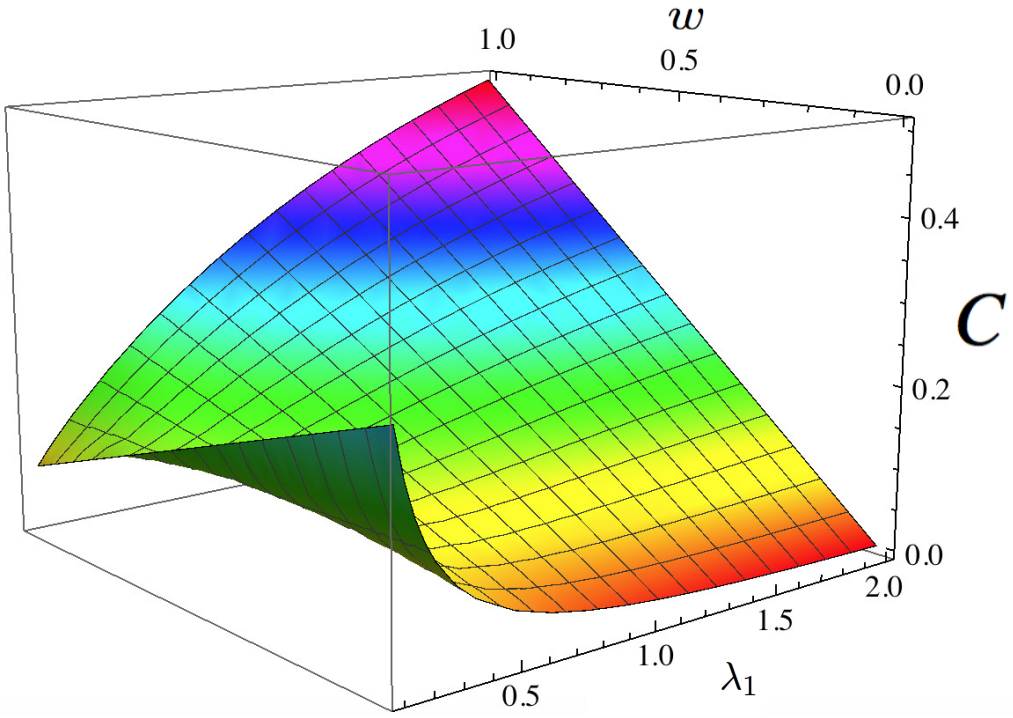


Figure 4.4: System cost metric C over rekeying rate λ_1 and weighting parameter w

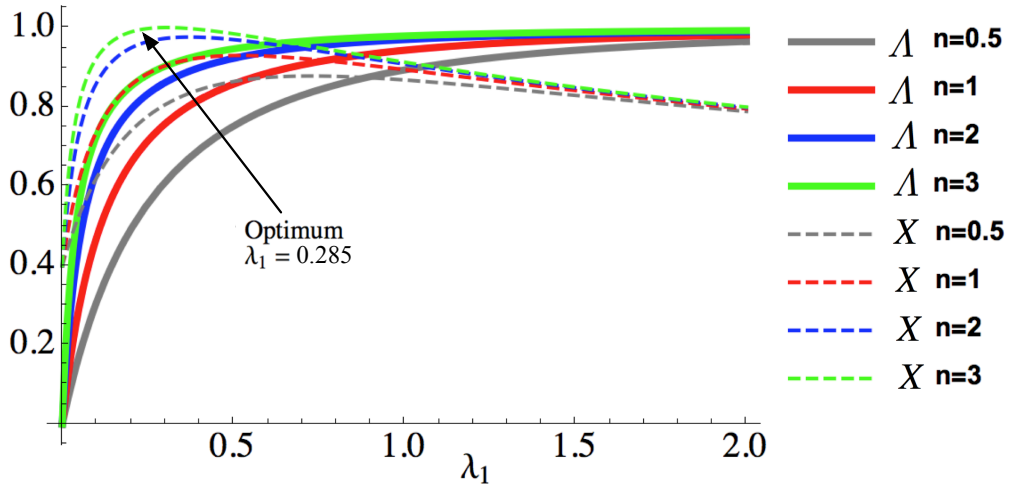


Figure 4.5: Confidentiality metric Λ and throughput X over the rekeying rate λ_1

in the compromised system. Intuitively, in this case when we trigger the rekeying process more often, the security cost will decrease. When we put all weight on the rekeying effort (w = 1), the cost increases with the rekeying rate. The light color in the middle of the figure shows the optimum rekeying rate. For the middle values of the weighting parameter w, the optimum rekeying rate for the lowest cost decreases when we put more weight on the rekeying effort cost. For each specific rekeying rate λ_1 the system cost is a straight line weighting the two kinds of cost. In this figure, we get the largest rekeying effort cost and lowest security cost at rekeying rate $\lambda_1 = 2.0$.

- **Performance analysis**

Figure 4.5 shows the system security and performance metrics, i.e. system confidentiality Λ (defined in Eq. 3.14) and throughput X (defined in Eq. 3.16), changing with the rekeying rate λ_1. Also 4 rekeying options are considered here, i.e. $n = 0.5, 1, 2, 3$ respectively. It can be seen that the confidentiality metric Λ monotonically increases with growing rekeying rate λ_1. It also increases when the multiple of the rekeying rate in the compromised state n is larger. This is because the security improves when the system launches the rekeying process more frequently, as the system is more likely to be brought back to good state from the timing attack state and the compromised state. At small values of the rekeying rate, the system throughput is low because more time is spent in the compromised state when the offloading throughput is not contributing. We find the highest throughput when the rekeying rate $\lambda_1 = 0.285$, when $n = 3$. After obtaining the maximum

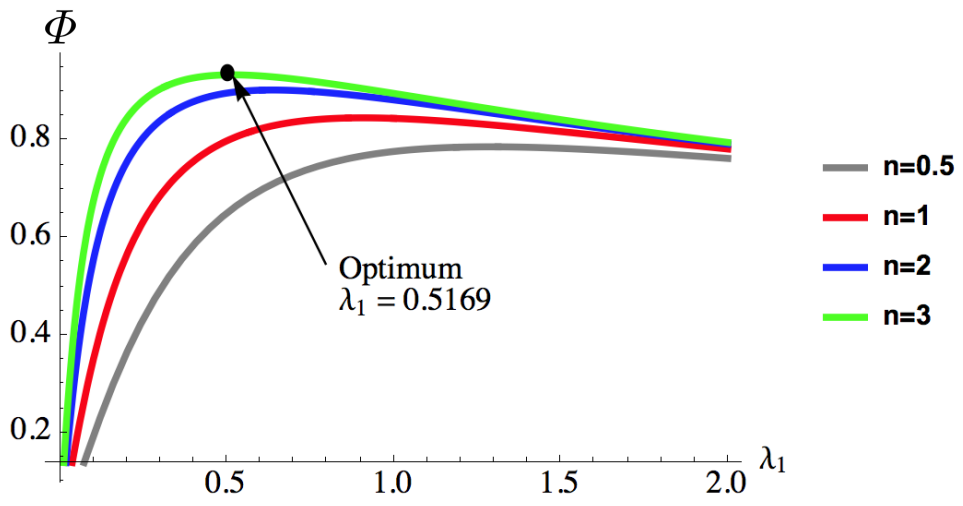


Figure 4.6: Security and Performance Tradeoff with Rekeying Rate λ_1

throughput, the more often the server triggers the rekeying act, the more often the server denies offloading requests. As a result, the system throughput decreases with the rekeying rate.

- **Tradeoff analysis**

At last, we present the security and performance tradeoff analysis for the offloading system in Figure 4.6. The tradeoff metric Φ (defined in Eq. 3.17) increases rapidly with the rekeying rate λ_1 at its low values, as the system security improves quickly. We find the optimum rekeying rate for the best security and performance tradeoff at $\lambda_1 = 0.5169$, when $n = 3$ (n is the multiple of rekeying rate in compromised state). This optimum rekeying rate is different from the one for the lowest system cost since they look at different aspects of evaluating the system. It is also different from the rekeying rate for the highest system throughput. That is because the optimum rekeying rate for throughput is not the optimum for the system security.

However, after reaching the optimum value, the tradeoff metric decreases much slower as the rekeying rate is getting larger. When the rekeying rate has a large value, the multiple parameter n does not affect the tradeoff metric much as the rekeying act is triggered frequently enough. The system tradeoff metric decreases because of the degrading system throughput at large rekeying rates.

We show a method that can be used by the system administrator to find out how to tune the security mechanism (rekeying) in the system. The system cost can act as a criterion for service providers to

charge their users. If users need higher security level, they have to pay more for offloading services. The system administrator can also use the result to obtain the minimum cost or maximum security performance tradeoff for the system.

4.5 Summary

In order to proceed to a quantitative treatment of the security-performance tradeoff of offloading systems we have proposed a hybrid CTMC queueing model for an offloading system under the specific threat of timing attacks. We have shown how to formulate measures that include both security and performance attributes and that optimize the tradeoff between the two. System metrics are also proposed which take into account both the rekeying effort a system makes and the sensitive information loss. The optimum rekeying rate is found for the tradeoff metric depending on n, the parameter for the rekeying rate in compromised state. We found that with carefully selected parameters, we can configure the offloading system to achieve an optimal security and performance tradeoff.

Chapter 5

A Secure and Cost-efficient Offloading Policy

We present an offloading scheme which is the combination of regular rekeying and random padding in this chapter. We quantitatively analysis the security and performance metrics which system architects need to make informed tradeoff decisions involving system security.

In order to proceed to a quantitative treatment of the tradeoff problem, a hybrid CTMC and queueing model has been proposed in the last chapter to model the mobile cloud offloading system which treats security and performance attributes respectively. By extending this model, we introduce a random padding countermeasure in the offloading system for mitigating the information leakage through time side-channels.

This chapter proposes and evaluates a secure and cost-efficient offloading scheme which optimizes the performance and security tradeoff of the offloading system, with the major contributions being twofold:

- We propose a secure and cost-efficient offloading scheme for mobile cloud computing by combining renewing the server key regularly with inserting random delays into the server processing time. The numerical results based on experimental data show that the security performance tradeoff of offloading can be improved through our scheme.
- We implement a system that allows us to compare the impact of different random padding strategies on the expected success rate of timing attacks. We tune the mean and the variance of the random paddings and investigate the impact on mitigating the time side-channel information leakage. It is revealed that the variance of random delays is the decisive factor to mitigation effectiveness of a random padding and the extra number of measurements an at-

tacker has to make to conduct a timing attack grows linearly with the standard deviation (SD) of the random padding.

5.1 The Offloading Scheme

A timing attack is a practical threat to the mobile cloud offloading system. Besides renewing the server's private key periodically, we introduce a random padding countermeasure to the system in order to mitigate this attack.

Figure 5.1 depicts the diagram of the performance and security model of a CTMC and an open queueing model which we proposed in the previous chapter for describing dynamic behavior of a mobile cloud offloading system. We extend the hybrid model by adding random delays in the offloading queue as a countermeasure to mitigate timing attacks.

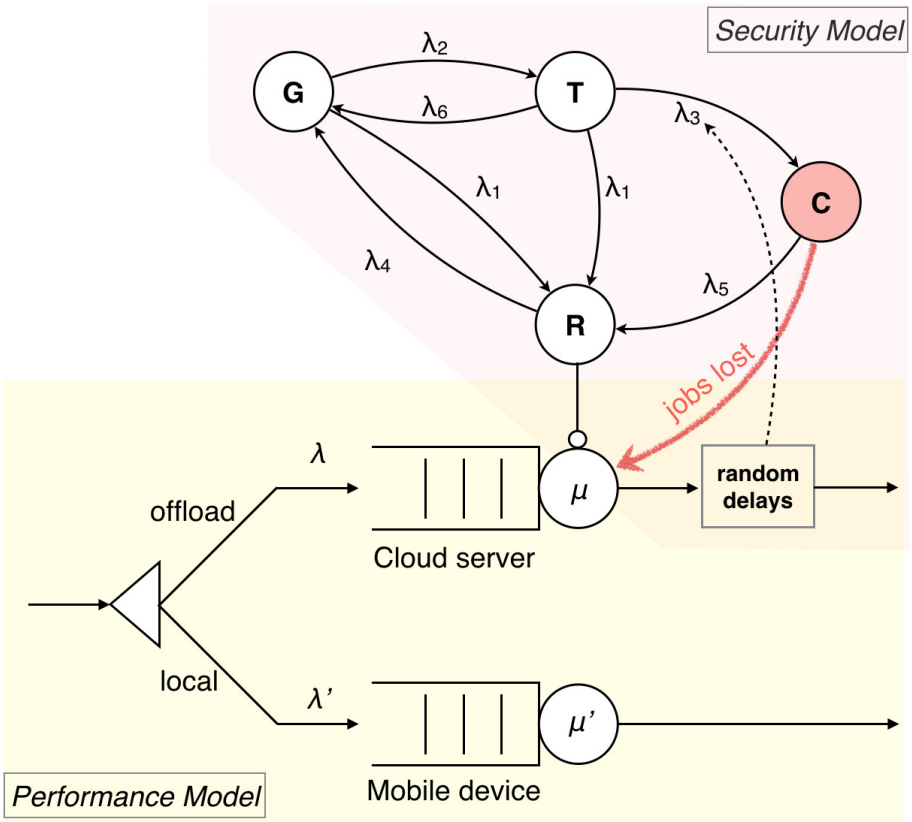


Figure 5.1: Performance and Security model for a generic mobile cloud offloading system

Random delays are padded for each service response in order to mitigate timing information leakage. When random delays are interposed, the attacker needs more samples to average and successfully guess the secret in the server. So it takes more time for it to conduct the timing attack. As a result, the attacking rate λ_3 decreases as this mitigation method is taken. This process is represented by the dashed arc in Figure 5.1.

5.2 Experiments

In order to empirically evaluate our approach for mitigation to timing attacks and for improving the security and performance tradeoff of mobile cloud offloading systems, we set up a simulation using the OMNeT++ tool. Experimental and numerical analyses have been done to investigate:

1) the mitigation effectiveness of the random delay countermeasure against timing attacks;
2) the impact of the random padding parameters on the mitigation effectiveness. We study how the mean and the variance affect the number of additional measurements an attacker may have to make; and
3) the optimal system rekeying rate for the performance and security tradeoff.

5.2.1 Experiment setup

Our server and client applications are developed using the OMNeT++ tool based on the INET 2.6 framework. The connection between two hosts use the TCP protocol as shown in Figure 5.2. All tests were run under Mac OS X 10.10 on a 2.6 GHz Intel Core i5 processor with 8 GB 1600 MHz DDR3 RAM.

A timing attack uses statistical analysis of the time one application takes to do some calculation in order to learn about the data it is operating on. Timing attacks on secrets are fundamentally limited by the ability of an attacker to accurately measure differences in response times across a network. In our implementation, we explore the limits and mitigation effectiveness of random delay countermeasure against timing attacks by deploying a client application to record and analyze the amount of time taken by the server application to compare two values bit by bit. We mimic a Brumley's remote timing attack which is discussed in Section 2.2.1.

