Pervasive and Mobile Computing 45 (2018) 4–18

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

# A secure and cost-efficient offloading policy for Mobile Cloud Computing against timing attacks

Tianhui Meng [a,*], Katinka Wolter [a], Huaming Wu [b], Qiushi Wang [a]

[a] Department of Mathematics and Computer Science, Freie Universität Berlin, Takustr.9, 14195 Berlin, Germany
[b] Center for Applied Mathematics, Tianjin University, 300072 Tianjin, China

ARTICLE INFO

*Article history:*
Received 11 December 2016
Received in revised form 22 September 2017
Accepted 30 January 2018
Available online 12 February 2018

*Keywords:*
Mobile Cloud Computing
Security
Computation offloading
Side-channel attack

ABSTRACT

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 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. 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. This paper considers the specific threat of timing attacks against MCC systems. We present and evaluate a secure and cost-efficient offloading scheme which is the combination of regular rekeying and random padding. In order to proceed to a quantitative treatment, a hybrid Continuous-time Markov chain (CTMC) and queueing model is put forward, and the tradeoff analysis of the security and performance attributes is carried out. We propose security metrics which system architects need to make informed tradeoff decisions involving system security. The numerical results based on experimental data show that the security performance tradeoff 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.


## 1. Introduction

Mobile devices 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, recording health data and social networking. While 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 and network bandwidth. On the other hand, cloud servers are readily to be used by the mobile systems.

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) [1]. This is different from the traditional client–server architecture, where a thin client always migrates computation to a server [2]. In mobile cloud

* Corresponding author.
*E-mail address:* tianhui.meng@fu-berlin.de (T. Meng).

https://doi.org/10.1016/j.pmcj.2018.01.007
1574-1192/

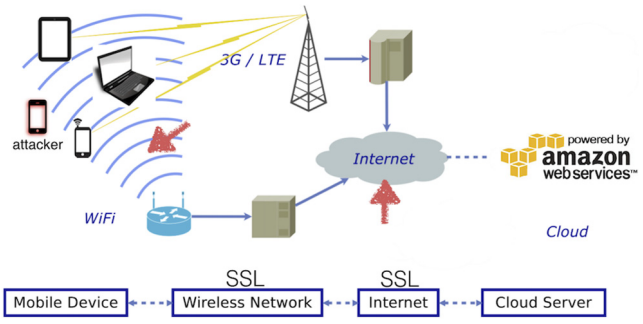



**Fig. 1.** An illustration of mobile cloud offloading system.

offloading, applications are divided into those parts that can be offloaded and those that must be executed on the mobile device, such as the user interface [3].

Offloading is an important technique in mobile cloud computing (MCC) [4]. Due to the relatively low battery capacity of mobile devices as well as fragile mobile networks, a significant amount of research has been performed on offloading computation to the cloud to achieve two main objectives: to extend battery lifetime [5–8], and to shorten execution time of heavy applications [4,9–11].

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 [12]. Protecting user privacy and data secrecy from an adversary is a key to establish and maintain consumers' trust in the mobile platform, especially in 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 [13–16].

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 (Fig. 1). We mainly address the server security in the mobile offloading system. A possible threat against the offloading server is the side-channel attack, which is not covered by traditional cryptographic security [17].

The first approach to conduct side-channel attacks in the cloud environment was proposed by Ristenpart [18]. Using the Amazon EC2 service as a case study, they showed that it is possible to mount keystroke timing attacks using cache-based load measurements for cross-VM keystroke monitoring. Among side-channel attacks, timing attacks, whose remote feasibility has been proved in [19], make a practical threat against web services as well as mobile cloud offloading systems [20]. One group from Stanford [21] showed they accomplish to extract RSA private key by measuring the web server response time. Offloading is particularly vulnerable to these timing attacks because it often needs many times sending/receiving.

Quantitative analyses of system security have received attention for several years [22,23]. In 2003, Zhang [24] proposed an approach to evaluate the network security situation objectively using Network Security Index System (NSIS). More recently a number of model-based evaluations of security mechanisms have been published [25–27]. Previous work on the security of computing and information systems has been mostly assessed from a level or rank 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. Few studies have considered the quantitative evaluation of security and the tradeoff with performance [28].

Researchers in [29] 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 [30]. 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 paper goes beyond existing approaches by proposing a combined security strategy of a rekeying mechanism and adding random delays in the encryption execution flow and considering a quantitative treatment of security problem.

In this paper, we propose and evaluate a secure and cost-efficient offloading approach which optimizes the performance and security tradeoff of the system. In order to proceed to a quantitative treatment of the tradeoff problem, a hybrid Continuous-time Markov chain and queueing model is used to model the MCC system which treats security and performance attributes respectively.

The major contributions of this paper are summarized as follows:

- 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 have implemented 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 an attacker needs more samples to conduct a timing attack when random paddings are deployed in the system and the variance of random delays is the decisive factor in mitigating effectiveness of a random padding. The extra number of measurements an attacker has to make grows linearly with the standard deviation (SD) of the random padding.
- We show how to formulate metrics that include both, performance and security aspects and that optimize the tradeoff between the two. By solving the model, optimal rekeying rates are determined for system cost and tradeoff metrics.

The remainder of this paper is structured as follows. In Section 2, we overview the system and timing attacks. A hybrid model for a mobile cloud offloading system under timing attacks is proposed in Section 3. The system security and performance metrics on which the evaluation based are addressed in Section 4. Section 5 gives model analysis of the security and performance attributes using quantitative assessment methods. Experiments and numerical analysis are presented in Section 6. The paper is concluded in Section 7.

## 2. System overview and related work

In this section, we present an overview of the mobile cloud offloading system and timing attacks. We further show the frequently-used countermeasures against timing attacks.

