EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

The performance analysis of public key cryptography-based authentication

Kaiqi Xiong[1,*]

[1]University of South Florida, Tampa, Florida 33620, USA

Abstract

Several Kerberos-based authentication techniques using public-key cryptography have been proposed. Public-key cryptography can be used to eliminate a single point failure problem in the Key Distribution Center (KDC) and achieve better scalability. Public Key Cryptography for Cross-Realm Authentication in Kerberos (PKCROSS) and Public Key Utilizing Tickets for Application Servers (PKTAPP, a.k.a. KX.509/KCA) are considered two notable techniques. The latter was suggested to improve the former, but their actual computational and communication times have been poorly understood. This paper first presents a thorough performance evaluation of the two protocols based on analytical analysis and queueing network models. As shown, PKTAPP does not scale better than PKCROSS. Then, this paper gives a new public key cryptography-based group authentication technique. We show that the new technique can achieve better scalability than PKCORSS and PKTAPP and our performance methodology is effective.

Received on 12 January 2018; accepted on 16 January 2018; published on 15 May 2018

Keywords: Security, Performance Evaluation, Complexity Analysis

doi:10.4108/eai.15-5-2018.154770

1. Introduction

There are an explosive growth in a variety of collaborative group services in academia and industry for the past ten years [18]. Example applications include video and audio conferencing, replicated servers and databases, and in particular Web services in that services components from several universe service providers can be flexibly integrated into a composite service regardless of their location, platform, and execution speed [5]. To ensure quality of services, these service providers are required to work together. The rapid growth in collaborative applications has heightened the need for a reliable group communication system. Without group authentication, the communication system will not be possible to be reliable.

A mature, reliable, secure network authentication protocol, Kerberos [29], allows a client to prove its identity to a server without sending confidential data across the network. Partitioning of the world into realms served by different application servers is a way to improve the scalability of Kerberos. In each realm, Kerberos consists of a client, application servers and a key distribution center (KDC). The client may represent a group of business users who request services from application servers. The KDC maintains a shared symmetric key with every client and the application servers in the realm. In case KDC is compromised (i.e., a single failure), all the symmetric keys will be divulged to the attacker and will have to be revoked. Recovering from such a compromise requires the re-establishment of new shared keys with the client and the application servers in the realm. In terms of time, effort and financial resources, it is very costly for such a recovery.

Kerberos has been extended based on public key cryptography since public key cryptography simplifies the distribution of keys in Kerberos. It eliminates a single point of failure. Integrating public key cryptography into Kerberos represents the enhancements of the current Kerberos standard. Several Kerberos-based authentication techniques using public-key cryptography have been proposed in the last decade. Among them include Public-Key Cross Realm Authentication in Kerberos (PKCROSS) [25] and Public-key Cryptography for Initial Authentication in Kerberos (PKINIT)

*Corresponding author. Email: xiongk@usf.edu

[41]. Moreover, the scalability of network security infrastructures is becoming a serious concern as the explosive growth of collaborative applications such as Web services continues unabated. Public key based Kerberos for Distribution Authentication (PKDA) [37] and Public Key Utilizing Tickets for Application Servers (PKTAPP, a.k.a. KX.509/KCA) [30] and [31] have been proposed to enhance the security and scalability of Kerberos.

Among these Internet drafts, PKINIT is a core specification. Both PKCROSS and PKTAPP use variations of PKINIT message types and data structures for integrating public key cryptography with Kerberos in different authentication stages. PKTAPP was originally introduced as PKDA. It implemented PKDA using the message formats and exchanges of PKINIT. Microsoft has adopted the Internet draft specification of PKINIT for the support of public key cryptography in the Windows 2000 and 2003 implementations of Kerberos [17]. It has its own protocol that is the equivalent of KX509/KCA. There were preliminary discussions regarding an adoption of a common protocol between the Kerberos WG and Microsoft. The MIT Kerberos consortium will drive these discussions [4]. According to Altman [3], the standardization of PKCROSS and PKTAPP will be the next for Kerberos and PKI integration. The Kerberos Consortium at MIT was formed in September 2007 led by initial board members from Apple, Google, MIT, Microsoft and Sun and sponsored by 19 leading companies, universities and government agencies [7]. It is expected that Kerberos-based authentication and authorization will be as ubiquitous as TCP/IP-based networking itself. PKINIT and PKCROSS are listed as Projects 10 and 11 in the Kerberos Consortium's proposal respectively [32]. As stated above, PKCROSS and PKTAPP use variations of PKINIT message types and data structures in the design of the authentication protocols. We believe that PKCROSS and PKTAPP will revive soon. Hence, this paper only considers these two notable techniques: PKCROSS and PKTAPP. The understanding of the variants of the two protocols can be achieved through the performance analysis of these two protocols.

Sirbu and Chuang [37] argued that PKCROSS would not scale well in a large network. PKTAPP was proposed to improve the scalability of PKCROSS. However, the actual costs associated with these techniques have been *poorly understood* so far. Although PKTAPP is shown to poorly perform when there are two or more application servers in one remote realm in Harbitter and Menasce [24], it remains *unknown* which technique performs better in a large network where application servers are within *multiple* KDC remote realms that are typical in many applications.

In the design of protocols, performance evaluation is a fundamental consideration. Two conventional approaches have been proposed to analyze the performance of a security protocol. The first one is to implement under study protocols and then take their measurements in real systems (for example, see Amir et al. [1]). This approach seems very good but it is time-consuming. Whenever the under-study protocol is changed, the updated protocol has to be re-implemented and the measurement data of the protocol implementation has been re-collected. This approach cannot be applied to our study of PKCROSS and PKTAPP. The implementation of original PKCROSS and PKTAPP is not available due to lack of resource and funding [4]. (Note: the implementation of PKTAPP's variant KX.509) was just released last year [30].) The second one is to count the number of operations and then compute their corresponding costs within a under study protocol (for example, see Amir et al. [2] and Steiner et al. [38]). This approach is straightforward. It has widely used by researchers but it is not very easy to be used when the protocol becomes complicated such as the case of multiple KDC remote realms. Furthermore, in this case, an authentication request cannot often be processed fast enough, so it should wait in a queue. However, the second approach does not consider the waiting time of an authentication request in a queue. Hence, in order to efficiently and effectively analyze the performance of a protocol, we use a queueing network model as our protocol evaluation tool.

In this paper, we first presents a thorough performance evaluation of PKCROSS and PKTAPP in terms of computational and communication costs. Although the complexity of message exchanges in multiple KDC remote realms, we figure out the number of secret, private and public key operations required in PKCROSS and PKTAPP. Then, we demonstrate their performance difference using open queueing networks in which we gain a better understanding of these two techniques compared to the approach of only counting the number of operations in the two techniques. An in-depth analysis of these two techniques shows that PKTAPP does not scale better than PKCROSS. This is a contrary result as expected by PKTAPP protocol designers. Thus, this paper further gives a new public key cryptography-based group authentication technique. While giving the design and security of the new authentication technique, we mainly focus on the study of its performance compared to PKCROSS and PKTAPP in this paper. Our performance analysis indicates that the new technique achieves better scalability as compared to PKCORSS and PKTAPP. It should be pointed out that our proposed performance methodology is an effective way to analyze these authentication techniques which can be thus extended to analyze other security protocols as well. The preliminary results of this research appeared in [40]. The proposed performance methodology can be

applied to understand other security protocols (see [26] and [36]).