The experiments are conducted as follows: As an attacking client sends a guessed value g (a 256-bit value), the server application receives the value and compares it with the one that has been stored in the server from the first bit to the last. Once the server finds that one bit in the received value is different, it sends back an alert to the client. Otherwise, the server continues to compare the next bit.

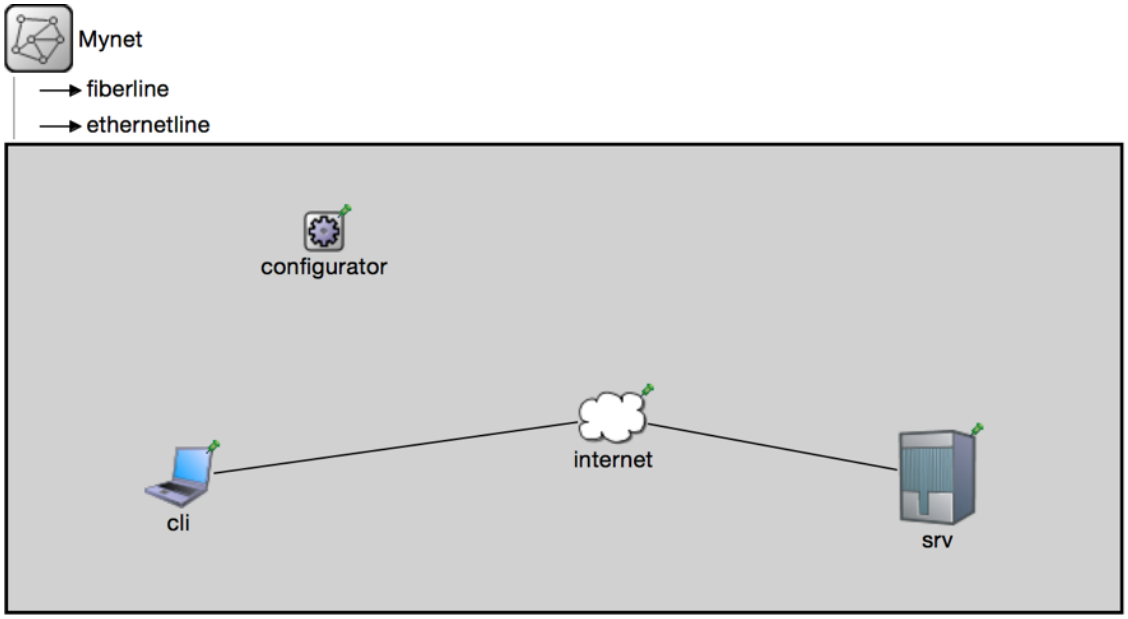


Figure 5.2: Experiment setup

So if the first bits are the same, it takes slightly longer for the server to send back the result. The client records the response time of the server and makes guesses of the server's secret based on its measurements.

A random padding countermeasure is deployed on the server side. After the server processes each message received from the client, random delays drawn from different distributions are padded before sending the results to mitigate the timing information leakage.

5.2.2 Convolution method

First, we analyze the completion time distribution for timing attacks. We have discussed the implementation of Brumley's timing attack in Section 2.2.1. A complete Brumley's timing attack can be viewed as a binary search for a system secret and it consists of several steps to recover the ith bit of the secret. The attacker repeats these steps to recover the secret bits one by one. We call

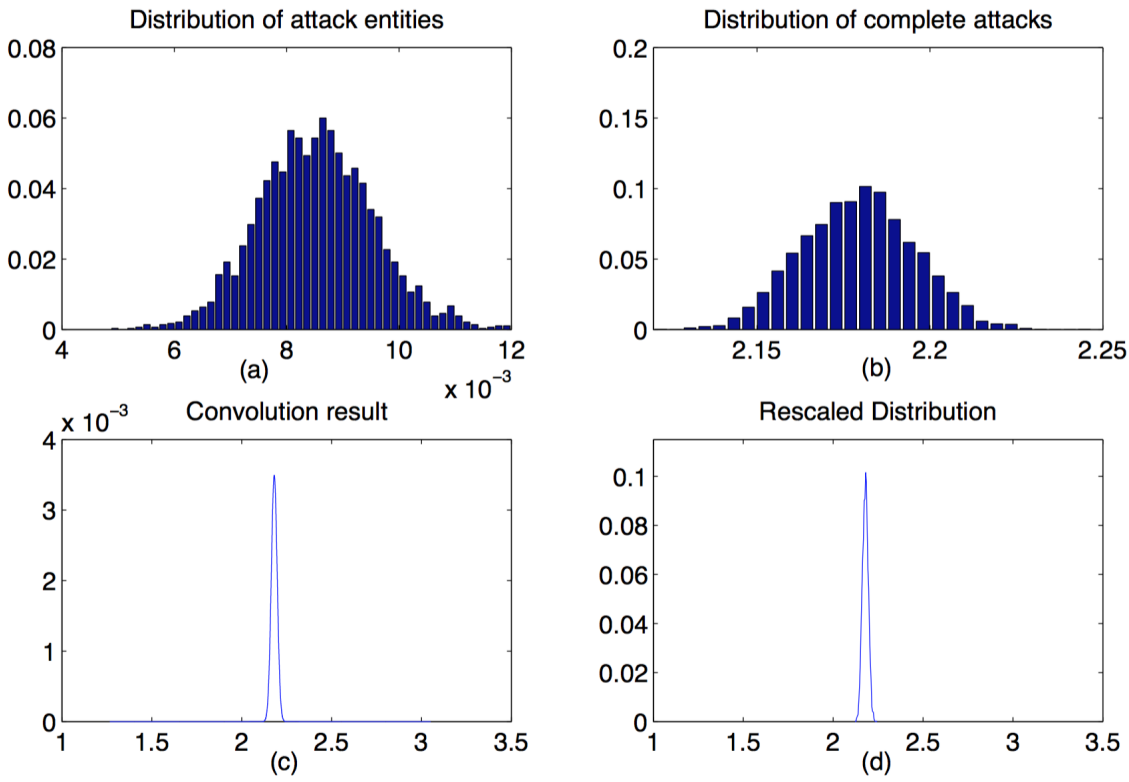


Figure 5.3: Test and Verify the Convolution Method. (a) The time distribution for an attack entity. (b) The time distribution of complete attacks which consists of 256 entities. (c) The result of interactively convolution method. (d) The rescaled distribution of complete attacks.

the process to recover 1 bit an *attack entity*.

After recovering the half-most significant bits of the system secret, the attacker can use Coppersmith's algorithm [33] to retrieve the complete factorization. Then the system is successfully compromised by the attacker by timing attack. From the setup of Brumley's remote timing attack, a typical attack takes approximately 2 hours, and to get its distribution may take days. So we try to simplify this process using the convolution method.

For each secret bit, the attacking behavior can be regarded as a single entity. And these entities are assumed to be independent and identically distributed (i.i.d.). When the distribution of the attack entity time is known, the cumulative distribution function (CDF) of one complete attack duration can be computed by the interactive convolution method. It needs 256 attack entities to factor a RSA-1024 bit key. To simplify the computational process, we propose Algorithm 1 by doing the convolution pairing.

Algorithm 1 Interactively convolution algorithm

```
1: Input: CDF of an attack entity: p
2: Output: CDF of the complete attack
3: for i = 1:8 do
4:     p ← conv(p,p);
5: end for
6: return p
```

Then we can get the 256 attack entities distribution by 8 self-convolutions. The results are shown in Figure 5.3. The mean of Figure 5.3(c) is 2.181 h and the variance is 0.000264 respectively. For Figure 5.3(d), the mean is 2.179 h and the variance is 0.000267. We test and verify that the convolution method is adequate for our scenario. This method can radically decrease the computation time for the subsequent evaluation.

5.3 Comparison of Distributions

We draw random delays from different random distributions. This experiment aims at comparing the impact of different random distributions to the limits of timing attacks against offloading systems. The parameters, the mean and the variance of different distributions are shown in Table 5.1. We set the mean of all random distributions as 0.1 ms. For the Erlang distribution, it is difficult to get a large variance because the shape parameter has to be integer.

The attacking client sends two messages separately with a certain bit equals 0 and 1 to the server. Random delays are added after the server processes each message received from the client. Different numbers of timing samples are taken from the client measurement. When the client can distinguish the time difference of server application processing two different messages from statistical analy-

Table 5.1: Continuous Distributions

	Mean	Variance	SCV
Weibull(0.05, 0.5)	0.1	0.05	5
Uniform(0, 0.2)	0.1	0.0033	0.33
Exponential(0.1)	0.1	0.01	1
Truncated normal(0.1, 0.1)	0.1	0.01	1
Erlang(5, 0.1)	0.1	0.002	0.2

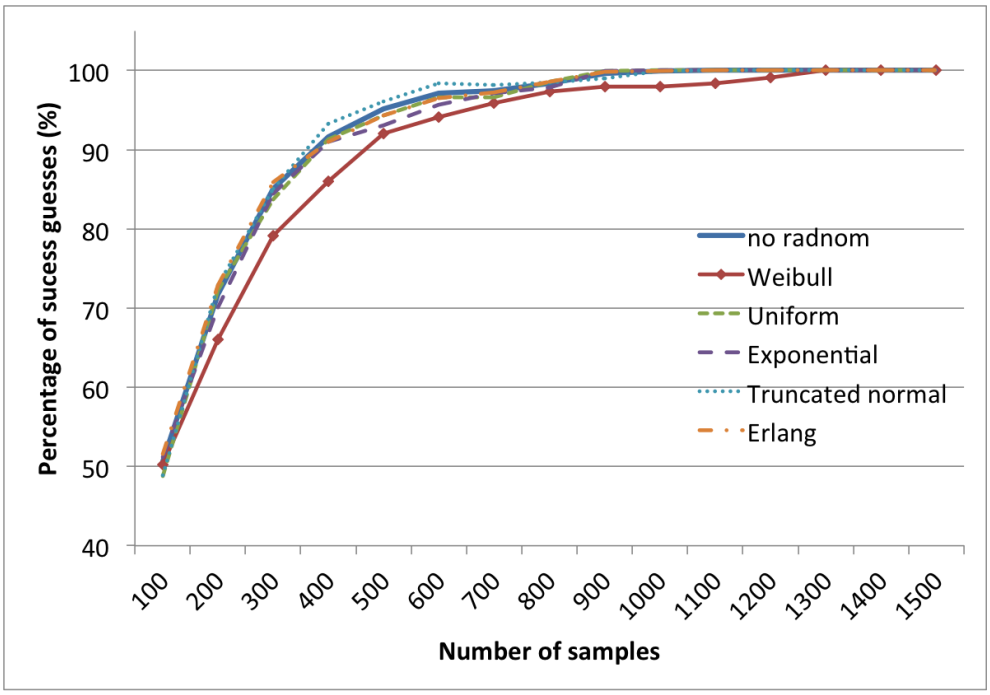


Figure 5.4: Comparison of different random delay distributions

sis of the samples, we call it a successful attack. We use the percentage of successful guesses to represent the mitigation effect against timing attacks of random delay countermeasure.

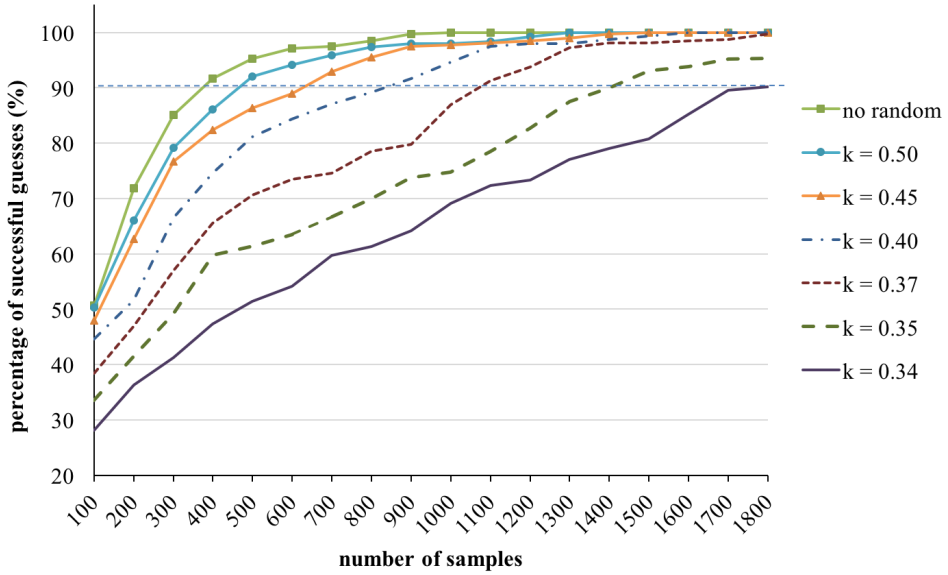
The result is depicted in Figure 5.4. It shows that the Weibull distributed delays can mitigate the timing attacks as the attacker needs more samples to guess the secret than the situation if no random delays are added. The results of the other three random distributions are the superposition of the result with no random padding. The impact of the other three distributions is negligible because their variance is small.

In the next section, we choose the Weibull distribution to draw random delays because it is widely used in reliability engineering and failure analysis and it is easy to change the variance of Weibull distribution by tuning the parameters.

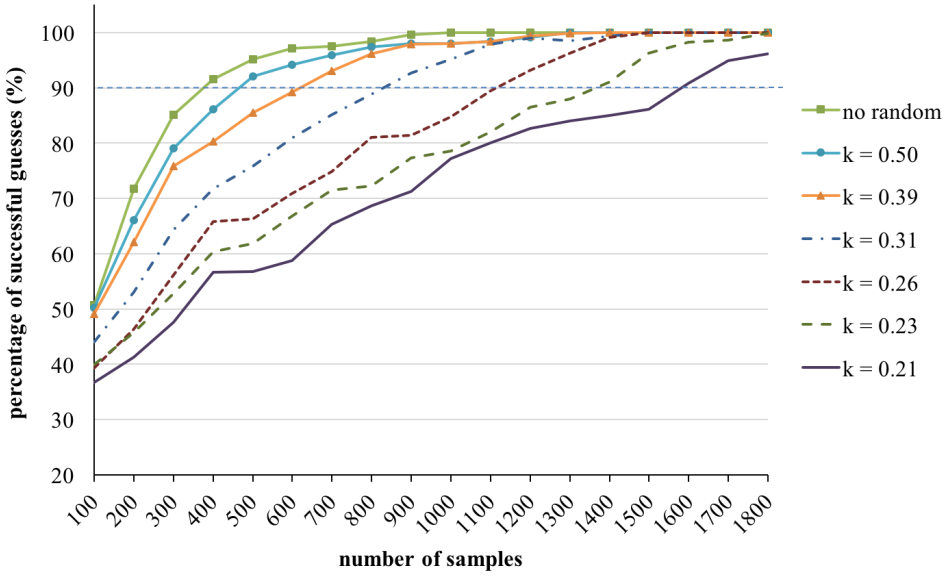
5.4 Mitigation Effectiveness of Random Delays

In this section, we investigate the timing attack resilience of mobile cloud offloading systems with random delay paddings. It is worth mentioning that Figure 5.5 has to be read together with Table 5.2.