### 2.1. Mobile cloud offloading system overview

The general architecture of a mobile cloud offloading system is depicted in Fig. 1. 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 [31]. 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 mobiles. Real-time multimedia applications, especially real-time strategy game, fitness application are some applications that benefit from this approach. For instance, a health care application is running on a smart watch. The data of medical sensors have to be transited to the cloud server to make a diagnosis or prediction. And the data should be encrypted due to privacy protection.

In the 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 [32].

The considered system is vulnerable to timing attacks in which the attacker in the worst case will eventually guess the system's RSA private key by analyzing the time measurements to the server. If the attacker successfully compromise the system, 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 system throughput.

### 2.2. Timing attacks

Implementations of cryptographic algorithms often perform computations in non-constant time, due to performance optimization [33]. 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 [34].

Fig. 2 shows an illustration of a timing attack. We take Brumley and Boneh's timing attack [21] as an example to show how timing attacks are carried out in practical systems. They accomplish to extract RSA private key by measuring an OpenSSL [35] based web server's response time. In their implementation, a regular client establishes a connection with the cloud server using a three-way handshake. During a handshake the SSL server performs an RSA decryption with its private key after receiving the CLIENT-KEY-EXCHANGE message. And checks the decrypted 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 value for computing a master secret and continues the SSL protocol.

In Brumley and Boneh'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 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

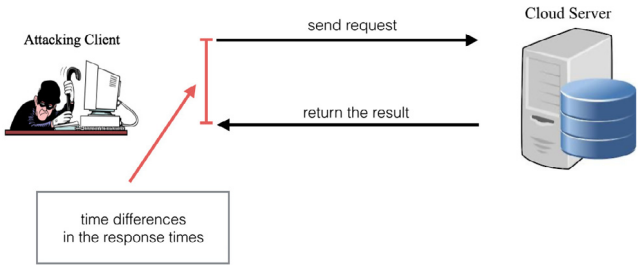



**Fig. 2.** An illustration of a timing attack.

value of $g$. With carefully analyzing the response time measurements, they exploit the difference in the processing time of Montgomery reduction and present algorithms to exact the server's private key bit by bit.

Because timing attacks deduce information about a secret key only from runtime measurements of successive requests, they are real threats to mobile cloud offloading systems. 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, 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 [36] and wireless sensor networks (WSNs) [37].

The key point of timing attacks is to distinguish the timing difference in the processing time, instead of the actual processing time. As we all know, network data are very noisy. Many researchers have worked on building tools to filter the data. Crosby et al. [38] analyzed the limits of attacks based on accurately measuring network response times and jitter induced by the network and by the end hosts. It was shown that with good filtering and carefully statistic analysis, an attacker could distinguish very tiny time difference from the network data. Their filters can significantly reduce the effects of jitter, allowing an attacker to reliably distinguish remote timing differences as low as 20 $\mu$s across the Internet.

### *2.3. Defending 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 [20]. So people also try to hide or mask the timing information. Series of works [39–42] 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 [43,44].

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 [45]. Even though Kocher [46] observed that this technique can be rendered useless by increasing the number of timing measurements, it 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 [47] 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 [48–50].

Blinding technique is another widely deployed countermeasure against timing attacks on cryptosystems. The idea was first introduced by Kocher [46] to adapt the techniques used for blinding signatures to prevent timing attacks. It was improved by Adi Shamir [51] 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 [52]. 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) [53]. A less restrictive way of dealing with timing leaks is to ensure

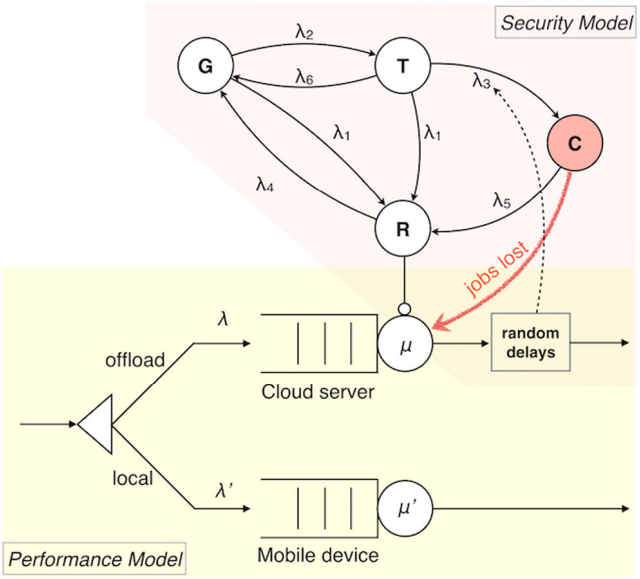



**Fig. 3.** Performance and security model for a generic mobile cloud offloading system.

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 [46].

## 3. The hybrid system model

In this section, we specify the system models and show how we abstract the models from real systems life-time. To quantitatively analyze the performance and security attributes of the 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 system. Our model is aimed to deal with a general mobile cloud offloading system with a master secret (e.g. a Tape volume key) stored in the cloud server, that is used for the encryption and decryption operations. Fig. 3 depicts the diagram of performance and security model made of CTMC and open queueing model which we proposed in [54] for describing dynamic behavior of a mobile offloading system. Different from our previous work, we add random delays in the offloading queue as a countermeasure to mitigate timing attacks. This is a generic model that enables us to take into account the behavior of both actors, the system and the attacker.

In the upper part of Fig. 3 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. Whereas, the lower part of Fig. 3 is the 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.

### 3.1. Attack model

The aim of an attacker is to extract the private key stored in the server which is used to encrypt the communications. The attacker sits in the system as a normal cloud service user as shown in Fig. 1. The attacker is connected to a mobile network via base stations and WiFi access points and then connected to the cloud server through Internet. In timing attacks to the system, the attacker continues to send requests to the server and after the jobs are processed, the results are sent back. In addition the attacker records each response time for a certain query and tries to guess the private key of the server by comparing time differences from several request queues using Brumley and Boneh's timing attack discussed in Section 2.2.