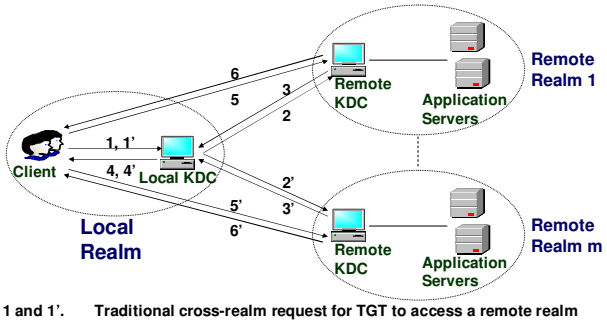
The rest of this paper is organized as follows. In Section 2, we begin by giving an in-depth discussion of PKCROSS and PKTAPP and then presents their performance evaluation using queueing theory. Section 3 gives a definition of the new public key cryptography-based group authentication technique along with the details of messages. We present performance analysis of the proposed technique compared to PKCROSS and PKTAPP in Section 4. The related work is discussed in Section 6. We finally conclude our discussion in Section 7.

2. PKCROSS and PKTAPP

Kerberos [29] is a network authentication protocol for providing a secure communication between a user workstation (or called a client) and application servers that was developed at the Massachusetts Institute of Technology in 1988. The latest version of Kerberos is Version 5. It divides the world into realms, each with user workstations, a single primary KDC, back-up KDCs, and application servers in which the KDC is a trusted intermediary.

In the Kerberos protocol, the client engages in a multiple-step authentication to obtain access to the application server whereby the client first obtains a relatively short lived credential: a Ticket-Granting Ticket (TGT) from the Authentication Service running on a (KDC), and then obtain a session ticket for a particular application server by presenting the TGT to a centralized Ticket-Granting Service (TGS) running on the KDC. The client presents the session ticket to the application server for authenticating herself/himself by showing knowledge of a secret session key. The secret session key was securely passed to the client by the KDC. Kerberos is stateless. It is extremely valuable from the scalability point of view. Cross-realm authentication is necessary when a client and an application server with different network domains fall into different Kerberos realms.

It is well-known that a public key security system is easier to administer, more secure, less trustful, and more scalable than a symmetric key security system. Public key security doesn't use a trusted key management infrastructure since the burden of key management falls to public key clients [12]. In a public key infrastructure, public key clients need constantly and vigorously check the validity of the public keys that they use. Public key cryptography shifts the burden on key management from the KDC/TGS in Kerberos to its Certificate Authority (CA) who may be considered as a trusted intermediary. CA issues a public key certificate that is relatively long lived credential. The burden is determined by the number of times that clients want



1 and 1'. Traditional cross-realm request for TGT to access a remote realm
2 and 2'. AS_REQ with PKCROSS flag set
3 and 3'. AS_REP with PKCROSS ticket encrypted using local KDC's public key
4 and 4'. Traditional reply with cross-realm TGT using SKC
5 and 5'. Ticket-Granting Service Request (TGS-REQ)
6 and 6'. Ticket-Granting Service Reply (TGS-REP)

Figure 1. The PKCROSS Message Flow

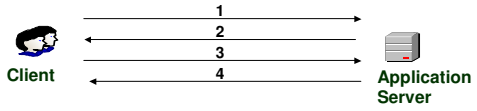
to authenticate to application servers in Kerberos. It might not be affordable in time-sensitive applications if one large-scale PKI deployment is needed for group authentication in a large network.

2.1. Protocol Analysis

PKCROSS [25] is one notable protocol of integrating public key cryptography with Kerberos to address the problem of network authentication among the client and the application servers in a large number of realms. Figure 1 illustrates the authentication steps of PKCROSS. The cross realm KDC to KDC authentication is achieved by using public key cryptography.

First, just like Kerberos the KDC in PKCROSS is burdened by the need to constantly renew short-lived TGTs, and TGS must be involved whenever a client wants to request a service from an application server. Thus, if a large number of users in a single realm request services that may be a typical case in Web services, the KDC in the realm becomes a serious single point of failure due to a KDC compromise and possibly a performance bottleneck. Recovering such a compromise requires the re-establishment of secret keys for all users within this realm. When a number of users is large, such a recovery is very costly. PKINIT facilitates public-key based authentication between users and their local KDC. Hence, one possible solution to eliminating the single point of failure is to combine PKCROSS with PKINIT, and thus public-key based authentication is integrated with the *entire* Kerberos environment. However, it is easy to see that this solution is not feasible for time-sensitive applications, since the users might suffer from a long delay or even denial of services due to a high computational cost for the calculation of a public key used in the entire environment.

Second, in PKCROSS the local KDC of the client issues all short-lived TGTs and all session tickets in its



1. A standard Kerberos version 5 message that includes PA-PK-AS-REQ pre-authentication field
2. A standard Kerberos version 5 message that includes PA-PK-AS-REP pre-authentication field
3. AS_REQ in which the client requests a service from the application server, using the traditional secret-key procedure defined in RFC1510
4. AS_REQ in which the client receives a reply from the application server, using the traditional secret-key procedure defined in RFC1510

Figure 2. The PKTAPP Message Flow

Table 1. The Operations of Encryption and Decryption When n Application Servers are in m Remote Realms

Protocols	Entities	# Secret Key	# Private Key	# Public Key
PKTAPP	Client	$2n+1$	$n+1$	$3n$
	Application server	$3n+1$	n	$4n$
	Total	$5n+2$	$2n+1$	$7n$
PKCROSS	Client	$m+6$	0	0
	Local KDC	5	$m+1$	$3m$
	Remote KDC	$3m+n$	m	$4m$
	Application server	$3n$	0	0
	Total	$4(n+m)+11$	$2m+1$	$7m$

realm, and communicates with a remote KDC. Hence, it can easily become a performance bottleneck since all these authentication transactions have to transit the KDC. Thus, PKTAPP [31] has been proposed to address the issue. Figure 2 shows the message exchange of PKTAPP. The PKTAPP technique allows the client to communicate directly with the application servers so that the number of messages between them is reduced in an authentication process. But, as is seen in [24], even though PKTAPP requires fewer message exchanges than PKCROSS during client authentication with remote application servers, PKCROSS outperforms PKTAPP when two or more application servers are in a single remote realm. This is because PKTAPP requires more public-key message exchanges than PKCROSS that only requires one pair of public-key message exchanges. Below, we are going to study their performance in multiple remote realms.

We employ the first approach to analyzing the performance of the two techniques in which we are required to carefully compute the number of secret, private and public operations used in PKCROSS and PKTAPP in the case of a multiple remote realm. Table 1 summarizes the encryption and decryption operations performed in these two techniques where m and n are the number of remote realms and the number of applications servers within these realms respectively. Denote by c_j the computational times per secret, private, or public key operation respectively where $j = 1, 2, 3$. Then, $c_1 < c_2 < c_3$. Let $f_j(n, m)$, $j = 1, 2$, be the total computational times of encryption and decryption operations in PKTAPP and PKCROSS. Thus, from Table 1 we have that

$$\begin{aligned} f_1(n,m) &= (5c_1 + 2c_2 + 7c_3)n + 2c_1 + c_2 \\ f_2(n,m) &= 4c_1 n + (4c_1 + 3c_2 + 7c_3)m + 11c_1 + c_2 \end{aligned}$$

Note that $f_1(n, m)$ does not depend on m, which can be hence abbreviated as $f_1(n)$. Then, we have the following proposition.

Proposition 2.1. For each authentication, PKCROSS requires less computational time than PKTAPP if and only if the number of application servers n is more than $\lceil m + \frac{3(m+3)c_1}{c_1+2c_2+7c_3} \rceil$, or the number of remote realms m is less than $\lfloor n - \frac{3(n+3)c_1}{4c_1+2c_2+7c_3} \rfloor$.