We evaluate the mitigation effectiveness of random delay countermeasure against timing attacks



(a)



(b)

Figure 5.5: Comparison of the effectiveness of Weibull distributed random delays with different parameter sets. (a) Mitigation effectiveness of Weibull random delays with same *scale parameter*. (b) Mitigation effectiveness of Weibull random delays with same *mean*.

Table 5.2: The parameter sets of the Weibull distribution for the two experiments in Figure 5.5

(a)

shape k	scale η	mean(ms)	variance	SCV	n(sample)
no random					375
0.50	0.0500	0.1000	0.0500	5.00	470
0.45	0.0500	0.1239	0.1043	6.79	625
0.40	0.0500	0.1662	0.2725	9.87	830
0.37	0.0500	0.2092	0.5642	12.89	1070
0.35	0.0500	0.2515	0.9980	15.78	1400
0.34	0.0500	0.2944	1.6151	18.64	1750

(b)

shape k	scale η	mean(ms)	variance	SCV	n(sample)
no random					375
0.50	0.0500	0.1000	0.0500	5.00	470
0.39	0.0287	0.1000	0.1043	10.43	625
0.31	0.0116	0.1000	0.2725	27.25	830
0.26	0.0053	0.1000	0.5642	56.42	1120
0.23	0.0027	0.1000	0.9980	99.80	1370
0.21	0.0015	0.1000	1.6151	161.51	1580

by comparing the number of response time measurements an attacker needs to achieve a certain level of successful guesses about the server secret. Different numbers of timing attacks are taken by the client. When the client can tell the secret bit of the server from statistical analysis of the samples, we call it a successful guess.

The impact of different randomly distributed delays to the limits of timing attacks has been compared in the previous section. It has been shown that Weibull distributed delays can mitigate the timing attacks more effectively than the random delays picked from some other common distributions, such as uniform, exponential and Erlang distributions. As the attacker needs more samples to guess the server's secret. So we choose Weibull distributed delays as the mitigation countermeasure against timing attacks.

We perform two experiments with different parameter sets for the Weibull distribution as shown in

Table 5.2. The first experiment is conducted by changing the Weibull distribution *shape parameter* $k \in \{0.5, 0.45, 0.4, 0.37, 0.35, 0.34\}$ while keeping the *scale parameter* $\eta = 0.05$. We set these parameters in order to increase the *variance* of random delays. The result is depicted in Figure 5.5 (a). It is showed that the Weibull distributed delays can mitigate the timing attacks as the attacker needs more samples to guess the secret than if no random delays are added.

It is assumed that the attacker uses an error detection and correction strategy as described in [23], so 90% successful guesses is sufficient for a successful attack. We record the numbers of samples on the 90 percentile of successful guesses in the subtables. As one can see, the attacker only needs 375 timing samples to make 90% successful guesses when there is no random delay padding. However, when the Weibull distributed delays with $k = 0.5$ are used, it needs 470 samples to get the same percentage of successful attacks, that is, the attacker needs to spend more effort in the timing attack procedure. As the shape parameter k increases (at the same time the variance is larger), the attacking client needs more samples to guess the server's secret. As a consequence, the effectiveness of the mitigating countermeasure is getting better compared with no random delay padding.

In order to analyze the impact of the *mean* and *variance* of a Weibull random delay to the mitigation effectiveness against timing attacks, we conduct the second experiment by adjusting the two parameters as to keep the mean constant while increasing the variance. We set the mean to $0.1ms$ and the variance is the same as in the first experiment (Figure 5.5b). Surprisingly, the results are nearly the same as in the first experiment (Figure 5.5a), i.e. the attacker needs the same number of measurements for a successful guess. This outcome indicates that the mean is a negligible factor as changes of the mean does not affect the mitigation effectiveness. However the variance of random delays is the primary influencing factor to the mitigation effectiveness.

To support our argument, we conduct an experiment changing the mean of the Weibull random delays while keeping the variance constant. The result is presented in Figure 5.6. It indicates that changing the mean does not significantly affect the mitigation effectiveness as the results are superposed on each other. The attacker needs nearly the same number of samples to conduct a successful timing attack when different Weibull distributed random delays are superposed. Different random padding policies with the same variance have the same effect on mitigating timing attacks even though the mean is growing.

5.4.1 Quantitative Relationship

In this subsection, we quantitatively analyze the relationship between the variance of Weibull random variable and the mitigation effectiveness of random delay countermeasure in Figure 5.7. We use the number of extra samples the attacker needs to present the mitigation effectiveness against

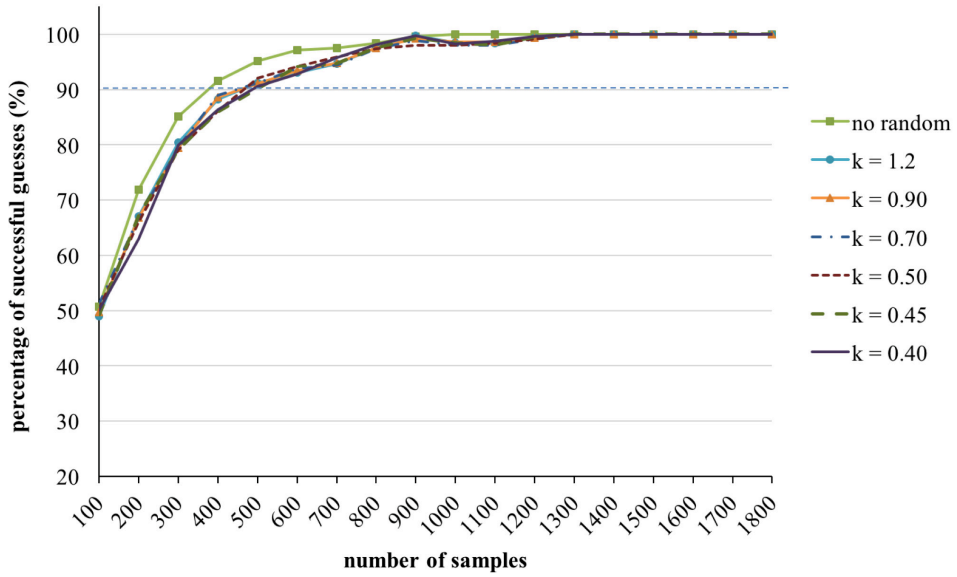


Figure 5.6: The effectiveness of Weibull distributed random delays with the same variance but different means

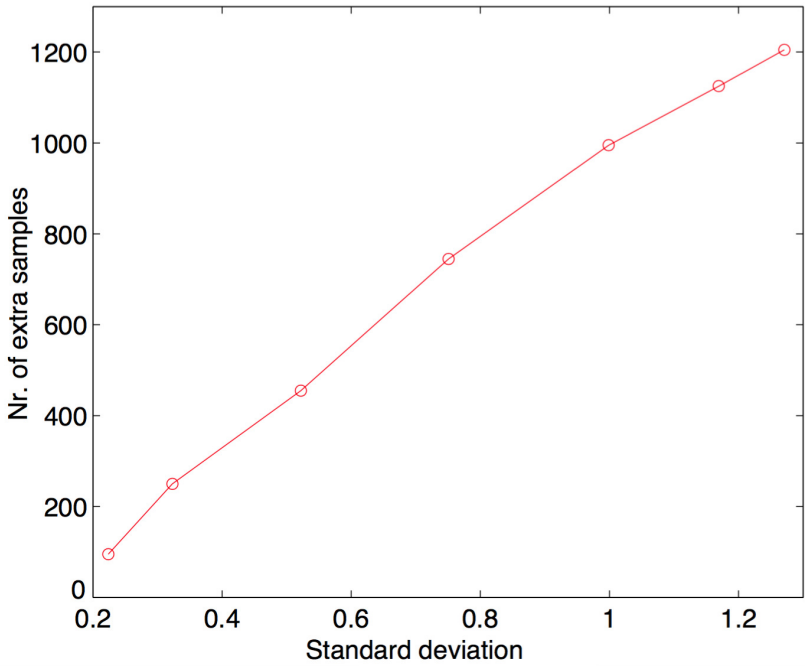


Figure 5.7: The number of extra samples as a function of the standard deviation of Weibull distributed random delays

timing attacks, i.e., when Weibull distributed delays are added, the extra number of measurements that the attacker has to make to get the same level of successful guesses.

One can see that the number of extra samples needed by the attacker grows linearly with the standard deviation of the Weibull random delay. This observation matches previous results in the power side-channel [31].

In the next subsection, numerical results are presented to analyze the tradeoff between the security and performance in mobile cloud offloading systems.

5.5 Numerical Results

In this section, we evaluate the our offloading policy using the model analysis methods. The rekeying rate λ_1 is the parameter that a system administrator can tune and the goal of this evaluation is to find the optimal rekeying rate for the mobile offloading system.

We use the experiment results to parameterize the system model. The transition rates for the CTMC model are taken from the implementation of our previous work [98]. The timing attack success rate λ_3 is taken from the timing attack experiment in Section 5.4.

Figs. 5.8 and 5.9 show the system security and performance metrics, i.e. system confidentiality Λ and throughput X, changing with the rekeying rate λ_1. The rekeying rate λ_1 indicates how often the system launches the rekeying process. It can be seen that the confidentiality measure monotonically increases with growing rekeying rate λ_1. This is because security improves when the system launches the rekeying process more frequently. One can also see security improves with growing shape parameter k of the Weibull distributed delays. That is because when k is larger, the variance of the random delays are larger and the mitigation effect against timing attacks is better.

However, the throughput does not grow monotonically with the rekeying rate. It has an optimum, which means one can tune the rekeying rate to obtain a maximum system throughput. Figs. 5.9 reveals that the system throughput grows when the shape parameter k increases. Even though adding random paddings brings an overhead to the system response time, the performance is still improved because random paddings mitigate the information leakage which reduces the throughput lost in the insecure state. The system administrator can choose the offloading strategy with the highest throughput from Figure 5.9. But the optimal rekeying rate does not mean the optimal security situation.

We present the security and performance tradeoff analysis for the mobile cloud offloading system in Figure 5.10. We have computed the tradeoff metric in Section 3.2.3 which is higher better. We find the optimum rekeying rate for the best security and performance tradeoff at $\lambda_1 = 0.5169$, when

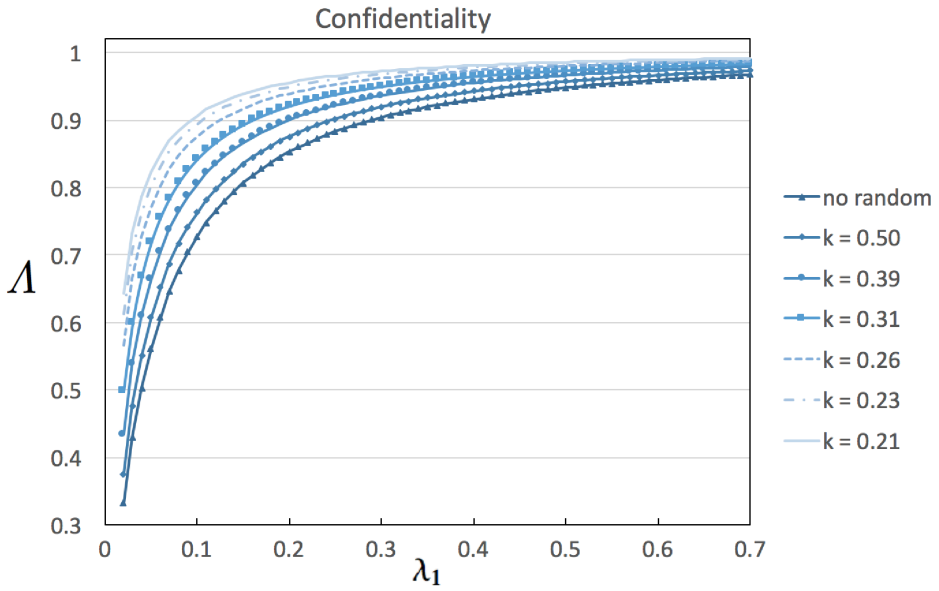


Figure 5.8: Confidentiality metric Λ changing with Rekeying Rate λ_1

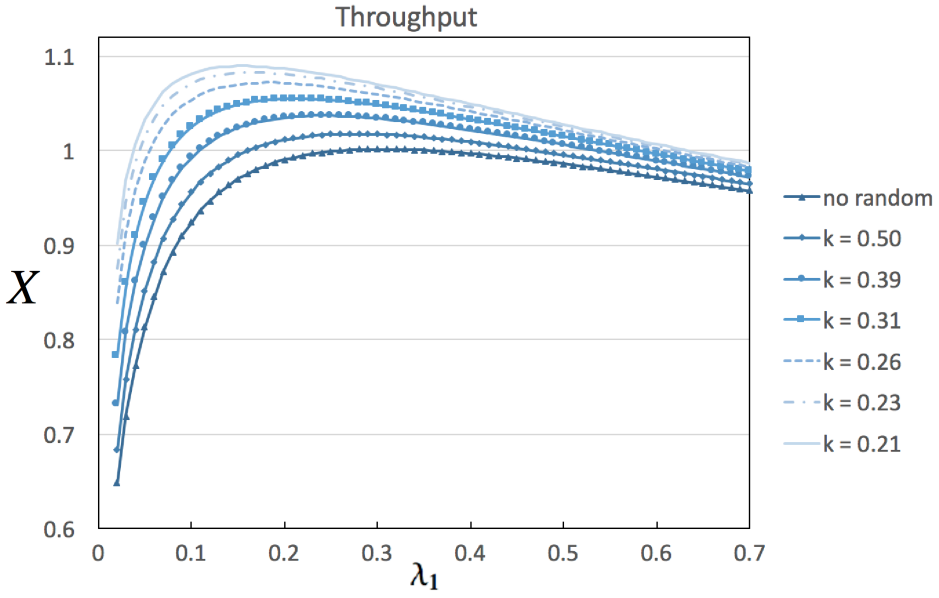


Figure 5.9: Throughput metric X changing with Rekeying Rate λ_1

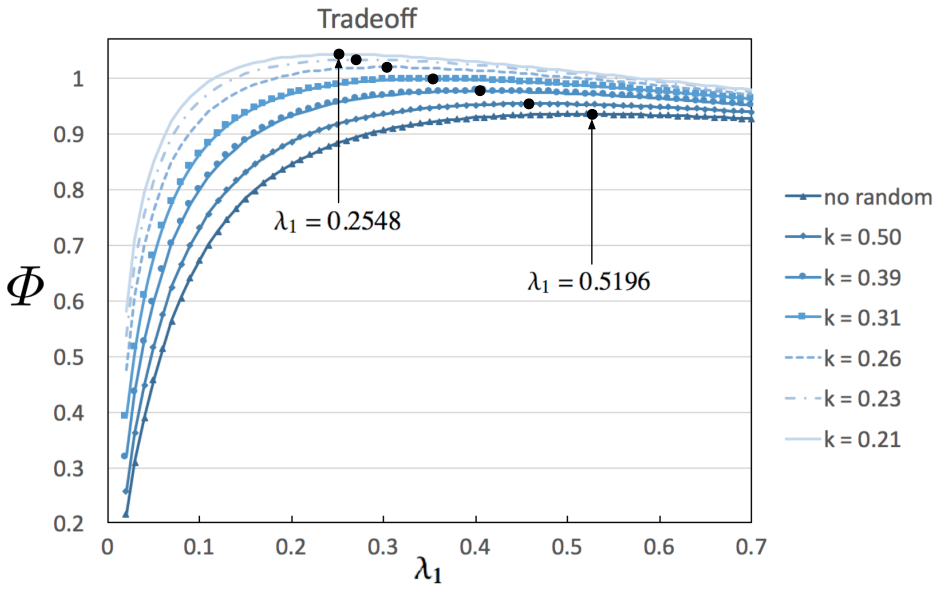


Figure 5.10: Security and Performance Tradeoff changing with Rekeying Rate λ_1

no random delays are added. When the rekeying rate has a large value, the system tradeoff metric decreases because of the degrading system throughput at large rekeying rates. One can see that the system has the lowest tradeoff metric when no random delays are padded. As Weibull distributed random delays are inserted to mitigate the timing information leakage, the tradeoff metric is getting greater. This shows that the random delay countermeasure can improve the system security against timing attacks while meeting the system performance requirements. From Figure 5.10, Weibull distributed random delays with the shape parameter $k = 0.21$ is the most desired padding policy we can choose and the optimal rekeying rate for the tradeoff metric is $\lambda_1 = 0.2548$. Then we get the optimal offloading policy for the considered mobile cloud offloading system. From Table 5.3, it can be seen that the optimal rekeying rate decreases when the variance of Weibull distributed delay is growing. As a consequence, the system needs to make less effort to launch the rekeying process, which leads to less system cost. This trend can be seen from the dashed arrow in Figure 5.10.