If the attacker successfully obtains the private key from the timing results, he may access the system, read and even modify other users' information without authorization. Obviously, this requires the attacker to spend effort, where we use time to represent the attacking effort. This time is modeled as a random variable which follows a probability distribution. Deterministic, exponential, hypo-exponential, hyper-exponential, gamma, Weibull and log-logistic, etc. are some of the distribution functions that are widely used in the context of security analysis [55]. As it is common for rare events, the attacker arrival time is difficult to simulate or model. We use exponential distribution to model the attacker arrival time and the time a timing attack takes, because exponential distribution is simple and it does not lose generality.

### 3.2. Security model

The upper part of Fig. 3 shows the CTMC model representing the states of the mobile cloud offloading system. From the real system life time, we abstract four operational states for the mobile cloud offloading system. The states of CTMC and the model parameters 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 private key of the system
- $R$ Rekeying state in which the system renews its key
- $\lambda_1$ rate at which the system launches the rekeying process in state $G$ and state $T$
- $\lambda_2$ rate at which an attacker triggers a timing attack to the system
- $\lambda_3$ rate at which a timing attack succeeds to break the private key
- $\lambda_4$ rate at which the system is brought back to the good state by the rekeying process
- $\lambda_5$ rate at which the system launches the rekeying process in state $C$
- $\lambda_6$ rate at which the attacker fails to conduct a successful timing attack.

After initialization, the system starts to operate properly in the good state $G$. The mobile offloading system is under the specific threat of timing attacks conducted by random attackers. When an attacker enters the system, the system is in danger and we define this as the timing attack state $T$. 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 comes, the system is brought to the timing attack state $T$ at rate $\lambda_2$. In this state, the attacker is still conducting the timing attack to get the server key and it is not yet able to access confidential information.

The attacker has to make a effort before he successfully cracks the system secret by timing attack, and the system moves to the compromised state $C$ at rate $\lambda_3$. Consequently the mean time a timing attack takes is represented by $\lambda_3^{-1}$. If the attacker fails to conduct a successful timing attack, the system will go back to the good state $G$ by the arc $\lambda_6$.

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 $C$ and we call the incident of entering the compromised state a *security failure*.

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. The arcs from other states to state $R$ represent these operations in the server. 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 $\lambda_1$ 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 $\lambda_4$.

However this rekeying process is not free. It brings cost to the system as the effort to trigger this rekeying process. Therefore, it is beneficial to recommend an optimal rekeying interval for the private key replacement cycle (or an optimal rekeying rate $\lambda_1$). After the rekeying process, the system is secure and it is brought back to the initial state $G$. The life cycle of the system starts again. 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 6.4.

### 3.3. Performance model

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. We put a inhibitor arc on the cloud server. 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 compromise 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 Fig. 3. In this state, only jobs processed locally on the mobile device contribute to system throughput.

In order to mitigate timing information leakage, random delays are padded for each service response. 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 $\lambda_3$ decreases as this mitigation method is taken. This process is represented by the dashed arc.

The lower part of Fig. 3 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 $\lambda$ and $\lambda'$ 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 $\mu$, which also includes the data transmission time. For jobs dispatched to execute locally, the service rate is $\mu'$ which is assumed to be lower than $\mu$.

## 4. Security and performance metrics

After defining the model and its parameters, we must establish the metrics we want to investigate. Security and performance metrics are presented respectively in this section. In addition to analyzing the metrics independently, the tradeoff between different metrics is also addressed.

### 4.1. Security metrics

As security metrics we define confidentiality and system (security) cost that are functions of the steady-state probability of the CTMC model. The steady-state probabilities $\pi_i$ may be interpreted as the proportion of time that the CTMC spends in state $i$, where $i \in \{R, G, T, C\}$.

If a timing attack to the mobile cloud offloading system is successful, the attacker obtains the private key and can browse unauthorized files thereafter. The entered states denote the loss of confidentiality. Therefore, the steady-state confidentiality metric can be computed as

$$\Delta = 1 - \pi_C \ . \tag{1}$$

We also define a system cost metric. In the considered scenario, 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. The rekeying cost and the data disclosure cost are both the invested time, that is, the steady-state probability of the CTMC in those states. We define a weight $v$ and its complement $1 - v$ for the two kinds of cost. We use normalized weights for simplicity. Mathematically, the system cost is computed as the weighted sum:

$$\begin{aligned} \mathcal{C} &= Cost_{security} + Cost_{rekeying} \\ &= (1-v)\pi_C + v\pi_R \end{aligned} \tag{2}$$

where $\pi_i$ denotes the proportion of time that the continuous-time Markov chain is in state $i$, $i \in \{R, C\}$. $0 \leq v \leq 1$ is the weighting parameter used to share relative importance between the loss of sensitive information and the effort needed to rekey regularly.

### 4.2. Performance metrics

The performance metrics of interest describe the system in terms of throughput, completion time, or response time, as defined in queueing theory or networking. Here we use the throughput (denoted by $X$) as the performance metric for the mobile cloud offloading system. For each queue, the throughput equals the mean number of jobs in the queueing station ($E[N]$) divided by the mean time a job spends in the queueing station ($E[R]$). The system throughput equals the sum of the two queues. We analyze the throughput in detail in Section 5.2

### 4.3. Tradeoff metric

In order to investigate how system security will interact with performance attribute, we define a tradeoff metric as:

$$\mathcal{T} = \Delta \times X \ . \tag{3}$$

The 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, which is the better the higher.

## 5. Computing metrics

In this section, we derive and evaluate the security and performance attributes of the mobile cloud offloading system using methods for quantitative assessment of dependability.