Proof. According to $f_1(n, m)$ and $f_2(n, m)$, their difference is given by $f_1(n, m) - f_2(n, m) = -9c_1 + (c_1 + 2c_2 + 7c_3)n - (4c_1 + 2c_2 + 7c_3)m$. Hence, $f_1(n, m) - f_2(n, m) \geq 0$ if and only if $-9c_1 + (c_1 + 2c_2 + 7c_3)n - (4c_1 + 2c_2 + 7c_3)m \geq 0$, which implies Proposition 2.1. □

It is anticipated that the client is connected to the local KDC by a LAN, the client and the local KDC are connected to a remote KDC by a WAN, and a remote KDC and its application servers are connected by a WAN. Assume that all WANs have an identical communication time and a local area network (LAN) has a neglected communication time compared to a WAN. From Figures 1 and 2, we note that the number of WAN communications are $4n$ for PKTAPP and $4m + 2n$ for PKCROSS. Let $g_j(n, m)$, $j = 1, 2$, be the transaction times of PKTAPP and PKCROSS. *The transaction time* is defined as the computational time of the total encryption and decryption operations plus the communication time per authentication in a technique. Also, denote by d the time spent in a WAN communication. Then, the following conclusion holds.

Proposition 2.2. For each authentication, PKCROSS uses less transaction time than PKTAPP if and only if the number of application servers n is more than $m + \lceil \frac{(3c_1+2d)m+9c_1}{c_1+2c_2+7c_3+2d} \rceil$.

Proof. For $j = 1, 2$, $g_j(n, m)$ can be computed by $g_1(n, m) = f_1(n) + 4n$ and $g_2(n, m) = f_2(n, m) + 4m + 2n$ and thus their difference is given by $g_1(n, m) - g_2(n, m) = (c_1 + 2c_2 + 7c_3 + 2d)n - (4c_1 + 2c_2 + 7c_3 + 4d)m - 9c_1$ which easily derives Proposition 2.2. □

Note that if $n = 1$ (so, $m = 1$), $m + \lceil \frac{(3c_1+2d)m+9c_1}{c_1+2c_2+7c_3+2d} \rceil > 1 = n$, and if $m = 1$, $m + \lceil \frac{(3c_1+2d)m+9c_1}{c_1+2c_2+7c_3+2d} \rceil = 1 + \frac{12c_1+2d}{c_1+2c_2+7c_3+2d} < 2$ since c_1 is significantly smaller than c_3. This means that

Corollary 1. When $m = 1$, we have that PKTAPP requires less transaction time than PKCROSS if $n = 1$ but more transaction time than PKCROSS if $n \geq 2$.

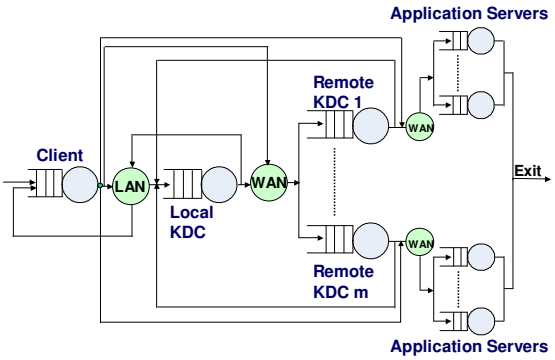


Figure 3. A Queueing Network with m Remote Realms for PKCROSS

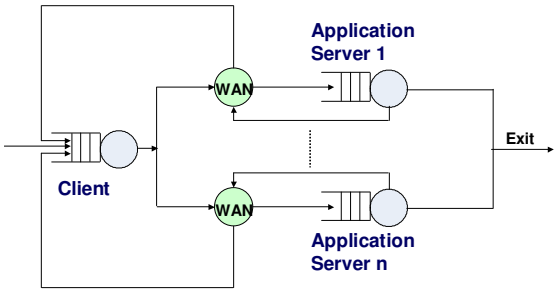


Figure 4. A Queueing Network with n Application Servers for PKTAPP

In Proposition 2.2 we have shown that the number of application servers should be $\lceil \frac{(3c_1+2d)m+9c_1}{c_1+2c_2+7c_3+2d} \rceil$ more than the number of remote KDC realms so as to ensure that PKCROSS uses less transaction time than PKTAPP. But, the transaction time does not take into account the time required to wait in a queue for a service (i.e., waiting time) when multiple authentication requests present in any one of the authentication entities. Response time is used when such a case is considered. That is, *the response time* is the transaction time plus the waiting time per authentication request. A further discussion of the response time is given in next section.

2.2. The Calculation of Response Time via Queueing Networks

In a distributed system, a queueing network is an efficient tool to analyze system scalability. To investigate the scalability of PKTAPP and PKCROSS, we first model the entities, the client, the local KDC, the remote KDCs, the application servers and communication networks, as a queueing network. The client may represent a single or multiple users that request group authentication at a given rate and the authentication request is processed in these entities according to these two techniques. Figures 3 and 4 give queueing networks to describe the message flows of PKCROSS and PKTAPP where each system resource is modeled as a queue associated with a queueing discipline.

Since a public key consumes a significantly higher computation cost than a private key, it is not reasonable to assume that all client requests are served at the same average service time. Instead, a class-switching in [6] and [33] is employed to model the class transaction with switching from low- to high-priority class, as different types of encryption/decryption operations are required with different service times. Preemption-resume is one good way to implement a service for satisfying multiple class client requests. In this paper, we use a *preemptive-resume* priority discipline, i.e., the service of a class r_2 request can be interrupted if a higher-priority request of class r_1 $(r_2 > r_1)$ arrives during its service. The interrupted request resumes its service from where it stopped after the higher-priority request, and any other request with priority higher than r_2 that may arrive during its service, complete their service.

Our goal is to use the queueing networks to calculate the response time of an authentication request. The calculation is sketched below. Assume that the client requests group authentication at a rate λ. Based on the forced law, the throughput of an entity (the client station, the local KDC, the remote KDCs, or the application servers) is $X^{(j)} = \lambda v^{(j)}$ for class j job where $v^{(j)}$ is the number of visits to the entity by class j jobs. Then, the total throughput X is a sum of $X^{(j)}$ over all job classes at the entity. Thus, according to the utilization law, the utilization of the entity by class j jobs is $\rho^{(j)} = X^{(j)}\mu^{(j)} = \lambda v^{(j)}\mu^{(j)}$ where $\mu^{(j)}$ is the service time per visit to the entity by class j jobs. Hence, the total utilization ρ is a sum of $\rho^{(j)}$ over all classes at the entity. Thereby, the response time of the entity is $R = \frac{\mu}{1-\rho}$ where μ is an average service time of all classes at the entity, and the total response time per authentication request is a sum of vR over all entities.

To validate the accuracy of the queueing networks, we first simulated the scenario of these entities as shown in Figure 5 by doing the reference implementations of these two techniques under Windows XP. Moreover, a public-key cipher is usually associated with the calculations of 1024-bit precision numbers, so a public-key operation is computationally expensive and it costs as much as a factor of 1000 than an equivalent secret-key operations [15]. In the reference implementations we adopted the results from the Crypto++ ran on an Intel Core 2 1.83 GHz processor under Windows XP SP 2 in 32-bit mode [10]. Table 2 gives the time required to encrypt and decrypt a 64-byte block of data. Without loss of generality, we chose $c_1 = 0.000248$ msec, $c_2 = 0.07$ msec and $c_3 = 1.52$ msec. (Performance

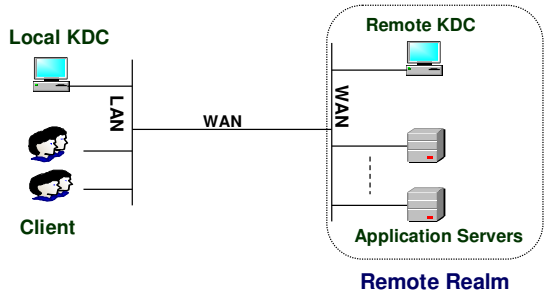


Figure 5. A Test Scenario With a Remote Realm

Table 2. The computational times of Encryption and Decryption Operations

Protocols and Operation	Key Length	Computational times (msec)
AES/ECB	128	0.000248
AES/ECB	256	0.000312
RSA encryption	1024	0.07
RSA decryption	1024	1.52
RSA encryption	2048	0.15
RSA decryption	2048	5.95