In summary, when random delays are deployed in mobile cloud offloading systems, one should try to enlarge the variance of the random delay while keeping the mean as low as possible by tuning the parameters. Using the results in Figure 5.9 and Figure 5.10, the system administrator can obtain the optimum rekeying interval for the minimum cost or maximum security performance tradeoff.

Table 5.3: Optimum rekeying rate for different Parameter-sets of Weibull distribution

Shape k	Scale λ	Mean	Variance	Optimal rekeying rate
no random				0.5169
0.50	0.0500	0.1000	0.0500	0.4643
0.39	0.0287	0.1000	0.1043	0.4045
0.31	0.0116	0.1000	0.2725	0.3518
0.26	0.0053	0.1000	0.5642	0.3030
0.23	0.0027	0.1000	0.9980	0.2738
0.21	0.0015	0.1000	1.3682	0.2548

5.6 Summary

In this chapter, we have proposed a secure and cost-efficient scheme for mobile cloud offloading systems against timing attacks by combining renewing the server key regularly with inserting random delays into the processing time. A random padding countermeasure has been added to our hybrid CTMC and queueing model for mitigation to timing attacks. We have set up a simulation to investigate the impact of the random padding parameters on the mitigation effectiveness.

The numerical results based on experimental data show that the security performance tradeoff of the offloading system is improved through the proposed scheme. Further, we found that the variance of random delays is the primary influencing factor to the mitigation effectiveness of random padding and that the extra number of measurements an attacker has to make grows linearly with the standard deviation of the random delays. So when random delays are deployed in mobile cloud offloading systems, one should tune the parameters to enlarge the variance while keeping the mean as low as possible.

Part III

Improving Client Security

Chapter 6

An Empirical Evaluation of Container Solutions

The analysis presented in chapter 4 and 5 mainly consider the server security of mobile cloud offloading systems, while we address the client security issues in this chapter. In mobile cloud computing, mobile clients can enhance the security of offloading by employing secure containers. A secure container is an authenticated, encrypted area of a user's mobile device designed to separate, isolate and protect enterprise data from attackers.

Bring Your Own Device (BYOD) security is a concern for many companies' IT departments. Many corporations implement a "secure container" solution that provides full separation of work and personal data on mobile devices to mitigate the dangers brought by BYOD. In this paper, we perform an empirical comparison between two popular secure containers for Android: Samsung Knox and IBM MaaS360. We first conduct benchmark tests to compare these two containers. Then in order to quantitatively assess the security property of these containers, we propose a measurement method based on a simulated attack. Our experimental results show that performance of compute-intensive applications in the Knox container will be affected drastically compared with when they are running on the device (outside the container), while for memory-intensive applications, performance will not deteriorate much in Knox and MaaS360 containers. We also found that with an overhead of 3.3 ms (0.58%) in mean response time, Knox container can extend the mean time to security failure (MTTSF) by 109.6 min (878%). Within the MaaS360 container, the MTTSF is prolonged by 10.2 min (81.4%) but the response time is 3.3 ms (0.58%) longer per job than without containers.

6.1 Android container solution

Bring Your Own Device (BYOD) refers to the policy of permitting employees to bring their personally owned devices (laptops, tablets, and smartphones) to their workplace, and to use those devices to access privileged company information and applications [100]. BYOD is on the rise and a recent Tech Pro Research report [1] found that 72% of enterprises polled were permitting BYOD or are planning to do so. While it may be beneficial for the employees to extract more work satisfaction and also upgrade to the latest hardware more frequently than in the painfully slow refresh cycles at most organizations, BYOD certainly exhibits new challenges in terms of security due to increased use of personal devices with potentially unknown threats.

The wide spread of mobile devices and their progressive functions, ranging from receiving emails to accessing bank accounts, make them a notable target for attackers [125]. This, together with the fact that users habitually store sensitive security information (SSI) in such devices, makes BYOD security a major concern for the IT department of a company.

The popularity of the BYOD security has attracted the attention of many researchers in recent years. Researchers in [39] propose a MUSES (Multi-platform Usable Endpoint Security) system to enforce an enterprise policy on the mobile devices. The system applies Machine Learning and Computational Intelligence techniques to refine its security policies. DeepDroid proposed in [132] is a custom instrumentation tool. DeepDroid also tries to enhance BYOD security by enforcing enterprise policy on Android devices. But it requires root access, in which case the device may be subject to other threats.

One promising solution to BYOD security is the secure container. A container is an authenticated, encrypted area of a user's mobile device designed to separate, isolate and protect enterprise data from attackers. Corporate data such as email, documents, and enterprise applications are encrypted and processed inside the container. This work/personal environment ensures that work data and personal data are separated and that only a work container is managed by the enterprise. The container is an important part of the Mobile Device Management (MDM) as all MDM products are built with an idea of containerization [38].

We focus on the secure container solutions on Android platforms, which is attractive for the following reasons. First, Android is the most popular mobile operating system which has occupied 82.8% market share in the second quarter of 2015 [46] on various hardwares. Second, Android is representative with respect to the state-of-the-art in mobile computing. Last, Android has become the major target of the attackers to mobile devices since the security review process of new applications is not rigorous enough in many application stores [63].

In this chapter we present an empirical assessment of security and performance attributes of secure container solutions (Samsung Knox and IBM MaaS360) on Android devices. First, we take benchmark tests to compare different containers. They show that mobile devices witness a notable deterioration in the performance of compute-intensive applications using the Knox container. However, for memory-intensive applications, performance does not degenerate much in both Knox and MaaS360 containers compared with running them on the device outside the container. Secondly, in order to perform a quantitative evaluation of the security attribute, we set up a testbed that conducts simulating attacks to different container solutions. Our experiment results show that with a Knox container, the security gain is that the MTTSF is extended by 109.6 min (878%), but the performance impairment is that the mean response time is 3.3 ms (0.58%) longer. As for a MaaS360 container, with an overhead of 3.3 ms (0.58%) per job, the MTTSF is extended by 10.2 min (81.4%). The performance loss also includes an additional 2.7 ms (0.53%) of algorithm execution time in the Knox container and 1.4 ms (0.28%) in the MaaS360 container.

6.2 Samsung Knox and IBM MaaS360

In this Section, we introduce the context of two widely used Android container solutions: Samsung Knox and IBM MaaS360.

6.2.1 Samsung Knox

Knox is Samsung's solution to BYOD security threats. It was first proposed in early 2013 with version 1.0.0. This version was deployed in Android 4.3 on Galaxy S3 and S4. The latest version is Knox 2.6 deployed in Android 6.0 *Marshmallow*. Knox provides a "secure container" which is a secure environment alongside the user's personal environment.

Knox Platform Architecture

Knox addresses mobile device security by building a comprehensive, hardware-rooted trusted environment with multi-layer architecture (shown in Figure 6.1):

- **Hardware Root of Trust** The hardware boot is protected by three components: 1) the Device Root Key (DRK) is a device-unique asymmetric key that is signed by Samsung through an X.509 certificate; 2) the Samsung Secure Boot key is used to sign Samsung-approved executables of boot components and 3) Rollback Prevention Fuses are hardware fuses that encode the minimum acceptable version of Samsung-approved executables.

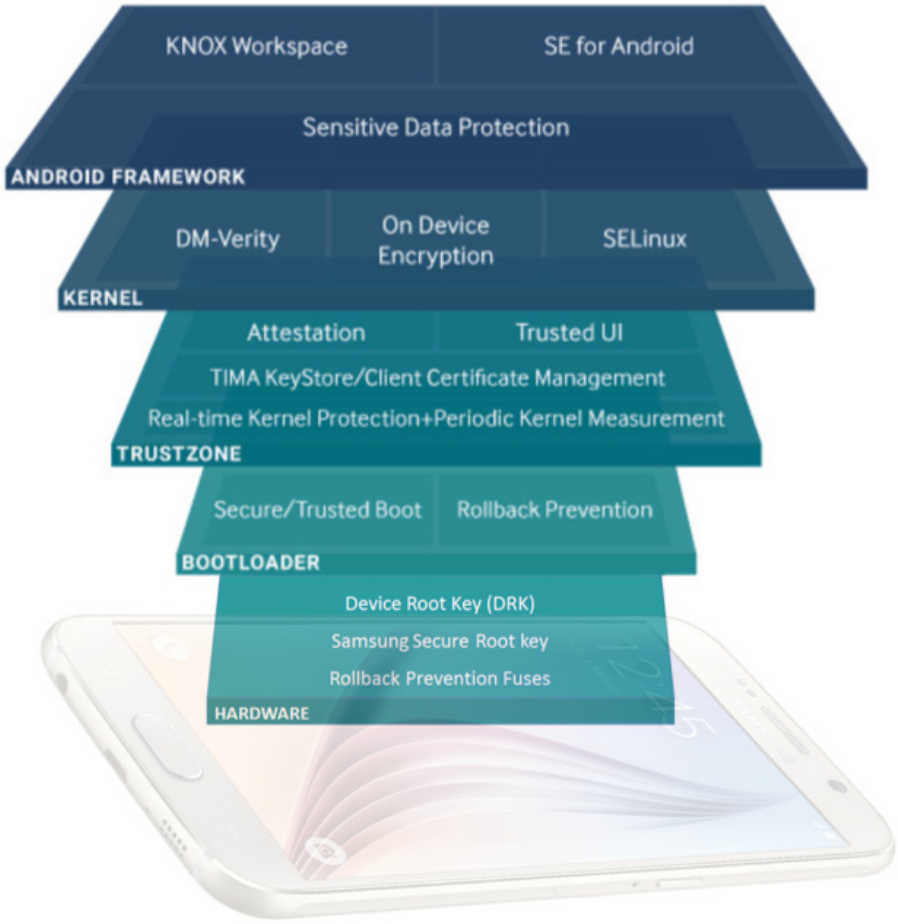


Figure 6.1: Samsung Knox Platform Architecture [114]

- **Secure Boot and Trusted Boot** Secure Boot is a security mechanism that prevents unauthorized bootloaders and operating systems from loading during the startup process. It guarantees the initial integrity of an Android kernel and the code running in the TrustZone. Trusted Boot is able to distinguish between different versions of authorized binaries and takes measurements of the bootloaders. At runtime, TrustZone applications use these measurements to make security-critical decisions.
- **TrustZone** TrustZone is a set of security extensions added to ARMv6 processors and greater [11]. These ARM processors can run a secure operating system (secure OS) and a normal operating system (normal OS) at the same time from a single core.
- **TIMA** TrustZone-based Integrity Measurement Architecture (TIMA) relies on the protection and isolation of the TrustZone's secure world from the normal world and ensure the operating system (OS) kernel integrity. TIMA contains periodic kernel measurement (PKM), real-time kernel protection (RKP) and remote attestation.
- **SE for Android** Knox introduced Security Enhancements for Android (SE for Android) to enforce Mandatory Access Control (MAC) policies [114]. These enhancements protect applications and data by strictly defining what each process is allowed to do, and which data it can access. This layer's security depends on the integrity of the kernel and the security policy stored on disk, which is guaranteed by TIMA.

All versions of Knox already tie the Samsung hardware to the security system running on the device – at a layer below the operating system. As the developer of both the hardware and security system, Samsung has the same advantage that BlackBerry has long enjoyed in such integration. Apple's iOS devices also have vertically integrated security, but Apple severely restricts access to that stack, so government agencies and others cannot customize it in the way that Samsung allows.

In addition to securing the operating system, Samsung Knox addresses the security of individual applications by using containers and data encryption.

Knox Workspace

Knox Workspace is designed to separate, encrypt and protect enterprise data from attackers.

- **Container environment** Knox Workspace provides a virtual Android environment to isolate enterprise applications and data in their own secure zone. Once activated, the Knox Workspace product is tightly integrated into the Knox platform. Applications outside Workspace cannot use Android inter-process communication or data-sharing methods with applications inside Workspace. For example, photos taken with the camera inside Workspace cannot be

viewed outside. The same restriction applies to copy and paste. Since KNOX 2.0, application wrapping is no longer required. Thus one can run any existing Play Store application in the Workspace container.

- **Sensitive Data Protection** Knox defines two classes of data – protected data and sensitive data. All data written by applications in Workspace is protected data. Protected data is encrypted on disk when the device is powered off. In addition, the decryption key for protected data is tied to the device's hardware. This makes protected data recoverable only on the same device. Even stronger protection is applied to sensitive data. Sensitive data remains encrypted as long as the Workspace is locked, even if the device is powered on. When a user unlocks Knox Workspace using their password, Sensitive Data Protection (SDP) allows sensitive data to be decrypted.

6.2.2 IBM MaaS360

MaaS360 is an enterprise mobility management platform developed by IBM. MaaS360 provides a comprehensive approach for companies to manage and safeguard their mobile devices, applications and content. As a fully integrated cloud platform, MaaS360 simplifies the deployment of MDM and allows visibility and control across mobile applications and documents from a single user interface. Furthermore, MaaS360 Mobile Application Security provides a container solution to help enterprises enforce authentication, set up single sign-on across containerized applications and configure data leak prevention (DLP) controls.