### 5.1. Confidentiality analysis

For the system security attributes, we have described the system's dynamic behavior by a CTMC model with the state space $X_s = \{R, G, T, C\}$ and the transitions between these states. In order to carry out the quantification analysis of

security, we first determine the stationary distribution $(\pi_R, \pi_G, \pi_T, \pi_C)$ of the CTMC model (The computation is shown in the Appendix):

$$\pi_R = \frac{[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi}, \tag{4}$$

$$\pi_G = \frac{(\lambda_1+\lambda_3+\lambda_6)\lambda_4\lambda_5}{\phi},\ \pi_T = \frac{\lambda_2\lambda_4\lambda_5}{\phi},\ \pi_C = \frac{\lambda_2\lambda_3\lambda_4}{\phi},$$

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

Given the steady-state probabilities, the confidentiality metric $\Delta$ and *Cost* metric can be computed via (1) and (2):

$$\Delta = 1 - \frac{\lambda_2\lambda_3\lambda_4}{\phi}, \tag{5}$$

$$C = v\frac{[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi} + (1-v)\frac{\lambda_2\lambda_3\lambda_4}{\phi}. \tag{6}$$

### 5.2. Throughput analysis

Let the total system life time be $T$. The steady-state probabilities $\pi_i$ represent the proportion of time that the CTMC spends in state $i$. In the good state $G$ and timing attack state $T$, the number of jobs served by the system is $\lambda(\pi_G+\pi_T)T+\lambda'(\pi_G+\pi_T)T$, 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 $\mu'\pi_R T$. When the system is compromised by an attacker, the jobs dispatched to offload do not contribute to the system throughput, because they are not secure and must be repeated. In the state $C$, the system throughput only covers the jobs executed locally $\lambda'\pi_C T$. Therefore, the system throughput is

$$\begin{aligned} X &= \frac{\lambda(\pi_G+\pi_T)T + \lambda'(\pi_G+\pi_T)T + \mu'\pi_R T + \lambda'\pi_C T}{T} \\ &= (\pi_G+\pi_T)\lambda + (1-\pi_R)\lambda' + \pi_R\mu'. \end{aligned} \tag{7}$$

Given the steady-state probabilities,

$$X = \frac{\lambda\lambda_4\lambda_5(\lambda_1+\lambda_2+\lambda_3+\lambda_5)}{\phi} + \frac{\lambda'\lambda_4[(\lambda_1+\lambda_2+\lambda_3+\lambda_6)\lambda_5+\lambda_2\lambda_3]}{\phi} + \frac{\mu'[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi}. \tag{8}$$

In the next section, we describe how to determine the optimal rekeying rate that maximizes the performance and security tradeoff.

## 6. Experiments and numerical results

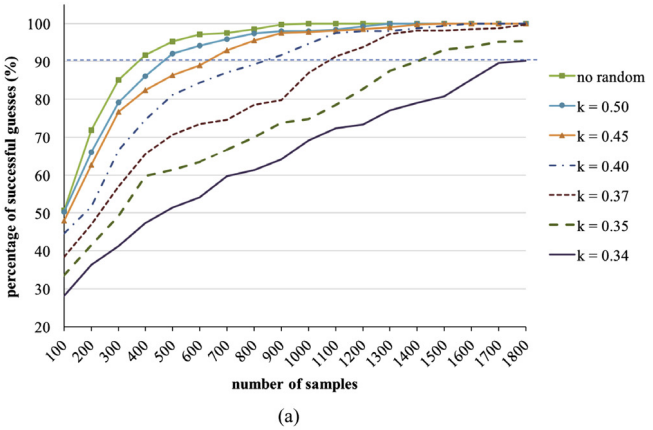
In this section, we empirically evaluate our approach for mitigation to timing attacks and for improving the security and performance tradeoff of mobile cloud offloading systems. Experimental and numerical analyses have been done to investigate:
(i) the mitigation effectiveness of random delay countermeasure against timing attacks;
(ii) 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
(iii) the optimal system rekeying rate for the performance and security tradeoff.

### 6.1. 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. 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.

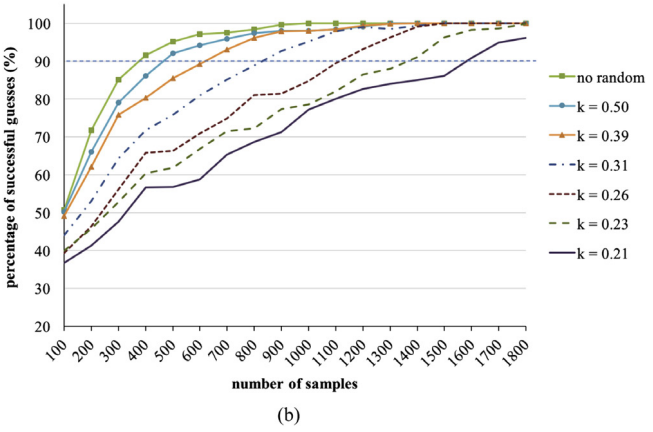
A timing attack uses statistical analysis of the time one application takes to do some calculation in order to learn about the data it is operating on. Timing attacks on secrets are fundamentally limited by the ability of an attacker to accurately measure differences in response times across a network. 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.

The experiments are conducted as follows: As an attacking client sends a guessed value (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. 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. 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.




(a)

| shape $k$ | scale $\eta$ | 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 $\eta$ | 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 |

**Fig. 4.** 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.

### *6.2. Mitigation effectiveness of random delay countermeasure*

In this section, we investigate the timing attack resilience of mobile cloud offloading systems with random delay paddings.

We evaluate the mitigation effectiveness of random delay countermeasure against timing attacks 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 random distributed delays to the limits of timing attacks has been compared in previous work [56]. 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 the subtables of Fig. 4. 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 Fig. 4(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 no random delays are added.