Table 3. The Comparison of Analytic and Simulated Response Times

m=1		The Number of Application Servers				
Protocols	Methods	1	2	4	8	16
PKCROSS	Analytic	102.1	122.1	162.1	242.1	402.1
	Simulated	102.3	122.5	162.9	244.2	408.8
	R-Err%	-0.22	-0.30	-0.47	-0.85	-1.63
PKTAPP	Analytic	82.10	164.05	327.96	655.76	1311.38
	Simulated	82.23	164.56	329.97	663.87	1344.23
	R-Err%	-0.15	-0.30	-0.61	-1.22	-2.44

evaluation based on an ECC key will be discussed in another paper.) Table 3 shows the accuracy of analytical response times obtained by the queueing methods compared to simulated results based on the scenario shown in Figure 5, where R-Err% is the relative error used to measure the accuracy of the analytic results compared to model simulation results, and it is defined by (analytic result - simulated result)/simulated result × 100. As seen in the table, the analytic response times match the simulated results very well.

To investigate performance with an increased number of application servers and remote realms, we further presented response times as a function of authentication request rates, i.e., throughput, when there are two remote realms, as shown in Figure 6. As is seen, PKCROSS has slightly less response time than PKTAPP when $n = 4$, called *a crossover number*. That is, PKCROSS is more efficient than PKTAPP if $n \geq 4$, but less efficient than PKTAPP if $n < 4$. Clearly, it follows from Table 3 that the crossover number is equal to 2 when $m = 1$. In general, Figure 7 shows crossover

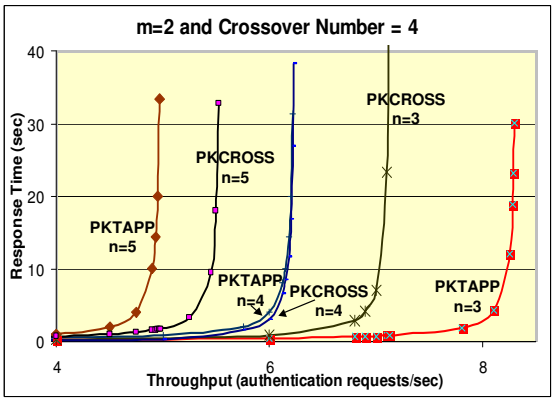


Figure 6. Response Time vs. Authentication Request Rate

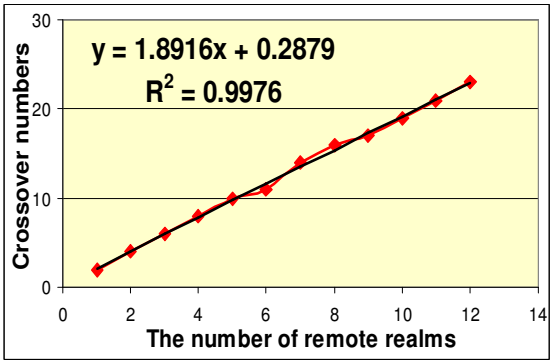


Figure 7. Crossover Numbers vs. The Number of Remote Realms

numbers with varying number of remote realms. The crossover numbers can be 99.8% perfectly fitted by the straight line: $n = 1.8916m + 0.2879$. This means that PKCROSS is more efficient than PKTAPP when $n \geq \lceil 1.8916m + 0.2879 \rceil$. In other word, PKTAPP does not scale better than PKCROSS. That is different from what PKTAPP's designers expected. In next section we propose a hybrid approach that has better scalability than PKCROSS.

3. A New Group Authentication Technique Using Public-Key Cryptography

As is seen above, in PKCROSS the client is first required to communicate with the KDC before talking the application server. PKTAPP uses a direct communication between the client and the application servers for the reduction of message exchanges and the relieve of a single point of failure on the client's KDC. While PKTAPP is more efficient in the case of only a single application server and thus only a single remote realm too, PKCROSS significantly performs better if the number of application servers is more than a crossover number as shown in Table 3 and Figure 7. Our proposed technique below will consider the advantages of both

Table 4. The Notation Used In The Case of A Single Remote Realm

C	Client
S_j	Application Server j ($j = 1, 2, \cdots, n$)
KDC_L	Local KDC, i.e., Client's KDC
KDC_R	Remote KDC, i.e., Application servers' KDC
K_C	Secret key of C
K_{S_j}	Secret key of S_j
$K_{C,S}$	Group key shared by C and all S_j
$\{\text{Message}\}_{KDC_L}$	Message encrypted with KDC_L's public key
$\{\text{Message}\}_{KDC_R}$	Message encrypted with KDC_R's public key
$[\text{Message}]_{KDC_R}$	Message signed with KDC_R's private key
KB_{KDC_R}	Public key of KDC_R
$Cert_{KDC_L}$	Certificate of KDC_L
$Cert_{KDC_R}$	Certificate of KDC_R
N_C	Nonce generated by C
N_{KDC_R}	Nonce generated by KDC_R
TMS_C	Timestamp generated by C
$TMS1_R, TMS2_R$	Timestamps generated by KDC_R

techniques. While it allows the client to deal directly with the application servers so that the number of messages is reduced in the authentication process like PKTAPP, the new technique still relies on the KDC for the authentication of the client and the application servers like PKCROSS.

3.1. A Single Remote Realm

In this section we define the group authentication technique among a client (denoted by C) and application servers (denoted by S_j) that are in a single remote realm (denoted by KDC_R), where $j = 1, \cdots, n$. The client C is a group key initiator, for example, representing either a group member in a video conference or a service provider in Web services applications who wants to initiate secure communication among groups or service providers. (How C is chosen is beyond the scope of our study in this paper.) The application servers S_j are the rest of either group members or service providers. Table 4 lists all notation used in this section. Figure 8 shows the message flow of the new group authentication technique whose message exchanges is given in detail as follows. (In order to emphasize our understanding of this new technique for its performance comparison with PKCROSS and PKTAPP, we briefly state the main goal of each exchange message.)

Message 1, $C \rightarrow KDC_R$**:**
N_C, "C", "KDC_R: S_1, ..., S_n",$K_C\{N_C, K_{C,S}$, "$KDC_R : S_1, \cdots, S_n$", $TMS_C\}$

Main Goal: In this message, the client C initiates a group key request for a secure communication with the application servers S_j.

For this purpose, it will encrypt the request message with a key, $K_{C,S}$, that the client C invents. To make sure that only S_j can get the key (of course, KDC_L and KDC_R should be able to get it as well), the key will

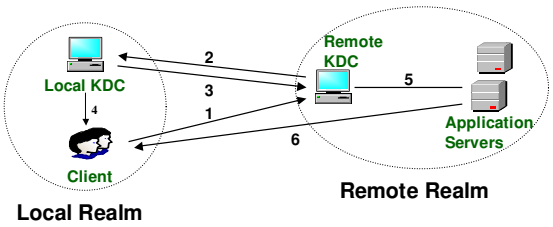


Figure 8. The Message Flow of A New Technique in a Single Remote Realm

be encrypted using the client C's key along with the destination, "KDC_R: S_1, ..., S_n". For S_j to get $K_{C,S}$, the KDC_R will first need to be authenticated by the KDC of the client C, and then S_j will need to be authenticated by their KDC: KDC_R and have KDC_R give S_j the key, $K_{C,S}$. KDC_R will be responsible for making sure that S_j are valid recipients and the client C is a valid sender validated by the client C's KDC, KDC_L. A common nonce, N_C, will also be invented to track the transaction and give the recipient something to authenticate with to the remote KDC server, KDC_R. A timestamp TMS_C is included to prevent replay attacks or prevent the key, $K_{C,S}$, from being given out more than once.