An overview of the complete MaaS360 package is shown in Figure 6.2. We summarize the important features as follows:

- **Containerization** MaaS360 Secure Productivity Suite (secure container) allows the IT department to manage all the emails, contacts, calendars, applications and the web from an isolated workspace on the employees' mobile devices. It protects enterprise data and applications with containerization which prevents access from compromised devices, such as jailbroken or rooted devices. With the container, MaaS360 allows to integrate security control of mobile applications and specification of encryption settings. It enforces data file protection and uses application-level tunneling for protected access to corporate data, without needing a device VPN.
- **Enterprise Gateway** MaaS360 Mobile Enterprise Gateway is an activated module as part of the MaaS360 Cloud Extender (CE) [61]. It provides mobile access to resources behind the firewall such as SharePoint, Microsoft Windows file sharing content, intranet sites and appli-

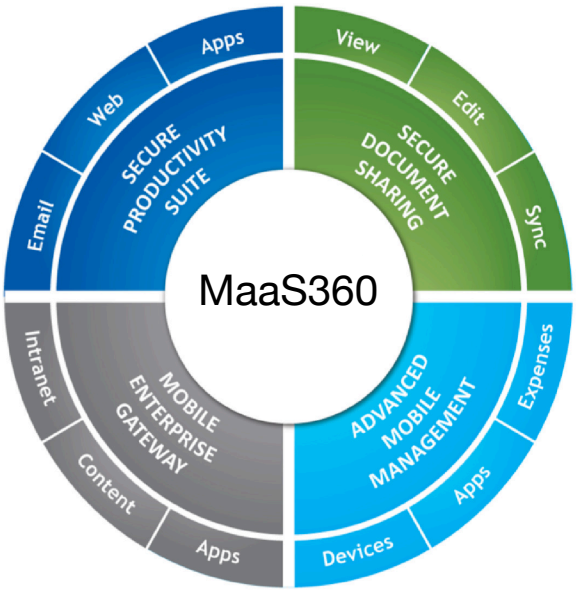


Figure 6.2: IBM MaaS360 overview [4]

cation data. Unlike browser-based applications, where device caching may lead to security leaks, MaaS360 Mobile Enterprise Gateway ensures that confidential data is never stored on devices in an unencrypted format, and that a user's ability to transfer that information elsewhere can be limited by administrative policy. MaaS360 Mobile Enterprise Gateway ensures that corporate data can only be viewed on authorized mobile devices and the communication between the enterprise gateway and the mobile devices are fully encrypted.

- **Authentication and access control** MaaS360 Advanced Mobile Management enforces on-device access control and compliance with policies and regulations. It can also deliver and update these policies remotely to the application container, based on user and device security posture.
- **SDK option** MaaS360 uses the Software Development Kit (SDK) option to help enable security controls directly in the application code and add containerization features enterprise applications.

IBM MaaS360 also allows to remotely locate, lock and wipe lost or stolen devices and selectively wipe corporate data while leaving personal data intact. Table 6.1 shows a comparison of these two secure container solutions.

Table 6.1: Comparison of Knox and MaaS360

Similarities	Differences
containerization; supporting email, contact and calendar management;	Knox ensures hardware root of trust; Maas360 supports enterprise gateway; Maas360 must work in the MDM model, but Knox can work in client mode only

6.3 Benchmark Results

In this Section, we explore the benchmark results of the mobile devices in different Android container solutions.

We adapt Geekbench 4 for Android developed by Primate Labs [3]. Geekbench provides a comprehensive set of benchmarks designed to measure processor and memory performance of mobile devices. It displays Single-Core Score and Multi-Core Score as shown in Figure 6.3. In mobile systems, the single core performance is a more import measure than the multi-core performance, because the multi-core performance is related to multi-task processing. However, in most use cases of mobile devices, only the main core may reach its full utilization, while the other cores are barely used. So we only compare the single-core performance of a mobile phone in different secure container solutions. For the experiments we used two different mobile phones, a Samsung Galaxy S6 (2.1 GHz Exynos 7420 CPU with 3 GB RAM) and a Samsung Galaxy S5 (2.46 GHz Qualcomm Snapdragon 810 CPU with 2 GB RAM).

Geekbench for Android uses a number of different tests, or workloads, to measure a mobile device's performance. The workloads are divided into three different sections:

Integer performance

Most software makes heavy use of integer instructions, which means a high integer performance indicates good overall performance of the mobile device.

- **AES**: The AES workload encrypts a generated text string using the advanced encryption standard (AES) [99]. AES is used in security tools such as SSL, IPsec, and PGP. Geekbench 4 uses AES instructions (AES-NI or ARMv8 AES) when available. Otherwise Geekbench uses its own AES implementation.

- **JPEG compression and decompression**: The JPEG workload compresses and decompress one digital image using lossy JPEG format. The workload uses JPEG library version 6b.

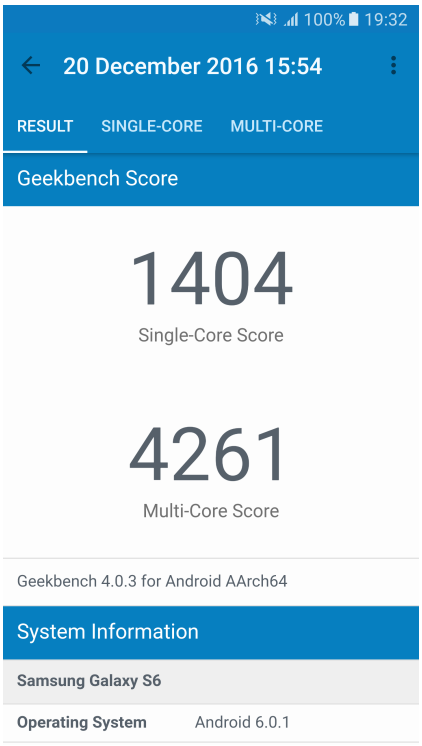


Figure 6.3: Screen shot of Geekbench 4

Floating point performance

While almost all software makes use of floating point instructions, floating point performance is especially important in video games, digital content creation, and high-performance computing applications.

- **SGEMM**: SGEMM is short for "Single float precision General Matrix Multiplication. This workload tests how fast a mobile device can calculate the product of two matrices. Matrix multiplication is such a common operation with a wide variety of practical applications that it has been implemented in numerous programming languages.
- **SFFT**: The Sparse Fast Fourier Transform (SFFT) [107] workload simulates the frequency analysis used to compute the spectrum view in an audio processing application.

Memory performance

The STREAM benchmark [94] is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. Software working with large amounts of data (e.g., digital content creation) relies on good memory bandwidth performance to keep the processor busy.

- **STREAM copy**: The STREAM copy workload tests how fast a device can copy large amounts of data in memory. It executes a value-by-value copy of a large list of floating point numbers.
- **STREAM bandwidth** The STREAM bandwidth sums the amount of data that the application explicitly reads plus the amount of data that the application explicitly writes.

Figures 6.4 and 6.5 show the results of the two mobile devices' performance under different workloads. We run the benchmark program ten times in each container and present the average results. All the workload results are "high better". We found that the Knox container has the lowest integer performance. As one can see in Figure 6.4, the Samsung S6 device processes 7.37 MB/s of AES encrypted data per CPU core, less than the MaaS360 container (7.91 MB/s) and the on device (7.93 MB/s) situations. As for the JPEG workload, the average compresses and decompress rate

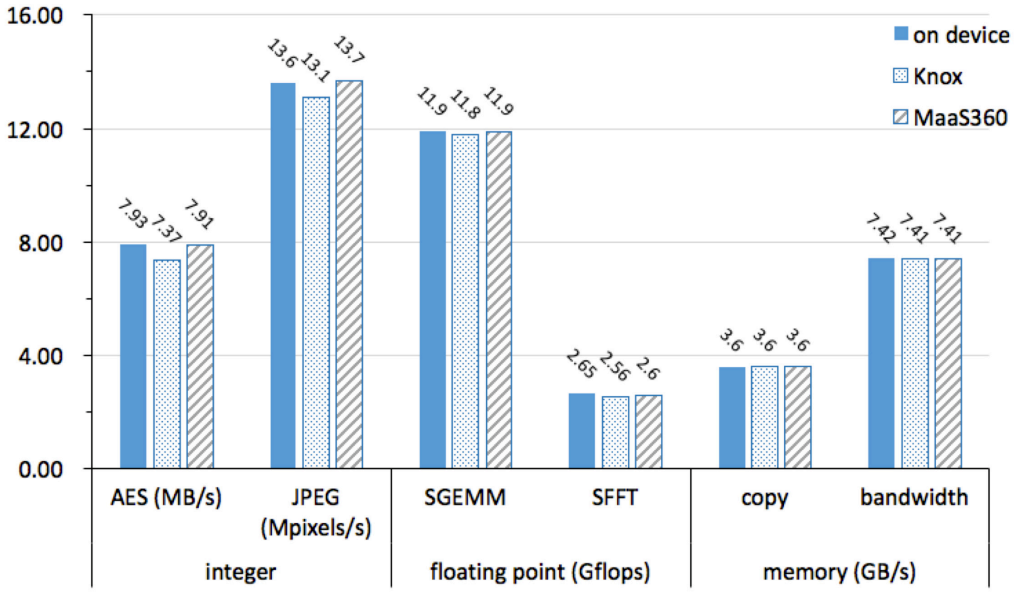


Figure 6.4: Benchmark results of Samsung Galaxy S6 under different workloads

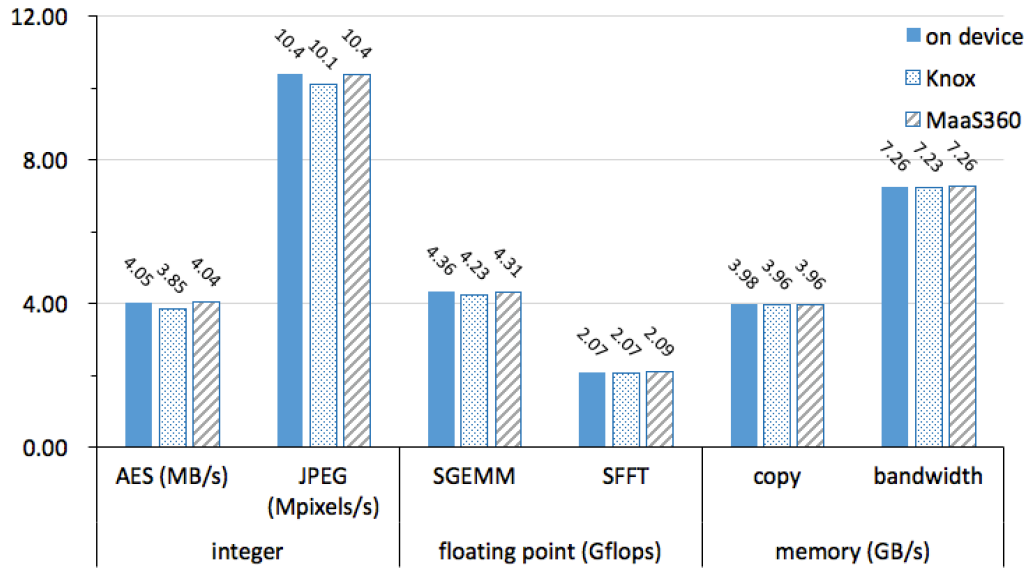


Figure 6.5: Benchmark results of Samsung Galaxy S5 under different workloads

is 13.1 Mpixels/s, while the rates are 13.7 Mpixels/s and 13.6 Mpixels/s in the MaaS360 container and on device respectively. However, the MaaS360 container does not affect the integer computation performance dramatically compared with the on device situation. This is probably because the Knox container has higher encryption level than the MaaS360 container, but we cannot get the encryption details for both containers. The floating point workload result also shows that the Knox container leads to performance deterioration when the mobile device is performing SGEMM computation (0.1 GFlops less than MaaS360 and on device situations).

However, the memory measurements of the mobile device in different containers are almost the same as on the device. The Samsung S6 device can copy 3.6 GB/s floating point numbers in memory in all the three situations. The memory bandwidth is 7.42 GB/s and 7.41 GB/s in the Knox container and the MaaS360 container, and 7.41 GB/s on the device. The benchmark result of Samsung S5 device (Figure 6.5) is similar to S6 and we do not repeat the details for simplicity. Because the Samsung S6 device has more computation power and memory than the S5, so its integer and floating point results are generally higher than the latter.

So in the Knox container, the performance of compute-intensive mobile applications will be affected, while for memory-intensive applications, the performance will not deteriorate obviously in Knox and MaaS360 containers compared with when they are running on the device.

6.4 Security and Performance Analysis

In this section, we present a quantitative security assessment and investigate how much performance we lose when gain more security using the containers. In order to proceed to a quantitative treatment of the security attributes of Android containers, we simulate attacks to the containers and evaluate the containers' defensive effect.

6.4.1 Metrics

The performance metrics of interest describe mobile systems in terms of throughput, completion time, or response time, as defined in queueing theory or networking. We use the mean response time (denoted by T) as the performance metric for the container solutions.

The mean time to security failure (MTTSF) is the security metric we use. We investigate the time an attacker needs to break the secure containers by performing timing attacks. A timing attack is a side-channel attack which poses a practical threat on mobile devices as its remote feasibility has been proved in [20].

6.4.2 Experiment Setup

We develop a target application (shown in Figure 6.6) to receive messages from clients and send back responses. We publish the target application through Google Play private channel and deploy it in a Knox container, a MaaS360 container and on the device (outside the containers). The simulated attacker performs timing attacks to the target application running on the mobile device. A timing attack uses statistical analysis of how long it takes to do some calculation in order to learn about the secret. The key idea of conducting a timing attack is to identify information by analysing the time differences [97]. The target application is developed using Android Studio 2.2.2 and deployed on the Samsung Galaxy S5 device. The attacking client is developed with Java and runs on a laptop. Figure 6.7 shows an illustration of the experiment setup.

Similar to the Brumley and Boneh's attack [21], in our scenario, the attacker sends two messages m_1 and m_2 to the target application. After receiving the message m_1, the application simulates a RSA algorithm and then sends an answer message back. When the client sends message m_2, the application waits an additional $1ms$ before sending back the answer. The attacking client takes measurements of the response times (T) of the target application and estimates the time difference based on the measurements. T' is the response time when the target application is deployed in a Knox or MaaS360 container.