It is assumed that the attacker uses an error detection and correction strategy as described in [57], so 90% successful guesses is adequate for his attack. We record the numbers of samples on the 90 percentile of success 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 success attacks. 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 comparing with no random delay padding is added.

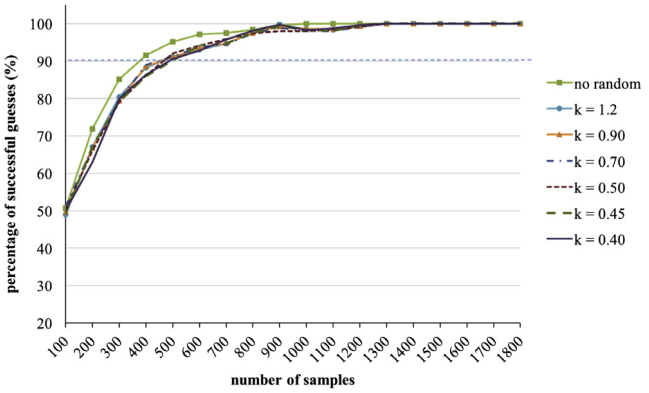



**Fig. 5.** The effectiveness of Weibull distributed random delays with the same variance but different means.

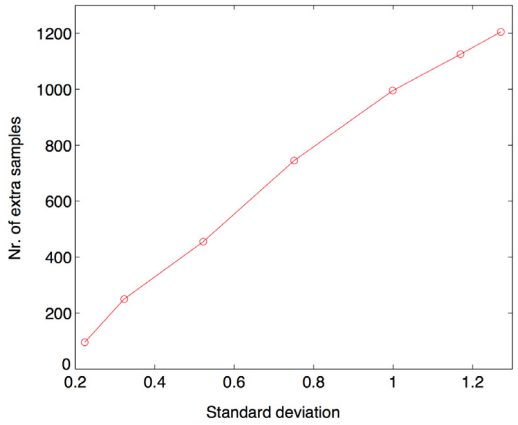



**Fig. 6.** The number of extra samples as a function of the standard deviation of Weibull distributed random delays.

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 0.1 ms and the variance is the same as in the first experiment (Fig. 4b). Surprisingly, the results are nearly the same as in the first experiment (Fig. 4a), 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 the mean does not affect the mitigation effectiveness observably. However the variance of random delays is the primary influencing factor to the mitigation effectiveness.

To support our standpoint, we conduct an experiment changing the mean of the Weibull random delays while keeping the variance constant. The result is presented in Fig. 5. 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 interposed. Different random padding policies with the same variance have the same effect on mitigating timing attacks even though the mean is growing.

### *6.3. Quantitative relationship between random delay variance and mitigation effectiveness*

In this subsection, we quantitatively analyze the relationship between the variance of Weibull random variable and the mitigation effectiveness of random delay countermeasure in Fig. 6. We use the number of extra samples the attacker needs to present the mitigation effectiveness against 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 [47].