Message 2, $KDC_R \rightarrow KDC_L$**:**
N_C, "C", $K_C\{N_C, K_{C,S}$, "$KDC_R : S_1, ..., S_n$", $TMS_C\}$, $\{N_C, N_{KDC_R}, TMS1_R$, AuthInfo$\}_{KDC_L}$, where AuthInfo consists of "KDC_R", $Cert_{KDC_R}$, $[N_C, N_{KDC_R}, TMS1_R$, "KDC_L", $KB_{KDC_R}]_{KDC_R}$.

Main Goal: The remote KDC, KDC_R, will authenticate the client C through KDC_L who is the local KDC of the client.

After receiving the message from the client C, KDC_R invents a nonce, N_{KDC_R}, and generates AuthInfo necessary to authenticate KDC_R. Then, it constructs a message to send to the client C's local KDC that contains N_{KDC_R}, N_C, $TMS1_R$, and AuthInfo, encrypted with KDC_L's public key. KDC_R will also forward the part of the original message encrypted with C's key. This is enough information for KDC_L to authenticate the client C and KDC_R to make sure only if KDC_L can decrypt this message. Note that the message uses "KDC_R" instead of "KDC_R: S_1, ..., S_n" for the sake of a privacy consideration. This is because the client C's local KDC is unnecessary to know who will be part of the secure communication except the client C. Additionally, this message does not explicitly contain the identity KDC_R since it is included in AuthInfo. "KDC_R" and its public key KB_{KDC_R} are uniquely determined by $Cert_{KDC_R}$ in AuthInfo. Thus, $Cert_{KDC_R}$ is used to prevent man-in-the-middle attacks. $TMS1_R$ serves to avert replay attacks.

Message 3, $KDC_L \rightarrow KDC_R$**:**
$\{$"C", "KDC_R", $N_{KDC_R}, K_{C,S}, Cert_{KDC_L}, TMS_C$, $TMS1_R\}_{KDC_R}$

Main Goal: After the client KDC_L authenticates the client C, it sends a reply to KDC_R.

Using the key of the name given, KDC_L will decrypt the message encrypted with its public key and the original message encrypted by K_C, and verify KDC_R's signature. Then, it will check if the N_C encrypted by key K_C matches the N_C encrypted by its public key. If they match, then the client C is really the client C and its message is not altered. KDC_L will then read the destination, in this case, "KDC_R." (Again, it does not include "KDC_R: $S_1, \cdots, S_n$ in the message for the sake of a privacy consideration.) Furthermore, KDC_L will make sure if KDC_R is a valid member. If so, it continues the processing. Accordingly, KDC_L will make sure that the timestamp is within the allowed clock-skew and that the key for this request has not already been given out. KDC_L will then give out the key, $K_{C,S}$. KDC_L will also encrypt N_{KDC_R} with KDC_R's public key to authenticate KDC_L to KDC_R. KDC_R will verify that the N_{KDC_R} received from KDC_L matches the N_{KDC_R} sent to KDC_L, and it will distribute the returned $K_{C,S}$ to S_j.

Message 4, $KDC_L \rightarrow C$**:**

$K_C\{K_{C,S}, "KDC_L", "FromKDC_R", N_C, TMS_C, TMS_L\}$

Main Goal: The KDC_L further informs the client C about its authentication and the share key $K_{C,S}$ with the application servers.

The client decrypts the share key $K_{C,S}$, "$FromKDC_R$", N_C, TMS_C, and TMS_L. The client needs to make sure that the N_C matches the nonce of a pending request, the timestamp is within the allowed clock-skew, and $K_{C,S}$ matches the group key that was previously created. The resources required for a brute force attack scale exponentially with increasing key size, not linearly. As a result, doubling the key size for an algorithm does not simply double the required number of operations, but rather squares them. Thus, using timestamps and nonce numbers makes the encrypted message random to some extent. Thus, this prevents a brute-force cryptographic attack and a plain-text attack on the secret key of the client.

Message 5, $KDC_R \rightarrow S_j$**:**

$K_{S_j}\{"C", K_{C,S}\}, K_{C,S}\{N_C, TMS_C, TMS2_R\}$

Main Goal: KDC_R informs its application servers S_j about their shared key $K_{C,S}$ with the client C.

The application servers S_j will decrypt the first message to get $K_{C,S}$, and then use the shared key $K_{C,S}$ to decrypt the second message to get TMS_C and $TMS2_R$. TMS_C is included so that S_j knows when the group key was inverted by C. To make sure that it is not a replay, S_j should keep all timestamps that were recently received, say in the last five minutes that is a parameter set approximately for the maximum allowable time skew. Then, S_j should check that each received timestamp from a given request initiator is different from any of the stored values. Any authentication request older than five minutes (or whenever the value of the maximum allowable time skew) would be rejected anyway, so S_j would not remember values older than 5 minutes.

Messages 6, $S_j \rightarrow C$**:**

$K_{C,S}\{N_C, TMS_C\}$, where $j = 1, 2, \cdots, n$.

Main Goal: In the end, the application servers S_j communicate with C for making sure that $K_{C,S}$ is not altered.

The application servers send replies to the client. Then, by using pre-generated $K_{C,S}$, the client decrypts the message to make sure that the group key $K_{C,S}$ is not altered. It also verifies nonce N_C and makes sure the received timestamp of the pending request is within the allowed clock-skew of the timestamp TMS_C.

In the new technique, note that $K_{C,S}$ can be replaced by K_{C,S_j} when C only wants to securely communicate with any one of application servers S_j. The aforementioned technique can be also modified so that a reply is issued to C by either KDC_L or KDC_R whenever needed after C is authenticated.

3.2. Multiple Remote Realms

In a large network, application servers are often within different network domains. For instance, in Web services service providers may be partitioned into many different realms due to their location flexibility. To work together, a service provider may wish to gain access to other service providers' application servers in remote realms. To support "cross-realm" authentication, the service provider's KDC needs to establish an either direct or indirect trust relationship with the other service providers' KDCs. Here, we briefly describe how the proposed technique is extended in multiple remote realms.

Assume that the n application servers S_j are distributed in m realms, each with KDC servers denoted by KDC_i $(i = 1, \cdots, m)$. Let n_i be the number of application servers within KDC_i, where $0 \le n_i \le m$ and $\sum_{i=1}^{m} n_i = n$. Clearly, the case of a single remote realm is associated with $n_1 = n$ and $n_i = 0\,(i = 2, \cdots, m)$. We further let $S_{i,k}$ be those application servers which belong to the set $\{S_j \,|j = 1, \cdots, n\}$ within realm i, where $k = 1, \cdots, n_i$. Then, the set $\cup_{i=1}^{m}\{S_{i,k}\,|k = 1, \cdots, n_i\}$ is identical to the set $\{S_j \,|j = 1, 2, \cdots, n\}$. Table 5 lists additional notation used in this section.

Figure 9 shows how the client authenticates to application servers. Authentication messages are given below.