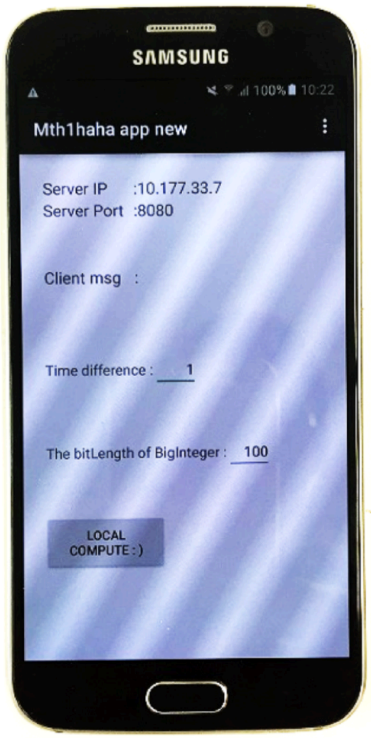


Figure 6.6: The target application

6.4.3 Simulating Attacks

We conduct a series of experiments that perform timing attacks to different container solutions and evaluate the number of samples a real attacker needs when it could identify an empirical resolution of $1ms$ in the Android application response time. In statistical terminology this amounts to performing a *hypothesis test* [35].

We simulate an attacker conducting empirical hypothesis tests and we evaluate the effectiveness of identifying the time difference in the response time. The simulated attacks follow this procedure: the hypothesis test H is given two sets of totally $N = 2500$ samples of measured response times $X = T_1[1], ...T_1[N]$ and $Y = T_2[1], ...T_2[N]$ corresponding to the two messages m_1 and m_2 sent to the target application. When the attacher sends message m_2, the target application in containers waits an additional $1ms$ before sending back the answer. So the application's secret is $T_2 - T_1 = 1ms$. The attacker sets the null hypothesis as $T_2 - T_1 \leq 1ms$. The attacker then performs the hypothesis test $H(X, Y)$ by randomly taking n samples from X and Y and comparing the median value of the samples. The number of samples n is set to be $n = \{2^i, i = 1, ..., 11\}$. To *reject* the

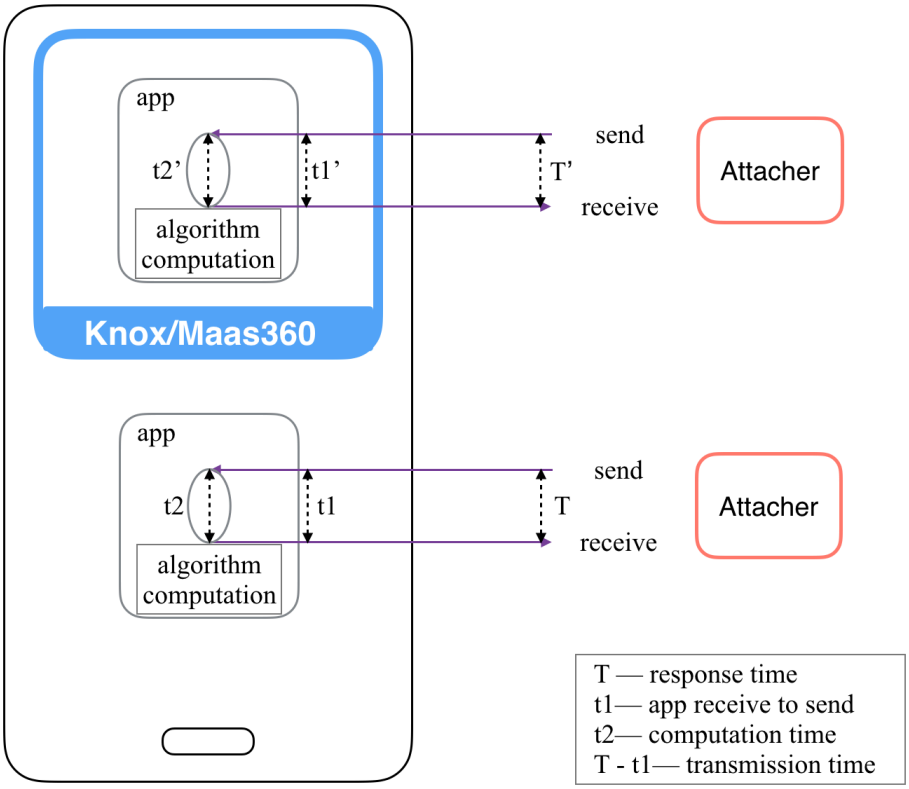


Figure 6.7: Illustration of the experiment setup

null hypothesis means that H believes that X and Y are from different distributions, which is not true. If the attacker correctly reject the null hypothesis, we call it a successful attack. We evaluate the false positive (FP) rate α for the attacker. We perform 1000 trials of randomly picked n samples from the measured response time X and Y and count how many times the test $H(X, Y)$ mistakenly accepts the null hypothesis.

As shown in Figure 6.7, in the target application, we also measure the time between receiving the messages and sending back the answer (t1) and the algorithm computation time (t2), because we want to compare the execution time of the application simulating a RSA algorithm in each container.

6.4.4 Results

First, we measure the response time of the target application and investigate how T changes with different containers. The histogram of response time, local execution time and transmission time

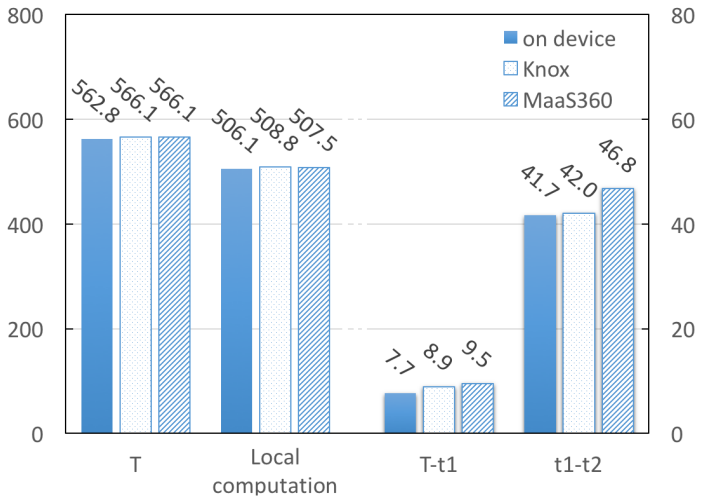


Figure 6.8: Response time, local execution time and transmission time for different secure containers on Samsung S5 device (ms)

of the target application on Samsung S5 device is shown in Figure 6.8. One can see that the mean response time (T) of the application in Knox and MaaS360 containers is $3.3ms$ (0.58%) longer than outside the containers. When the RSA algorithm is executing locally in the Knox container, it takes $2.7ms$ (0.53%) longer than outside the containers, but in the MaaS360 container it takes $1.4ms$ (0.28%) longer. So the Knox container impairs performance most. The attacking client is connected to the mobile device directly through WiFi so the transmission time (T - t1) is correspondingly small. The transmission time to the application in these two containers (8.9 ms and 9.5 ms) is also longer than on the device directly (7.7 ms) because the message must go through several layers of the container before it is transmitted to the target application. The relative growth rates are 15.6% and 23.4%, respectively.

Figure 6.9 shows how the variance of the response time measured by the attacking client varies with the number of samples s. From the figure we find that the number of samples required to reach a stable response time is very small. The attacker only needs about 9 samples to obtain a variation of under $0.5ms$ ($\approx 0.09\%$), well under that to perform a successful attack. When the target application is running outside the containers (on device), it requires 7 samples for the response time to reach convergence. However, in the Knox and MaaS360 containers, to obtain a stable response time, needs 9 samples and 8 samples respectively. So the attacker needs to make more effort to attack the containers on the mobile device, which means the container solutions can improve the security

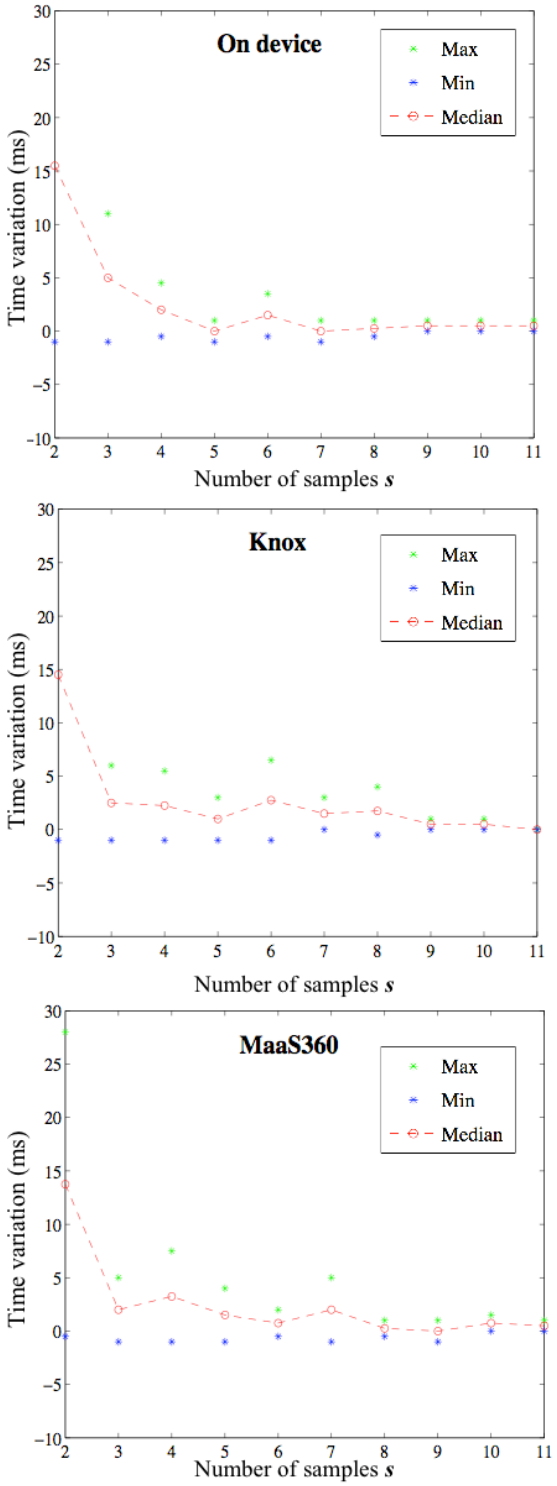


 Figure 6.9: The response time variance decrease as we increase the number of samples s

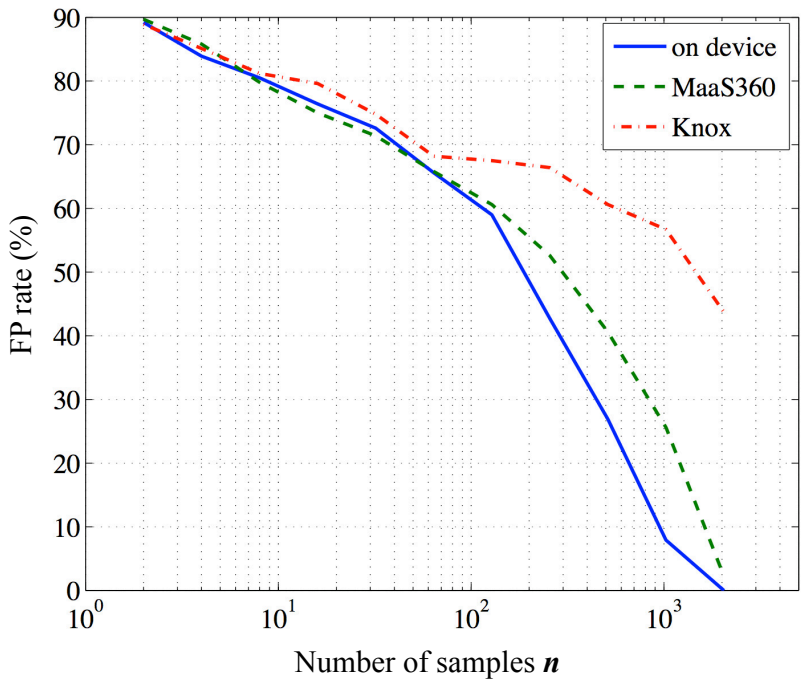


Figure 6.10: The empirical false positive (FP) rate α for the attacker changing with number of samples n

of the mobile applications. It is worth noting that this stable response time is not the final result. Using Brumley and Boneh's method, the attacker has to repeat this process n times to estimate the response time. So it requires a total number of $s \times n$ samples to make a successful guess.

To further quantitatively assess the security attribute of the container solutions, we use the hypothesis testing procedure proposed earlier in this section to compute the empirical false positive (FP) rate α for the attacker. The result is shown in Figure 6.10. One can see that for all n, the Knox container nearly always has the highest FP rate, i.e., it is most difficult for the attacker to make the right guesses from the response time measurements when the application is running in the Knox container. As for the MaaS360 container, when the number of samples n is greater than 100, we see that the attacker needs more samples than in the on device situation to make a right guess. As shown by the dashed line in Figure 6.10, the attacker requires 190, 300 and 1320 samples to get a FP rate of 50% when the target application is running on the device, in the MaaS360 container and in the Knox container, respectively. We assume that with an error detection and correction strategy [23], if a FP rate of 50% it is still possible for the attacker to make a successful guess. Then $MTTSF$ can

Table 6.2: The tradeoff analysis

	Loss in T (ms) / relative rate	Loss in execution time (ms) / relative rate	Gain in MTTSF (min) / relative rate
Knox	3.3 / 0.58%	2.7 / 0.53%	109.6 / 878%
MaaS360	3.3 / 0.58%	1.4 / 0.28%	10.2 / 81.4%

be computed as:

$$MTTSF = T \times s \times n \ , \tag{6.1}$$

where s is the number of samples to obtain a stable response time and n is the number of samples to make a successful guess.

The tradeoff analysis results are shown in Table 6.2. When the target application is running in the Knox container, the security gain is that MTTSF is 109.6 min (878%) longer than without containers, but the performance impairment is that the mean response time T is 3.3 ms (0.58%) longer. As for MaaS360 container, with an overhead of 3.3 ms (0.58%) in T, the MTTSF is extended by 10.2 min (81.4%). The performance loss also includes an additional 2.7 ms (0.53%) of algorithm execution time in the Knox container and 1.4 ms (0.28%) in the MaaS360 container. So in our case, the Knox container is the most secure solution in the presence of timing attacks, and running in the MaaS30 container is still more secure for an application than just running on the mobile device. But the users have to take the performance impairments into consideration when employing secure container solutions.

6.5 Summary

Secure containers are a promising solution to BYOD security concerns. In this chapter, we have presented an empirical approach for the assessment of the security and performance features in the paradigm of secure containers on Android platforms. Each aspect was demonstrated through comparing two popular deployed Android secure containers: Samsung Knox and IBM MaaS360.

We have conducted benchmark tests to the container solutions. To quantitatively assess the security attributes, we have set up a testbed that evaluates the number of samples a timing attacker needs to identify an empirical resolution of $1ms$ in the Android application response time. Through the

tradeoff analysis, it is found that when the target application is running in the Knox container, the security is enhanced and the MTTSF is 109.6 min (878%) longer than without containers. However, the performance impairment is that the mean response time T is 3.3 ms (0.58%) longer. As for MaaS360 container, with an overhead of 3.3 ms (0.58%) in mean response time, the MTTSF is extended by 10.2 min (81.4%). We also found that the performance of compute-intensive applications running in the Knox container degrades strongly, while the performance of memory-intensive applications is not seriously affected in both containers. So the Knox container can provide more security enhancement against timing attacks than the MaaS360 container. However one has to take the performance overhead into account, when using the Knox containers to protect his compute-intensive applications.