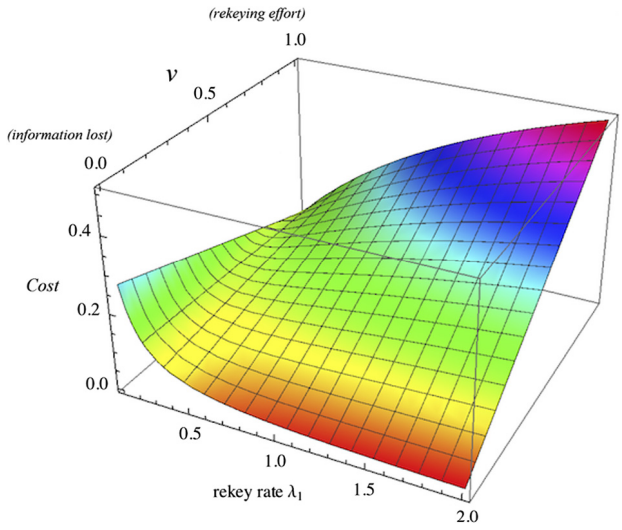



**Fig. 7.** Cost metric $\mathcal{C}$ over rekeying rate $\lambda_1$ and weighting parameter $v$.

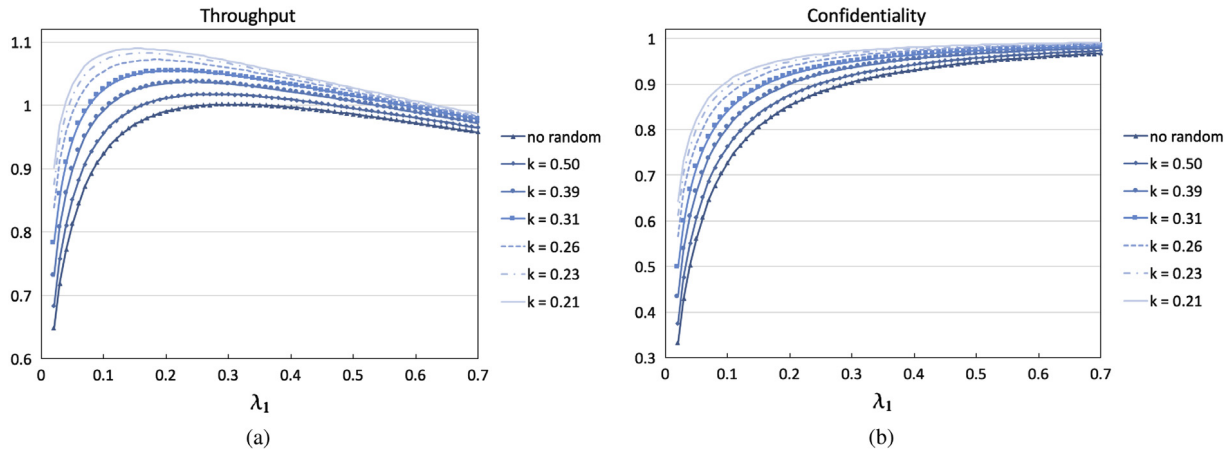



**Fig. 8.** Throughput metric $X$ (a) and confidentiality metric $\Delta$ (b) changing with rekeying rate $\lambda_1$.

### 6.4. Tradeoff analysis

In this section, we evaluate the metrics proposed in Section 4 using the model analysis methods. The rekeying rate $\lambda_1$ 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 [54]. The timing attack success rate $\lambda_3$ is taken from the timing attack experiment in Section 6.2. For the queueing model parameters we use experimental data from the offloading engine and OCR (Optical Character Recognition) implementation [9]. 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 $\mu = 1/1.191 \approx 0.84$. The arrival rates are $\lambda = 0.6$ and $\lambda' = 0.3$.

First, the effect of the weighting parameter $v$ on the system cost is analyzed in Fig. 7. We look at the marginal values first. When $v = 0$, we only consider the costs of losing sensitive information in the compromised system. It can be seen from the figure that the cost decreases monotonically with the rekeying rate $\lambda_1$. Intuitively, in this case when we trigger the rekeying process more often, the security cost will decrease because the system is more likely to be in the good state $G$. When all weight is assigned to the rekeying effort cost ($v = 1$), the *Cost* metric increases with the rekeying rate due to the increasing effort to trigger the rekeying process. The light color in the middle of the figure shows the optimum rekeying rate.

Fig. 8 shows the system performance and security metrics, i.e. throughput and system confidentiality, changing with the rekeying rate $\lambda_1$. The rekeying rate $\lambda_1$ indicates how often the system launches the rekeying process. It can be seen that

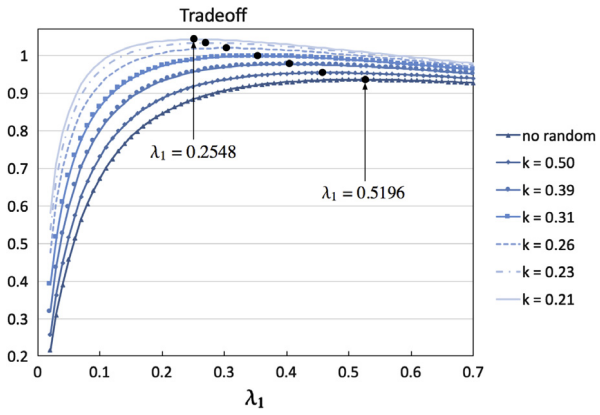



**Fig. 9.** Security and performance tradeoff metric $\mathcal{T}$ changing with rekeying rate $\lambda_1$.

**Table 1**
Optimum rekeying rate for different parameter-sets of Weibull distribution.

| Shape $k$ | Scale $\lambda$ | 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 |

the confidentiality measure monotonically increases with growing rekeying rate $\lambda_1$. This is because the security improves when the system launches the rekeying process more frequently. However, the throughput does not grow monotonically with the rekeying rate. It has an optimal point. After obtaining the maximum, the throughput metric decreases with growing rekeying rate $\lambda_1$. This is because the more often the server triggers the rekeying act, the more often the server denies clients' offloading requests. The system administrator can choose the offloading strategy with the highest throughput from Fig. 8(a). 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 Fig. 9. We can find the optimum rekeying rate for the best security and performance tradeoff at $\lambda_1 = 0.5169$, when 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 Fig. 9, 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 1, 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 Fig. 9.

In summary, when random delays are deployed in mobile cloud offloading systems, one should try to enlarge the variance of the random delay while keep the mean as low as possible by tuning the parameters. Using the results in Figs. 7 and 9, the system administrator can obtain the optimum rekeying interval for the minimum cost or maximum security performance tradeoff.

## 7. Conclusion

In this paper, 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. We have proposed a hybrid CTMC and queueing model for a mobile cloud 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. Based on the experimental results, the mitigation effectiveness of random padding countermeasure on the mobile cloud offloading systems is analyzed. The tradeoff analysis shows that through our strategy the system security performance tradeoff of can be enhanced comparing with no random delay padding is added. We found

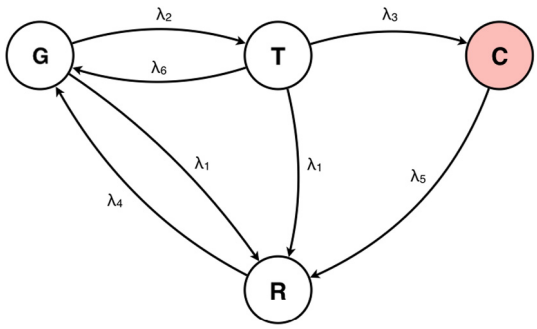



**Fig. A.10.** Security model of a mobile offloading system.

that the variance of random delays is the primary influencing factor to the mitigation effectiveness of the random padding countermeasure. So when random delays are deployed in mobile cloud offloading systems, one should tune the parameters to enlarge the variance while keep the mean as low as possible because a large mean would prolong the response time.