Messages 1 and 1', $C \rightarrow KDC_i$ for each fixed i:

N_C, "C", "KDC_i: $S_{i,1}$, ..., S_{i,n_i}", $K_C\{N_C, K_{C,S}$, "$KDC_i : S_{i,1}, \cdots, S_{i,n_i}$", $TMS_C\}$

Messages 2 and 2', $KDC_i \rightarrow KDC_L$**:**

N_C, "C", $K_C\{N_C, K_{C,S}$, "$KDC_i : S_{i,1}, \cdots, S_{i,n_i}$", $TMS_C\}$, $\{N_C, N_{KDC_i}, TMS1_i$, AuthInfo$_i\}_{KDC_L}$, where AuthInfo$_i$ consists of "KDC_i",

Table 5. The Additional Notation Needed In Multiple Remote Realms

$S_{i,k}$	Application Server k in realm i
KDC_i	Remote KDC server in realm i
$K_{S_{i,k}}$	Secret key of $S_{i,k}$
KB_{KDC_i}	Public key of KDC_i
$\{\text{Message}\}_{KDC_i}$	Message encrypted with KDC_i's public key
$[\text{Message}]_{KDC_i}$	Message signed with KDC_i's private key
N_{KDC_i}	Nonce generated by KDC_i
$Cert_{KDC_i}$	Certificate of KDC_i
$TMS1_i$ and $TMS2_i$	Timestamps generated by KDC_i

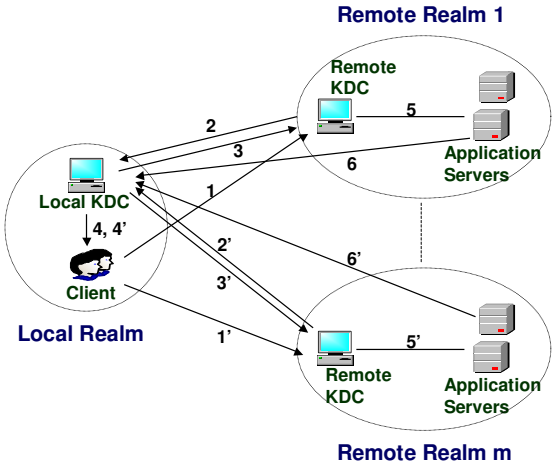


Figure 9. The Message Flow of The New Technique in Multiple Remote Realms

$Cert_{KDC_i}$, and $[N_C, N_{KDC_i}, TMS1_i, \text{“}KDC_L\text{”}, KB_{KDC_i}]_{KDC_i}$

Messages 3 and 3', $KDC_L \rightarrow KDC_i$ where $i = 1, \cdots, n$: $\{\text{“}C\text{”}, \text{“}KDC_i\text{”}, N_{KDC_i}, K_{C,S}, Cert_{KDC_L}, TMS_C, \}_{KDC_i}$

Messages 4 and 4', $KDC_L \rightarrow C$**:** $K_C\{K_{C,S}, \text{“}KDC_L\text{”}, \text{“}FromKDC_i\text{”}, N_C, TMS_C, TMS_L\}$

Messages 5 and 5', $KDC_i \rightarrow S_{i,k}$**:** $K_{S_{i,k}}\{\text{“}C\text{”}, K_{C,S}\}$, $K_{C,S}\{N_C, TMS_C, TMS2_i\}$, where $i = 1, \cdots, m$ and $k = 1, \cdots, n_i$.

Messages 6 and 6', $S_{i,k} \rightarrow C$**:** $K_{C,S}\{N_C, TMS_C\}$, where $i = 1, \cdots, m$ and $k = 1, \cdots, n_i$.

In the first three messages, the client C authenticates to her local KDC through remote KDC_i. Message 4 or 4' checks if the group key $K_{C,S}$ is valid, and Message 5 or 5' distributes the group key $K_{C,S}$ and authenticates designated application servers within their individual KDC realms. An explanation of these messages are similar to the one in Section 3.1.

4. The Performance Evaluation of The New Proposed Technique

In this paper we focus on studying the efficiency of the proposed group authentication technique. Hence, we first compute its computational and communication costs. Then, we give a thorough performance evaluation of the new technique using the queueing method proposed in Section 2. The security discussion of this new authentication will be given in Section 5.

Table 6. The Operations of Encryption and Decryption When n Application Servers are in m Remote Realms

Entities	# Secret Key	# Private Key	# Public Key
Client	3	0	0
Local KDC	2	1	$4m$
Remote KDC	n+1	$2m$	$3m$
Application server	$3n$	0	0
Total	$4n$+6	$2m$+1	$7m$

4.1. The Operations of Encryption and Decryption

The baseline transactions are constructed with one or more application servers in a remote realm as shown in Figure 8. Due to page limit, we directly consider the case of m remote realms, each with n_i application servers where $0 < n_i \leq n$ and $i = 1, 2, \cdots, m$. Table 6 summarizes the number of encryption and decryption operations performed in the proposed technique. As is shown in Tables 1 and 6, in our technique the computational burden on the client is mitigated to its local KDC compared to PKTAPP. Specifically, in term of the calculation of public key operations the burden on the client's local KDC is $O(m)$ in both the proposed technique and PKCROSS but the burden on the client is $O(n)$ in PKTAPP. Let us recall that m is the number of remote KDC servers and n is a total number of application servers where $1 \leq m \leq n$. Hence, their computational burdens may be significantly different when $m \ll n$. Note that the proposed technique uses public key cryptography to authenticate the client's KDC and the remote KDCs of application servers. So, it reduces the risk of a single failure on these KDCs.

Next, let us consider the operation costs of the proposed technique. Denote by $f_3(n, m)$ the total computational time of its encryption and decryption operations. Then, it follows from Table 6 that $f_3(n, m)$ is computed by

$$f_3(n, m) = 4c_1 n + (2c_2 + 7c_3)m + 6c_1 + c_2$$

Thus, we have the following proposition.

Proposition 4.1. (a) The proposed technique requires less computational time than PKCROSS. (b) For $n \geq 4$, the proposed technique requires less computational time than PKTAPP. But, when $1 \leq n < 4$, the proposed technique requires less computational time PKTAPP if and only if the number of remote realms m is less than $\lfloor n + \frac{(n-4)c_1}{2c_2+7c_3} \rfloor$.

Proof. The differences of computational times among the three techniques are given by $f_1(n, m) -$

$f_3(n,m) = -4c_1 + (c_1 + 2c_2 + 7c_3)n - (2c_2 + 7c_3)m$ and $f_2(n,m) - f_3(n,m) = 4c_1m + 5c_1$. Obviously, our technique requires less computational time than PKCROSS due to $f_2(n,m) - f_3(n,m) > 0$. Moreover, $f_1(n,m) - f_3(n,m) \geq 0$ if and only if $m \leq n + \frac{(n-4)c_1}{2c_2+7c_3}$, which implies (b) due to $n \geq m$. □

Similarly, we can easily get that

Proposition 4.2. For $m \geq 4$, the proposed technique requires less computational time than PKTAPP. But, when $1 \leq m < 4$, the proposed technique requires less computational time PKTAPP if and only if the number of application servers n should be more than $\lceil m - \frac{(m-4)c_1}{c1+2c_2+7c_3} \rceil$.

Furthermore, let $g_3(n,m)$ be the transaction time of the proposed technique, i.e., the computational time of its encryption and decryption operations plus its communication time. Note that the number of WAN communications required in the technique is $3m+2n$, and recall that d is the time spent in a WAN communication. Then, the following statements hold.

Proposition 4.3. (a) The proposed technique has less transaction time than PKCROSS. (b) The proposed technique uses less transaction time than PKTAPP if and only if the number of application servers n should be more than $\lceil m + \frac{(d-c_1)m+4c_1}{c1+2c_2+7c_3+2d} \rceil$.

Proof. For $j = 1, 2, 3$, $g_j(n,m)$ are given by $g_1(n,m) - g_3(n,m) = [f_1(n,m) - f_3(n,m)] + (2n - 3m)d = (c_1 + 2c_2 + 7c_3 + 2d)n - (2c_2 + 7c_3 + 3d)m - 4c_1$, and $g_2(n,m) - g_3(n,m) = [f_2(n,m) - f_3(n,m)]$. These easily derive (a)-(b) in this proposition. □

Note that when $n = 1$, we get that $m + \frac{(d-c_1)m+4c_1}{c1+2c_2+7c_3+2d} > 1 = n$ due to usually $d > c_1$, which imply $g_1(n,m) - g_3(n,m) < 0$. Also, when $m = 1$ and $n \geq 2$, $m + \frac{(d-c_1)m+4c_1}{c1+2c_2+7c_3+2d} = 1 + \frac{d+3c_1}{c1+2c_2+7c_3+2d} < 2 \leq n$ due to $c_1 < c_3$, which imply $g_1(n,m) - g_3(n,m) > 0$. Thus, we have that

Corollary 2. PKTAPP is more efficient than the proposed technique when $n = 1$, but less efficient when $m = 1$ and $n \geq 2$, in term of transaction time.