Chapter 7

Conclusions and Outlook

7.1 Conclusions

In Mobile Cloud Computing (MCC), offloading is a promising solution to overcome current restrictions of mobile systems, with migrating heavy workloads to remote servers. However, the benefits of the offloading technique are not free as it exhibits costs in terms of security. In this thesis we have studied both the theoretical and practical aspects of offloading policies for MCC systems. The main goal has been to make quantitative assessment of the security and performance attributes and to propose secure and cost-efficient offloading policies based on tradeoff analysis to satisfy the system requirements. We discuss contributions as follows:

- *Quantitative security assessment*: through several proposed stochastic model based approaches, we provide solutions to quantitatively assess the security attributes of the mobile cloud offloading system.
- *Security and performance tradeoff*: by proposing a hybrid CTMC and queueing model, we show methods to formulate metrics that include both, performance and security aspects and that optimise the tradeoff between the two. By solving the hybrid model, the optimal rekeying rate is determined for the system metrics, which can be a guideline for configuring offloading systems. We found that with carefully selected parameters, one can configure the offloading system to achieve an optimal security and performance tradeoff.
- *Secure and cost-efficient offloading policy*: through combining renewing the server key regularly with inserting random delays into the server processing time, we propose a secure and cost-efficient offloading scheme for MCC. Our experimental results show that the security performance tradeoff of offloading can be improved through our scheme.

We implement a system that allows us to compare the impact of different random padding strategies on the expected success of timing attacks. We tune the mean and the variance of the random paddings and investigate the impact on mitigating the time side-channel information leakage. It is revealed that the variance of random delays is the decisive factor to mitigation effectiveness of a random padding and the extra number of measurements an attacker has to make grows linearly with the standard deviation (SD) of the random padding. So when random delays are deployed in mobile cloud offloading systems, one should tune the parameters to enlarge the variance while keeping the mean as low as possible.

- *Improving Client security*: with respect to client security issues, we perform an empirical comparison between two popular secure container solutions for Android: Samsung Knox and IBM MaaS360. Our experimental results shows that the Knox container can provide more security enhancement against timing attacks than the MaaS360 container. We also found that mobile devices witness a notable deterioration in the performance of compute-intensive applications using the Knox container. However, for a memory-intensive application, performance does not degenerate much in Knox and MaaS360 containers compared with running on the device (outside the container).

7.2 Outlook

The main goal of using the proposed offloading policy is to satisfy the security and performance requirements of the MCC system. Although several methods and approaches have been proposed, considering the specific application environment, the following questions are expected to be investigated:

(1) We may consider extension of the theoretical work on system modeling for offloading systems. In our models, we only considered there is one attacker in the system at one time and all the jobs are the same. Our results hold for a generic offloading system. But in the real scenario, the system may involve several kinds of jobs and there may be more than one attacker in the offloading system. So we could conduct validation based on real workloads and more realistic application examples to gain insights about efficiency of the proposed offloading policies.

(2) In order to improve the offloading security, renewing the server key regularly is employed in the proposed offloading policy. But this is a very simple key refresh protocol and may not be efficient when implemented in practical systems. So extending the analysis to include a efficient key refresh protocol will be the future work.

(3) In the model analysis, we have evaluated the performance and security metrics based on the

steady-state distributions. A more functional dynamic model is expected to investigate the dynamic security assessment of the offloading systems.

(4) Authentication of respondent devices is another important security challenge existing in the mobile cloud offloading scenario. We could extend our research to consider the timing attack threat together with authentication issues. At the same time, the MTTSF parameter is sometimes hard to measure in practical systems. Therefore, another objective of our future work is to investigate the way how to estimate and measure the MTTSF.

(5) Multi-site offloading is another interesting research topic as it is possible for mobile devices to save more time and energy by offloading to several cloud service providers. In the future we plan to extend our offloading server to a multi-site system to investigate whether simple management directives for multi-site offloading can be derived. We would also study how the timing attack will act in the multi-site offloading scenario.

Appendix A

Continuous Distribution Functions

Here we list the probability density function and some important properties of the continuous probability distributions used in this thesis.

A.1 The Uniform Distribution

A continuous random variable X that is equally likely to take any value in a range of values (a, b), with $a < b$, gives rise to the *uniform* distribution. Such a distribution is uniformly distributed on its range. The probability density function of X is given as

$$f_X(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise.} \end{cases} \tag{A.1}$$

The mean of the uniform distribution is obtained as

$$E[X] = \int_{-\infty}^{\infty} x f(x)\, dx = \frac{a+b}{2}. \tag{A.2}$$

Its variance is computed from

$$\mathrm{Var}[X] = E[X^2] - (E[X])^2 = \frac{(b-a)^2}{12} \tag{A.3}$$

A.2 The Normal Distribution

The *normal* (or *Gaussian*) distribution is a very common continuous probability distribution. If X is a Gaussian random variable then its probability density function $f_X(x)$, sometimes denoted by $N(\mu, \sigma^2)$, is

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}. \tag{A.4}$$

The mean of the normal distribution is $E[X] = \mu$ and the variance is $\text{Var}[X] = \sigma^2$.

A.3 The Exponential Distribution

The *exponential* distribution is the probability distribution that describes the time between events in a *Poisson* process. The probability density function for an exponential random variable, X , with parameter $\lambda > 0$, is given by

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{A.5}$$

The mean of the exponential distribution is $E[X] = \lambda^{-1}$ and the variance is $\text{Var}[X] = \lambda^{-2}$. One of the most important properties of the exponential distribution is that it possesses the *memoryless* property. This means when the distribution of a "waiting time" until a certain event does not depend on how much time has elapsed already.

A.4 The Erlang Distribution

The *Erlang* distribution is a continuous probability distribution with two parameters: a positive integer 'shape' k and a positive real 'rate' λ. The probability density function for a Erlang distributed random variable X is

$$f_X(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \qquad x > 0. \tag{A.6}$$

The mean of the Erlang distribution is $E[X] = k/\lambda$ and the variance is $\text{Var}[X] = k/\lambda^2$. The Erlang distribution with shape parameter $k = 1$ simplifies to the exponential distribution.

A.5 The Weibull Distribution

The *Weibull* distribution is also a continuous probability distribution that has two parameters: a 'scale' parameter η and a 'shape' parameter k. The probability density function of a Weibull random variable X is

$$f_X(x) = \begin{cases} \frac{k}{\eta}(\frac{x}{\eta})^{k-1} e^{-(\frac{x}{\eta})^k} & x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{A.7}$$

The mean and variance of the Weibull distribution can be expressed as

$$E[X] = \eta\Gamma\left(1 + \frac{1}{k}\right), \tag{A.8}$$

and

$$\mathrm{Var}[X] = \eta^2 \left[\Gamma\left(1 + \frac{2}{k}\right) - \left(\Gamma\left(1 + \frac{1}{k}\right)\right)^2\right] \tag{A.9}$$

Bibliography

[1] BYOD, IoT and wearables thriving in the enterprise. http://www.techproresearch.com/article/byod-iot-and-wearables-thriving-in-the-enterprise/, 2016.

[2] eduroam – World Wide Education Roaming for Research & Education. https://www.eduroam.org/, 2016.

[3] Geekbench 4. http://geekbench.com/, 2016.

[4] IBM MaaS360 Mobile Application Security. http://www-03.ibm.com/software/products/en/maas360-mobile-application-security, 2017.

[5] iOS Security Guide - Apple. https://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2017.

[6] Open Web Application Security Project (OWASP), 2017.

[7] Samsung Knox. https://www.samsungknox.com/en, 2017.

[8] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In *Cryptographers' Track at the RSA Conference*, pages 225–242. Springer, 2007.

[9] O. Aciiçmez, W. Schindler, and Ç. K. Koç. Improving Brumley and Boneh timing attack on unprotected SSL implementations. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 139–146. ACM, 2005.

[10] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: a virtual mobile smartphone architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 173–187. ACM, 2011.

[11] ARM. ARM Security Technology Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009.

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patter-

son, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[13] A. Askarov, D. Zhang, and A. C. Myers. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 297–307. ACM, 2010.

[14] M. Barbera, S. Kosta, A. Mei, and J. Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 1285–1293. IEEE, 2013.

[15] S. Belaïd, V. Grosso, and F.-X. Standaert. Masking and leakage-resilient primitives: One, the other (s) or both? *Cryptography and Communications*, 7(1):163–184, 2014.

[16] R. Bhadauria, R. Chaki, N. Chaki, and S. Sanyal. A survey on security issues in cloud computing. *IEEE Communications Surveys and Tutorials*, pages 1–15, 2011.

[17] BlackBerry. Secure work space for iOS and Android - blackberry. http://de.blackberry.com/content/dam/blackBerry/pdf/business/english/bes10/BES10-2-SWS-EMM-data-sheet.pdf.

[18] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 621–628. ACM, 2007.

[19] B. A. Braun, S. Jana, and D. Boneh. Robust and efficient elimination of cache and timing side channels. *arXiv preprint arXiv:1506.00189*, 2015.

[20] B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. In *Computer Security–ESORICS 2011*, pages 355–371. Springer, 2011.

[21] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[22] M. Cagalj, T. Perkovic, and M. Bugaric. Timing attacks on cognitive authentication schemes. *IEEE Transactions on Information Forensics and Security*, 10(3):584–596, 2015.

[23] C. Chen, T. Wang, and J. Tian. Improving timing attack on RSA-CRT via error detection and correction strategy. *Information Sciences*, 232:464–474, 2013.

[24] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):795–809, 2004.

[25] X. Chen. Decentralized computation offloading game for mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 26(4):974–983, 2015.

[26] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communi-

cation in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.

[27] J.-H. Cho, R. Chen, and P.-G. Feng. Performance analysis of dynamic group communication systems with intrusion detection integrated with batch rekeying in mobile ad hoc networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 644–649. IEEE, 2008.

[28] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[29] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 7. ACM, 2010.

[30] Cisco. Cisco MDS 9000 family storage media encryption configuration guide. http://www.cisco.com/c/en/us/td/docs/switches/datacenter/mds9000/sw/6_2/configuration/guides/sme/smebook.html, 2013.

[31] C. Clavier, J.-S. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems—CHES 2000*, pages 252–263. Springer, 2000.

[32] D. Cock, Q. Ge, T. Murray, and G. Heiser. The last mile: An empirical study of timing channels on sel4. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 570–581. ACM, 2014.

[33] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[34] J.-S. Coron and I. Kizhvatov. An efficient method for random delay generation in embedded software. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 156–170. Springer, 2009.

[35] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):17, 2009.

[36] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[37] H. Cui, X. Yuan, and C. Wang. Harnessing encrypted data in cloud for secure and efficient image sharing from mobile devices. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2659–2667. IEEE, 2015.

[38] T. E. Danford and S. K. Batchu. Virtual instance architecture for mobile device management systems, Nov. 15 2011. US Patent 8,060,074.

[39] P. de las Cuevas, A. Mora, J. Merelo, P. Castillo, P. García-Sánchez, and A. Fernández-Ares. Corporate security solutions for BYOD: A novel user-centric and self-adaptive system. *Computer Communications*, 68:83–95, 2015.

[40] S. Deng, L. Huang, J. Taheri, and A. Zomaya. Computation offloading for service workflow in mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 26(12):3317–3329, Dec 2015.

[41] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya. Computation offloading for service workflow in mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3317–3329, 2015.

[42] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, pages 167–182. Springer, 1998.

[43] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[44] G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. Cacheaudit: A tool for the static analysis of cache side channels. In *Usenix Security*, pages 431–446, 2013.

[45] G. Doychev and B. Köpf. Rational protection against timing attacks. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 526–536. IEEE, 2015.

[46] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu. Frequent subgraph based familial classification of android malware. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 24–35. IEEE, 2016.

[47] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[48] P. M. Frank. *Introduction to system sensitivity theory*, volume 11. Academic press New York, 1978.

[49] R. Gabner, H.-P. Schwefel, K. A. Hummel, and G. Haring. Optimal model-based policies for component migration of mobile cloud services. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium On*, pages 195–202. IEEE, 2011.

[50] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–

1171, 2010.

[51] Google. Android security white paper. https://static.googleusercontent.com/media/www.google.co.il/iw/IL/work/android/files/android-for-work-security-white-paper.pdf, May 2015.

[52] P. Gupta and S. Gupta. Mobile cloud computing: the future of cloud. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 1(3):134–145, 2012.

[53] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *USENIX Security Symposium*, 2011.

[54] Z. Hao, Y. Tang, Y. Zhang, E. Novak, N. Carter, and Q. Li. SMOC: A secure mobile cloud computing platform. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2668–2676. IEEE, 2015.

[55] B. R. Haverkort. *Performance of computer communication systems: a model-based approach*. Wiley, 1998.

[56] Z. He, X. Deng, B. Yang, K. Dai, and X. Zou. A SCA-resistant processor architecture based on random delay insertion. In *Computing and Communications Technologies (ICCCT), 2015 International Conference on*, pages 278–281. IEEE, 2015.

[57] J. I. Hong and J. A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2):287–303, 2001.

[58] G. Huerta-Canepa and D. Lee. An adaptable application offloading scheme based on application behavior. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 387–392. IEEE, 2008.

[59] E. Hyytiä, T. Spyropoulos, and J. Ott. Offload (only) the right jobs: Robust offloading using the markov decision processes. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pages 1–9. IEEE, 2015.

[60] A. Iacovazzi and A. Baiocchi. Internet traffic privacy enhancement with masking: Optimization and tradeoffs. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):353–362, 2014.

[61] IBM. Cloud extender architecture. http://www.ibm.com/support/knowledgecenter/SS8H2S/com.ibm.mc.doc/ce_source/references/ce_architecture.htm.

[62] ISO. ISO/IEC 27000:2016(E). http://standards.iso.org/ittf/PubliclyAvailableStandards/, 2016.

[63] A. K. Jha and W. J. Lee. Analysis of Permission-based Security in Android through Pol-

icy Expert, Developer, and End User Perspectives. *Journal of Universal Computer Science*, 22(4):459–474, 2016.

[64] P. Jindal and B. Singh. Performance evaluation of security-throughput tradeoff with channel adaptive encryption. *International Journal of Computer Network and Information Security*, 5(1):49, 2013.

[65] U. Kanonov and A. Wool. Secure Containers in Android: the Samsung KNOX Case Study. *arXiv preprint arXiv:1605.08567*, 2016.

[66] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a computation offloading framework for smartphones. In *International Conference on Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2010.

[67] I. Khalil, A. Khreishah, and M. Azeem. Consolidated identity management system for secure mobile cloud computing. *Computer Networks*, 65:99–110, 2014.

[68] A. N. Khan, M. M. Kiah, S. U. Khan, and S. A. Madani. Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(5):1278–1299, 2013.