## Appendix. CTMC stationary distribution

The steady-state probabilities $\{\pi_i, i \in X_s\}$ of the CTMC can be computed by solving the system of linear equations [58]

$$\boldsymbol{\pi}\mathbf{Q} = 0, \tag{A.1}$$

where $\boldsymbol{\pi} = [\pi_R, \pi_G, \pi_T, \pi_C]$. $\mathbf{Q}$ is the infinitesimal generator (or transition-rate matrix) which describes the dynamic behavior of the security model. It can be obtained from Fig. A.10 as:

$$\mathbf{Q} = \begin{array}{c} \\ R \\ G \\ T \\ C \end{array} \begin{array}{c} \begin{array}{cccc} R & G & T & C \end{array} \\ \begin{pmatrix} -\lambda_4 & \lambda_4 & 0 & 0 \\ \lambda_1 & -\lambda_1-\lambda_2 & \lambda_2 & 0 \\ \lambda_1 & \lambda_6 & -\lambda_1-\lambda_3-\lambda_6 & \lambda_3 \\ \lambda_5 & 0 & 0 & -\lambda_5 \end{pmatrix} \end{array} \tag{A.2}$$

In addition, we have the total probability relationship:

$$\sum_i \pi_i = 1 \quad i \in X_s. \tag{A.3}$$

Then we can get the steady-state probability vector $\pi$ of the CTMC states by solving Eqs. (A.1) and (A.3). We obtain:

$$\begin{aligned} \pi_R &= \frac{[(\lambda_1+\lambda_2)(\lambda_1+\lambda_3)+\lambda_1\lambda_6]\lambda_5}{\phi}, \\ \pi_G &= \frac{(\lambda_1+\lambda_3+\lambda_6)\lambda_4\lambda_5}{\phi},\ \pi_T = \frac{\lambda_2\lambda_4\lambda_5}{\phi},\ \pi_C = \frac{\lambda_2\lambda_3\lambda_4}{\phi}, \end{aligned} \tag{A.4}$$

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

## References

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, ACM, 2010, pp. 49–62.
[2] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. 18 (1) (2013) 129–140.
[3] H. Wu, W. Knottenbelt, K. Wolter, Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics, in: Teletraffic Congress, ITC 27, 2015 27th International, IEEE, 2015, pp. 134–142.
[4] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2015) 974–983.
[5] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: Can offloading computation save energy? Computer (4) (2010) 51–56.
[6] H. Wu, Q. Wang, K. Wolter, Tradeoff between performance improvement and energy saving in mobile cloud offloading systems, in: Communications Workshops, ICC, 2013 IEEE International Conference on, IEEE, 2013, pp. 728–732.
[7] H. Qian, 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, 2015, pp. 1–8. http://dx.doi.org/10.1109/SNPD.2015.7176219.
[8] S. Deng, L. Huang, J. Taheri, A. Zomaya, Computation offloading for service workflow in mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2015) 3317–3329. http://dx.doi.org/10.1109/TPDS.2014.2381640.
[9] Q. Wang, 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, ACM, New York, NY, USA, 2015, pp. 3–13.