By using Proposition 4.3 we calculated the minimal number of application servers so as to ensure that our technique requires less transaction time than PKTAPP with varied $d = 0.12$ msec, 4.8 msec and 10 msec when $c_1 = 0.000248$ msec, $c_2 = 0.07$ msec and $c_3 = 1.52$ msec in Table 7. We observed that the minimal number of application servers is sensitive to d rather than c_j ($j = 1, 2, 3$). Table 8 further presents the difference of transaction times between our technique and PKCROSS. We also noted that our proposed technique requires significantly less transaction time than PKCROSS, and the difference of transaction times between the two techniques are independent of the number of application servers. That is, the proposed technique is more efficient than PKCORSS.

Table 7. The Minimal Number of Application Servers

	The Number of Remote Realms									
# Servers	1	2	3	4	5	6	7	8	9	10
d=0.12	2	3	4	5	6	7	8	9	10	11
d=4.8	2	3	4	5	7	8	9	10	12	13
d=10	2	3	4	6	7	8	10	11	12	14

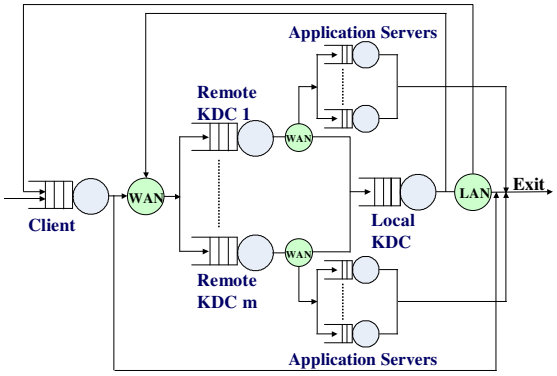


Figure 10. A Queueing Network with n Remote Realms for the New Technique

4.2. The Calculation of Response Time via a Queueing Network

Similar to Section 2, we can use a queueing network to characterize the message flow of the proposed technique where each system resource is modeled as a queue associated with a queueing discipline shown in Figure 10. Figures 11, 12 and 13 show response times as a function of authentication request rates, i.e., throughput, in the case of one, two or eight remote realms with varying number of application servers. As is seen, our technique performs better than PKCROSS in all cases. It has also been demonstrated that our technique is more efficient than PKTAPP when $n \geq 2$ for one remote realm in Figure 11, when $n \geq 4$ for two remote realms in Figure 12, and when $n \geq 12$ for eight remote realms in Figure 13 (roughly $n \geq 1.5m$ if $m > 1$). These crossover numbers from $m = 1$ to 12 are further depicted in Figure 14 where the crossover numbers are 99.6% perfectly fitted by the straight line: $n = 1.4406m + 0.6364$, which predicts the number of application servers required to ensure that our technique performs better than PKTAPP. Moreover, the line $n = 1.4406m + 0.6364$ is below the straight line $n = 1.8916m + 0.2879$ given in Section 2 due to $1.8916m + 0.2879 > 1.4406m + 0.6364$ when $m \geq 1$. This again confirms that our technique is more efficient

Table 8. The Difference of Transaction Times Between Our Technique and PKCROSS (i.e., $g_2(n, m)$-$g_3(n, m)$)

	The Number of Remote Realms									
Difference (msec)	1	2	3	4	5	8	12	16	20	24
d=0.12	0.1222	0.2432	0.3642	0.4852	0.6062	0.9691	1.45314	1.93711	2.42108	2.90504
d=4.8	4.8022	9.6032	14.404	19.205	24.006	38.409	57.6131	76.8171	96.0210	115.225
d=10	10.002	20.003	30.004	40.005	50.006	80.009	120.013	160.017	200.021	240.025

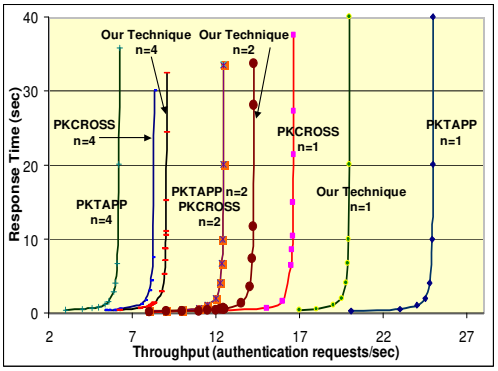


Figure 11. Response Times vs. Authentication Request Rates when $m = 1$

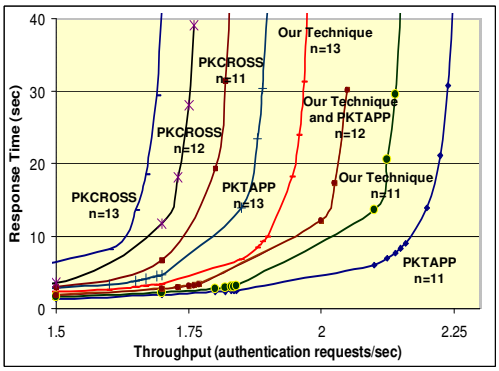


Figure 13. Response Times vs. Authentication Request Rates when $m = 8$

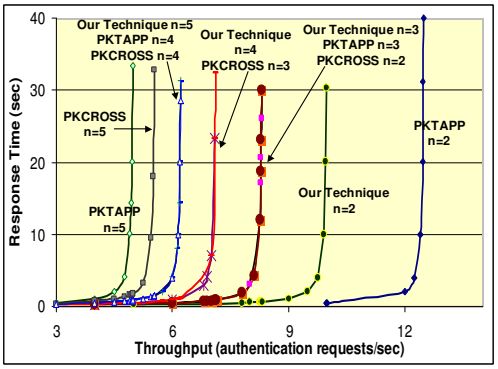


Figure 12. Response Times vs. Authentication Request Rates when $m = 2$

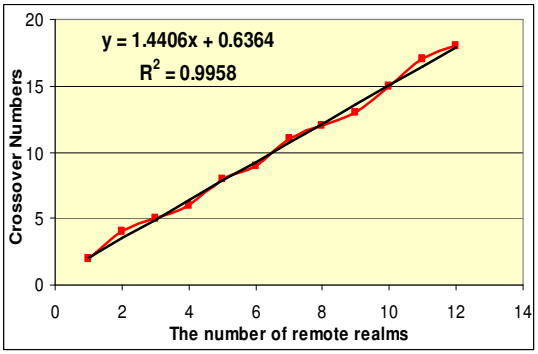


Figure 14. Crossover Numbers vs. Remote Realms

than PKCROSS. We also noted that the numbers n in Table 7 are smaller than the crossover numbers in Figure 14. This means that it is not sufficient *only if* transaction time is employed to analyze performance that researchers have often used.

5. Security Discussions

The proposed authentication technique allows for efficient message exchanges and the establishment of a group authentication among a client and multiple application servers. It is based on a security handshake. When presenting the message exchanges of the authentication technique in Sections 3.1 and 3.2, we have discussed its security. We further give the security discussion of this technique below.

A replay attack is one of common security pitfalls. The proposed technique is a server-based protocol providing authenticated transport (with key authentication and key freshness assurances) in six messages. If the clocks between the clients and the KDCs were not synchronized, and if old tickets and authenticators were not cached, a replay attack would become possible. However, the proposed technique employs both timestamps and nonces as well as these nonces are unpredictable in its authentication messages. Thus, if a message is received with a timestamp that is beyond the allowed clock skew, then the message will be discarded. Moreover, as long as both each KDC and each

application server in this proposed technique remember recent nonces within the allowed clock skew, then replay attacks are not possible. This means that in this technique, the clients and the KDCs runs must provide secure and reasonably synchronized clocks to prevent replay attacks. Subsequently, the proposed authentication technique will not suffer from replay attacks.