[69] M. A. Khan. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *Journal of Network and Computer Applications*, 56:28–40, 2015.

[70] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[71] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer, 1996.

[72] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 286–296. ACM, 2007.

[73] B. Köpf and D. Basin. Automatically deriving information-theoretic bounds for adaptive side-channel attacks. *Journal of Computer Security*, 19(1):1–31, 2011.

[74] B. Köpf and M. Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE*, pages 324–335. IEEE, 2009.

[75] B. A. Köpf et al. *Formal approaches to countering side-channel attacks*. PhD thesis, ETH, 2007.

[76] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[77] S. Kotipalli, Y.-B. Kim, and M. Choi. Asynchronous advanced encryption standard hardware with random noise injection for improved side-channel attack resistance. *Journal of Electrical and Computer Engineering*, 2014:19, 2014.

[78] D. Kovachev. Framework for computation offloading in mobile cloud computing. *IJIMAI*, 1(7):6–15, 2012.

[79] D. Kovachev, Y. Cao, and R. Klamma. Mobile cloud computing: a comparison of application models. *arXiv preprint arXiv:1107.4940*, 2011.

[80] M. G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 3–18. IEEE, 2002.

[81] M. G. Kuhn and R. J. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In *International Workshop on Information Hiding*, pages 124–142. Springer, 1998.

[82] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[83] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[84] B. W. Lampson. Hints for computer system design. In *ACM SIGOPS Operating Systems Review*, volume 17, pages 33–48. ACM, 1983.

[85] K. Lee and I. Shin. User mobility-aware decision making for mobile computation offloading. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*, pages 116–119. IEEE, 2013.

[86] S. R. Lenkala, S. Shetty, and K. Xiong. Security Risk Assessment of Cloud Carrier. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 442–449. IEEE, 2013.

[87] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246. ACM, 2001.

[88] N. Limnios and G. Oprisan. *Semi-Markov processes and reliability*. Springer Science & Business Media, 2001.

[89] J. D. Little. A proof for the queuing formula: L= λ w. *Operations research*, 9(3):383–387, 1961.

[90] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid, and D. Gollmann. Towards operational measures of computer security. *Journal of Computer Security*, 2(2):211–229, 1993.

[91] Y. Lu, M. P. O'Neill, and J. V. McCanny. FPGA implementation and analysis of random delay insertion countermeasure against DPA. In *ICECE Technology, 2008. FPT 2008. International Conference on*, pages 201–208. IEEE, 2008.

[92] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Performance Evaluation*, 56(1):167–186, 2004.

[93] T. T. Mapoka, S. J. Shepherd, and R. A. Abd-Alhameed. A new multiple service key management scheme for secure wireless mobile multicast. *IEEE Trans. Mob. Comput.*, 14(8):1545–1559, 2015.

[94] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. https://www.cs.virginia.edu/stream/, 2016.

[95] T. Meng, X. Li, S. Zhang, and Y. Zhao. A Hybrid Secure Scheme for Wireless Sensor Networks against Timing Attacks Using Continuous-Time Markov Chain and Queueing Model. *Sensors*, 16(10):1606, 2016.

[96] T. Meng, Q. Wang, and K. Wolter. Model-Based Quantitative Security Analysis of Mobile Offloading Systems Under Timing Attacks. In *Analytical and Stochastic Modelling Techniques and Applications*, pages 143–157. Springer, 2015.

[97] T. Meng and K. Wolter. Analysis of Mitigation Measures for Timing Attacks in Mobile-Cloud Offloading Systems. In *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 168–182. Springer, 2016.

[98] T. Meng, K. Wolter, and Q. Wang. Security and Performance Tradeoff Analysis of Mobile Offloading Systems Under Timing Attacks. In *Computer Performance Engineering*, pages 32–46. Springer, 2015.

[99] F. P. Miller, A. F. Vandome, and J. McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.

[100] K. W. Miller, J. M. Voas, and G. F. Hurlburt. BYOD: Security and Privacy Considerations. *It Professional*, 14(5):53–55, 2012.

[101] D. L. Nazareth and J. Choi. A system dynamics model for information security management. *Information & Management*, 52(1):123–134, 2015.

[102] M. F. Neuts. *Matrix-geometric solutions in stochastic models: an algorithmic approach*. Courier Corporation, 1981.

[103] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: from dependability to security. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):48–65, 2004.

[104] J. Niu, W. Song, and M. Atiquzzaman. Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications. *Journal of Network and Computer Applications*, 37:334–347, 2014.

[105] S. Ou, Y. Wu, K. Yang, and B. Zhou. Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments. In *Communications, 2008. ICC'08. IEEE International Conference On*, pages 1856–1860. IEEE, 2008.

[106] S. Ou, K. Yang, A. Liotta, and L. Hu. Performance analysis of offloading systems in mobile wireless environments. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1821–1826. IEEE, 2007.

[107] D. Potts, G. Steidl, and M. Tasche. Fast Fourier transforms for nonequispaced data: A tutorial. In *Modern sampling theory*, pages 247–270. Springer, 2001.

[108] H. Qian and D. Andresen. Reducing mobile device energy consumption with computation offloading. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*, pages 1–8, June 2015.

[109] H. Qian and D. Andresen. Automate scientific workflow execution between local cluster and cloud. *the International Journal of Networked and Distributed Computing (IJNDC), Atlantis press*, 2016.

[110] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 73–84. ACM, 2011.

[111] C. Rebeiro, D. Mukhopadhyay, and S. Bhattacharya. An introduction to Timing Attacks. In *Timing Channels in Cryptography*, pages 1–11. Springer, 2015.

[112] H. Rim, S. Kim, Y. Kim, and H. Han. Transparent method offloading for slim execution. In *Wireless Pervasive Computing, 2006 1st International Symposium on*, pages 1–6. IEEE, 2006.

[113] Samsung. Samsung Knox Approved by Department of Defense for use in US Government. http://www.samsung.com/sg/business/insights/news/samsung-knox-approved-by-department-of-defense-for-use-in-us-government, 2013.

[114] Samsung. White Paper: An Overview of the Samsung KNOX Platform. http://www.samsung.com/es/business-images/resource/white-paper/2014/02/Samsung_KNOX_whitepaper-0.pdf, November 2015.

[115] W. Schindler. A timing attack against RSA with the chinese remainder theorem. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 109–124. Springer, 2000.

[116] W. Schindler. Optimized timing attacks against public key cryptosystems, 2002.

[117] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, Nov. 23 1999. US Patent 5,991,415.

[118] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad. Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing*, 13(1):1–18, 2015.

[119] K. Sinha and M. Kulkarni. Techniques for fine-grained, multi-site computation offloading. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 184–194. IEEE Computer Society, 2011.

[120] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security Symposium*, volume 2001, 2001.

[121] W. J. Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[122] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[123] I. A. Sumra, J.-L. Ab Manan, and H. Hasbullah. Timing attack in vehicular network. In *Proceedings of the 15th WSEAS International Conference on Computers, World Scientific and Engineering Academy and Society (WSEAS), Corfu Island, Greece*, pages 151–155, 2011.

[124] K. S. Trivedi. *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons, 2008.

[125] D. Vecchiato, M. Vieira, and E. Martins. Risk Assessment of User-Defined Security Configurations for Android Devices. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 467–477. IEEE, 2016.

[126] C. Wang and Z. Li. A computation offloading scheme on handheld devices. *Journal of Parallel and Distributed Computing*, 64(6):740–746, 2004.

[127] E. K. Wang, T.-Y. Wu, C.-M. Chen, Y. Ye, Z. Zhang, and F. Zou. MDPAS: Markov Decision Process Based Adaptive Security for Sensors in Internet of Things. In *Genetic and Evolutionary Computing*, pages 389–397. Springer, 2015.

[128] Q. Wang. *Restart in Mobile Offloading*. PhD thesis, Freie Universität Berlin, 2015.

[129] Q. Wang and K. Wolter. Automated adaptive restart for accelerating task completion in cloud offloading systems. In *Autonomic Computing (ICAC), 2015 IEEE International Conference*

on, pages 157–158. IEEE, 2015.

[130] Q. Wang and K. Wolter. Reducing task completion time in mobile offloading systems through online adaptive local restart. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, pages 3–13, New York, NY, USA, 2015. ACM.

[131] S. Wang and S. Dey. Rendering adaptation to address communication and computation constraints in cloud mobile gaming. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[132] X. Wang, K. Sun, Y. Wang, and J. Jing. DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices. In *NDSS*, 2015.

[133] M. Weiss, B. Heinz, and F. Stumpf. A cache timing attack on AES in virtualization environments. In *Financial Cryptography and Data Security*, pages 314–328. Springer, 2012.

[134] K. Wolter and P. Reinecke. Performance and security tradeoff. In *Formal methods for quantitative aspects of programming languages*, pages 135–167. Springer, 2010.

[135] H. Wu. *Analysis of offloading decision making in mobile cloud computing*. PhD thesis, Freie Universität Berlin, 2015.

[136] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun. An optimal offloading partitioning algorithm in mobile cloud computing. In *International Conference on Quantitative Evaluation of Systems*, pages 311–328. Springer, 2016.

[137] H. Wu, Q. Wang, and K. Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 IEEE International Conference on Communications Workshops (ICC)*, pages 728–732. IEEE, 2013.

[138] H. Wu and K. Wolter. Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pages 90–97. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[139] C. Xian, Y.-H. Lu, and Z. Li. Adaptive computation offloading for energy conservation on battery-powered systems. In *Parallel and Distributed Systems, 2007 International Conference on*, volume 2, pages 1–8. IEEE, 2007.

[140] K. Yang, S. Ou, and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE communications magazine*, 46(1), 2008.

[141] D. Zhang, A. Askarov, and A. C. Myers. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 563–574. ACM, 2011.

[142] J.-f. Zhang, F. Liu, L.-m. Zheng, Y. Jia, and P. Zou. Using Network Security Index System to Evaluate Network Security. In E. Qi, J. Shen, and R. Dou, editors, *The 19th International Conference on Industrial Engineering and Engineering Management*, pages 989–1000. Springer Berlin Heidelberg, 2013.

[143] Y. Zhao and N. Thomas. Efficient solutions of a PEPA model of a key distribution centre. *Performance Evaluation*, 67(8):740–756, 2010.

List of Figures

List of Tables

List of Publications

Meng, T., Wolter, K., & Wang, Q. Security and Performance Tradeoff Analysis of Mobile Offloading Systems Under Timing Attacks. In *Computer Performance Engineering (pp. 32-46)*. Springer International Publishing, 2015

Meng, T., Wang, Q., & Wolter, K. Model-Based Quantitative Security Analysis of Mobile Offloading Systems Under Timing Attacks. In *Analytical and Stochastic Modelling Techniques and Applications (pp. 143-157)*. Springer International Publishing, 2015

Meng, T., & Wolter, K. Analysis of Mitigation Measures for Timing Attacks in Mobile-Cloud Offloading Systems. In *Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems (pp. 168-182)*. Springer International Publishing, 2016

Meng, T., Li, X., Zhang, S., & Zhao, Y. A Hybrid Secure Scheme for Wireless Sensor Networks against Timing Attacks Using Continuous-Time Markov Chain and Queueing Model. *Sensors*, 16(10), 1606. 2016.

Meng, T., Shang, Z., & Wolter, K. An Empirical Performance and Security Evaluation of Android Container Solutions. In *Cyber Security And Protection Of Digital Services (Cyber Security), 2017 International Conference On (pp. 1-8)*. IEEE, 2017.

Meng, T., Wolter, K., Wu, H., & Wang, Q. A Secure and Cost-efficient Offloading Policy for Mobile Cloud Computing against Timing Attacks. *revision in Pervasive and Mobile Computing*, 2017.

About the Author

Tianhui Meng received both his Bachelor's and Master's degrees in Information and Communication Engineering from Tianjin University, China in 2010 and 2013, respectively. Supported by China Scholarship Council (CSC), he worked as a doctoral candidate at Computer Systems & Telematics (CST) group at Freie Universität Berlin, Germany, supervised by Prof. Dr. Katinka Wolter since 2013. His current research interests include security of wireless and mobile network systems, mobile cloud computing and stochastic modeling.

Zusammenfassung

Während die letzten Jahrzehnte große Fortschritte der Hardware-Technologie erlebt haben, ist die Nachfrage nach neuen Applikationen viel größer geworden. Trotzdem sehen sich mobile Geräte immer noch mit Beschränkungen der Ressourcen konfrontiert sowie Batterielebensdauer, Speicherkapazität und Prozessorleistung. Im Feld Mobile Cloud Computing (MCC) ist Offloading eine populäre Technik, die aufgestellt wird, um die Kapazitäten von mobilen Systemen zu erweitern, indem sie komplexe Berechnungen auf ressourcenreiche Cloud-Server erleichtert. Zwar ist Offloading aus den Leistungs- und Energieperspektiven vortelhaft sein kann, stellt es aufgrund der erhöhten Datenübertragung über Netzwerke mit potenziellen unbekannten Bedrohungen sicherlich neue Herausforderungen dar.

Zu den möglichen Sicherheitsfragen gehören Timing-Attacken, die die traditionelle kryptographische Sicherheit nicht verhindern kann. Timing-Attacken gehören zu Side-Channel-Attacken, in denen der Angreifer versucht, ein System zu kompromittieren, indem er die Zeit analysiert, die das System benötigt, um auf verschiedene Abfragen zu antworten. Offloading ist besonders anfällig für Timing-Attacken, weil es viele Male senden / empfangen muss. Die Metriken von Offloading müssen neben den Leistungs- und Energieperspektiven auch Sicherheitsaspekte beinhalten. Diese Theorie behandelt sowohl theoretischen als auch praktischen Aspekte der Richtlinien von Offloading in MCC-System. Ganz anders als die früheren Arbeiten, die nur die Leistungs- und Energieperspektiven berücksichtigen, werden die Offloading-Richtlinien auf der Grundlage der Security-Performance-Tradeoff-Analyse in unsrer Arbeit präsentiert und evaluiert. Vorgeschlagene stochastische Modelle werden durch numerische Simulationen und reale Experimente angewendet und evaluiert. Insbesondere können die Beiträge dieser Arbeit wie folgendes zusammengefasst werden:

- Es werden mehrere stochastische modellbasierte Ansätze zur quantitativen Bewertung der Sicherheits- und Leistungsmerkmale des Mobile Cloud Offloading Systems vorgeschlagen.
- Methoden zur Formulierung von Metriken, die sowohl Leistungs- als auch Sicherheitsaspekte beinhalten und die den Kompromiß zwischen den beiden optimieren, werden untersucht.
- Eine sichere und kostengünstige Offloading Richtlinie unter Berücksichtigung der spezifischen Bedrohung von Timing-Attachen gegen MCC-Systeme wird vorgeschlagen und die Offloading Richtlinie wird mit Experimenten ausgewertet.
- Zwei weitverbreitete sichere Container für Android: Samsung Knox und IBM MaaS360, um die Client-Sicherheit in MCC-Systemen zu verbessern, werden verglichen.