[10] Q. Wang, K. Wolter, Automated adaptive restart for accelerating task completion in cloud offloading systems, in: Autonomic Computing, ICAC, 2015 IEEE International Conference on, IEEE, 2015, pp. 157–158.
[11] H. Qian, D. Andresen, Automate scientific workflow execution between local cluster and cloud, Int. J. Netw. Distrib. Comput. (2016).
[12] A.N. Khan, M.M. Kiah, S.U. Khan, S.A. Madani, Towards secure mobile cloud computing: A survey, Future Gener. Comput. Syst. 29 (5) (2013) 1278–1299.
[13] I. Khalil, A. Khreishah, M. Azeem, Consolidated identity management system for secure mobile cloud computing, Comput. Netw. 65 (2014) 99–110.
[14] Z. Hao, Y. Tang, Y. Zhang, E. Novak, N. Carter, Q. Li, SMOC: A secure mobile cloud computing platform, in: Computer Communications (INFOCOM), 2015 IEEE Conference on, IEEE, 2015, pp. 2668–2676.
[15] H. Cui, X. Yuan, C. Wang, Harnessing encrypted data in cloud for secure and efficient image sharing from mobile devices, in: Computer Communications, INFOCOM, 2015 IEEE Conference on, IEEE, 2015, pp. 2659–2667.
[16] T.T. Mapoka, S.J. Shepherd, R.A. Abd-Alhameed, A new multiple service key management scheme for secure wireless mobile multicast, IEEE Trans. Mob. Comput. 14 (8) (2015) 1545–1559. http://dx.doi.org/10.1109/TMC.2014.2362760.
[17] M. Cagalj, T. Perkovic, M. Bugaric, Timing attacks on cognitive authentication schemes, IEEE Trans. Inf. Forensics Secur. 10 (3) (2015) 584–596. http://dx.doi.org/10.1109/TIFS.2014.2376177.
[18] T. Ristenpart, E. Tromer, H. Shacham, S. Savage, Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, ACM, 2009, pp. 199–212.
[19] B.B. Brumley, N. Tuveri, Remote timing attacks are still practical, in: Computer Security–ESORICS 2011, Springer, 2011, pp. 355–371.
[20] B.A. Braun, S. Jana, D. Boneh, Robust and efficient elimination of cache and timing side channels, 2015, arXiv preprint arXiv:1506.00189.
[21] D. Brumley, D. Boneh, Remote timing attacks are practical, Comput. Netw. 48 (5) (2005) 701–716.
[22] D.M. Nicol, W.H. Sanders, K.S. Trivedi, Model-based evaluation: from dependability to security, IEEE Trans. Dependable Secure Comput. 1 (1) (2004) 48–65.
[23] W.H. Sanders, Quantitative evaluation of security metrics, in: Quantitative Evaluation of Systems, QEST, 2010 Seventh International Conference on the, 2010, pp. 306–306. http://dx.doi.org/10.1109/QEST.2010.50.
[24] J.-f. Zhang, F. Liu, L.-m. Zheng, Y. Jia, P. Zou, Using network security index system to evaluate network security, in: E. Qi, J. Shen, R. Dou (Eds.), The 19th International Conference on Industrial Engineering and Engineering Management, Springer Berlin Heidelberg, 2013, pp. 989–1000.
[25] T. Meng, Q. Wang, K. Wolter, Model-based quantitative security analysis of mobile offloading systems under timing attacks, in: Analytical and Stochastic Modelling Techniques and Applications, Springer, 2015, pp. 143–157.
[26] E.K. Wang, T.-Y. Wu, C.-M. Chen, Y. Ye, Z. Zhang, F. Zou, Mdpas: markov decision process based adaptive security for sensors in internet of things, in: Genetic and Evolutionary Computing, Springer, 2015, pp. 389–397.
[27] D.L. Nazareth, J. Choi, A system dynamics model for information security management, Inf. Manag. 52 (1) (2015) 123–134.
[28] K. Wolter, P. Reinecke, Performance and security tradeoff, in: Formal Methods for Quantitative Aspects of Programming Languages, Springer, 2010, pp. 135–167.
[29] S. Belaïd, V. Grosso, F.-X. Standaert, Masking and leakage-resilient primitives: One, the other (s) or both? Cryptogr. Commun. 7 (1) (2014) 163–184.
[30] A. Iacovazzi, A. Baiocchi, Internet traffic privacy enhancement with masking: Optimization and tradeoffs, IEEE Trans. Parallel Distrib. Syst. 25 (2) (2014) 353–362. http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.42.
[31] K. Sinha, 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, IEEE Computer Society, 2011, pp. 184–194.
[32] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wirel. Commun. Mob. Comput. 13 (18) (2013) 1587–1611.
[33] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, J.-L. Willems, A practical implementation of the timing attack, in: Smart Card Research and Applications, Springer, 1998, pp. 167–182.
[34] C. Rebeiro, D. Mukhopadhyay, S. Bhattacharya, An introduction to timing attacks, in: Timing Channels in Cryptography, Springer, 2015, pp. 1–11.
[35] O.S. Foundation, OpenSSL - Cryptography and SSL/TLS Toolkit, 2016, https://www.openssl.org/.
[36] I.A. Sumra, J.-L. Ab Manan, 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, 2011, pp. 151–155.
[37] T. Meng, X. Li, S. Zhang, Y. Zhao, A hybrid secure scheme for wireless sensor networks against timing attacks using continuous-time Markov chain and queueing model, Sensors 16 (10) (2016) 1606.
[38] S.A. Crosby, D.S. Wallach, R.H. Riedi, Opportunities and limits of remote timing attacks, ACM Trans. Inf. Syst. Secur. 12 (3) (2009) 17.
[39] A. Askarov, D. Zhang, A.C. Myers, Predictive black-box mitigation of timing channels, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, ACM, 2010, pp. 297–307.
[40] A. Haeberlen, B.C. Pierce, A. Narayan, Differential privacy under fire, in: USENIX Security Symposium, 2011.
[41] D. Zhang, A. Askarov, A.C. Myers, Predictive mitigation of timing channels in interactive systems, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, ACM, 2011, pp. 563–574.
[42] D. Cock, Q. Ge, T. Murray, 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, ACM, 2014, pp. 570–581.
[43] B. Köpf, D. Basin, An information-theoretic model for adaptive side-channel attacks, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, ACM, 2007, pp. 286–296.
[44] G. Doychev, B. Köpf, Rational protection against timing attacks, in: 2015 IEEE 28th Computer Security Foundations Symposium, IEEE, 2015, pp. 526–536.
[45] J.-S. Coron, I. Kizhvatov, An efficient method for random delay generation in embedded software, in: Cryptographic Hardware and Embedded Systems-CHES 2009, Springer, 2009, pp. 156–170.
[46] P.C. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in: Advances in Cryptology—CRYPTO96, Springer, 1996, pp. 104–113.
[47] C. Clavier, J.-S. Coron, N. Dabbous, Differential power analysis in the presence of hardware countermeasures, in: Cryptographic Hardware and Embedded Systems—CHES 2000, Springer, 2000, pp. 252–263.
[48] Y. Lu, M.P. O'Neill, J.V. McCanny, FPGA implementation and analysis of random delay insertion countermeasure against DPA, in: ICECE Technology, 2008. FPT 2008. International Conference on, IEEE, 2008, pp. 201–208.
[49] S. Kotipalli, Y.-B. Kim, M. Choi, Asynchronous advanced encryption standard hardware with random noise injection for improved side-channel attack resistance, J. Electr. Comput. Eng. 2014 (2014) 19.
[50] Z. He, X. Deng, B. Yang, K. Dai, X. Zou, A SCA-resistant processor architecture based on random delay insertion, in: Computing and Communications Technologies, ICCCT, 2015 International Conference on, IEEE, 2015, pp. 278–281.
[51] A. Shamir, Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks, Google Patents, 1999 US Patent 5,991,415.
[52] B.A. Köpf, et al., Formal Approaches to Countering Side-Channel Attacks (Ph.D. thesis), ETH, 2007.
[53] B. Köpf, M. Dürmuth, A provably secure and efficient countermeasure against timing attacks, in: Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE, IEEE, 2009, pp. 324–335.

[54] T. Meng, K. Wolter, Q. Wang, Security and performance tradeoff analysis of mobile offloading systems under timing attacks, in: Computer Performance Engineering, Springer, 2015, pp. 32–46.
[55] K.S. Trivedi, Probability & Statistics with Reliability, Queuing and Computer Science Applications, John Wiley & Sons, 2008.
[56] T. Meng, K. Wolter, Analysis of mitigation measures for timing attacks in mobile-cloud offloading systems, in: Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems, Springer, 2016, pp. 168–182.
[57] C. Chen, T. Wang, J. Tian, Improving timing attack on rsa-crt via error detection and correction strategy, Inform. Sci. 232 (2013) 464–474.
[58] W.J. Stewart, Probability, Markov Chains, Queues, and Simulation: the Mathematical Basis of Performance Modeling, Princeton University Press, 2009.