Note that Messages 2 and 2' do not leave clear except N_C and "C" in order that any clock synchronization problem from client can be avoided to misuse by an attacker. Moreover, the identity of the client and the integrity of Messages 2 and 2' can be authenticated through the information of "AuthInfo." The $Cert_{KDC_R}$ used in the field of "AuthInfo" is prevent man-in-middle attacks.

As is known, the traditional Kerberos transmits the identity of the client in plaintext. But, this is not a case in the proposed technique except the first two messages. This change enhances the client's privacy protection. It prevents an attacker from the tracking of the identities of the client's and the KDCs' who communicate each other for authentication. Additionally, an original version of Kerberos protocol [28] used the Needham-Schroder protocol as modified by Denning and Sacco (see [9] and [13]). However, there is a ticket invalidation issue in the Needham-Schroder protocol. A suggested fix was proposed in the extended Needham-Schroder protocol and in the Otway-Rees through a use of 7 and 5 messages respectively. The proposed authentication technique does not encounter a ticket invalidation issue.

Furthermore, the proposed technique is conceptually simple as it involves the client that generates a key and the KDCs that authenticate and distribute the key to group members via a pair-wise secure channel established with each group member. It works well in one-to-many multicast scenarios, for example, secure communications among service providers in Web services.

As is well-known, *denial of service (DoS) attacks* is one of the most common attacks these days [14]. DoS attacks cannot be prevented in PKCROSS and PKTAPP because authentication requests can be constantly sent to the KDC and the application server by an attacker [19]. While our proposed technique has several performance strengths as compared to PKCROSS and PKTAPP as shown in this paper, we have also realized that DoS attacks are out of our control. The client's request message might be passed through by the remote KDC to the client's local KDC with lack of any verification, which results in serious DoS attacks. To prevent them, we are required to securely migrate the client's request and maintain an authentication state at every remote KDC that might not be an easy task. This is why a DoS attack is one of the most difficult attacks that we can prevent, as happened in PKCROSS and PKTAPP. But, our technique is motivated by secure communications in Web services applications and it is mainly toward to e-business applications in which a trust-but-verify framework for Web services authorization has been proven an efficient method [39]. So, under the same framework we trust a client in a certain degree and allow the client to contact the KDC server of the application servers directly. Then, the authentication procedures proposed in this paper will follow. However, if the remote KDC server is badly under DOS attacks due to unauthenticated messages from the client, then we leave the option to switch back to the way in which a client needs to be first authenticated by its local KDC, e.g., using PKCROSS.

In the proposed technique, we also allow the client to create the group key $K_{C,S}$. This is particularly useful when the client does not need to get a reply from its local KDC for a further verification. If the reply is required, then we can easily mitigate the creation of $K_{C,S}$ from the client to its local KDC by slightly modifying the proposed technique accordingly.

6. Related Work

Kerberos (see RFC1510 [29]) has changed rapidly since 1999. Among them, there have been numerous proposals to integrate public key cryptography into Kerberos [8], [11], [16], [25], [19], [31], [34], [41], and [37]. These proposals address various concerns of Kerberos in distributed networks, for instance, security, scalability, and portability. Neuman, et al. [34] proposed PKINIT to enable use of public key cryptography for an initial authentication between the client and its local KDC. PKINIT is an extension of the Kerberos protocol. Its mechanism has been developed under the IETF Kerberos WG for eleven years [3] before PKINIT was approved as an IETF Proposed Standard (RFC4556, see Zhu and Tung [41]).

PKCROSS [25] has been proposed to simplify the administrative burden of maintaining cross-realm keys so that it improves the scalability of Kerberos in large multi-realm networks. Public key cryptography takes place only KDC-to-KDC authentication. PKINIT and PKCROSS are centralized KDC protocols. Sirbu and Chuang [37] extended these two protocols to create PKDA for improving scalability and addressing the single point of failure on the KDC. PKTAPP is only a slight variation on the PKDA specification. Both PKDA and PKTAPP use lengthy public-key message exchanges between the client and the application servers, so they may not be anymore efficient than public key enabled authentication once with a KDC and faster secret-key cryptography for subsequent encryption with application servers [24]. PKTAPP is also known as KX.509/KCA [16] and Windows has its own protocol which is the equivalent to KX.509/KCA

(see Altman [3]). Although the evolution of PKTAPP, its protocol structure has not been dramatically changed. We believe that PKCROSS and PKTAPP will revive soon. The Kerberos Consortium at MIT was just formed in September 2007 and listed PKCROSS as Project 10 in its proposal [32]. Kerberos on the web and Kerberos on mobile devices have been included in the strategic pillars of the MIT Kerberos Consortiym [21].

Performance evaluation is a fundamental consideration in the design of security protocols. However, performance associated with most of these protocols has been poorly understood. To analyze the performance of a protocol, we can first implement the protocol and then analyze measurement data taken from the implementation. But, the implementation of a protocol is very time-consuming and constricted to development resources and funding. While the implementation of KX.509 was released in 2007 [30], up to today, PKCROSS has not been implemented yet due to a matter of lack of development resources and funding [4]. A group of scientists have recently started to discuss the implementation of PKCROSS [22] and [23]. Hence, performance modeling has become an attractive approach since it can quickly provide a performance guidance used for a protocol design. The simplest modeling approach for a study of security protocols is to coount the number of secret, private, public key operations and then compute their corresponding costs (for example, see Amir et al. [2] and Steiner et al. [38]). This approach is straightforward. But, as shown in Sections 2 and 4, it is not easy to compute the number of operations for PKCROSS, PKTAPP and our authentication techniques since their message exchanges become complicated in the case of multiple KDC remote realms. Furthermore, this approach does not consider the waiting time of an authentication request in a queue. Hence, in order to efficiently and effectively analyze the performance of a protocol, we use a queueing network model as our protocol evaluation tool.

Queueing theory has been used in an analysis of chanllge/response authentication protocols in Liang and Wang [20]. But, they studied the performance of these protocols by only using a single $M/M/1$ queue which is the simplest case in queueing theory. Existing studies in Harbitter and Menasce [24] employed the performance modeling approach for examining the impact of PKCROSS and PKTAPP on network throughput based on their skeleton implementations and construction of *closed* queueing networks for a relatively simple case: there is only a single remote realm. This present paper also employs a performance modeling approach for a study of PKCROSS and PKTAPP but extends Harbitter and Menasce [24] in several essential and important aspects. *First*, while the performance of PKCROSS and PKTAPP was studied in Harbitter and Menasce [24], it remains unknown which technique performs better in multiple remote realms that are typical in an increasingly large network these days. It is very difficult to analyze the case of multiple remote realms due to the complexity of authentication message exchanges. The difficulty is in the complexity analysis of these protocols and the building and analysis of queueing network models that reflects the workload of authentication requests for these protocols in the case of *multiple* remote realms. *Second*, we explicit derive the formulas for calculating the computational and communication times of these protocols so as to easily determine which technique is better. *Third*, using a closed queueing network in Harbitter and Menasce [24] assumes there exist *constant* authentication requests in the queueing network. This means that the number of the client's authentication requests remains unchanged with time, which is obviously an unrealistic assumption in a real-world computer application such as Web services. Rather, we adopted an *open* queueing network where the client requests authentication at a given rate, i.e., the number of authentication requests in a computing system under study is *not constant*; it can be dynamically changed with time. *Fourth*, in order to better understand the performance of the authentication technique, we distinguish the processing ordering of multiple authentication requests by using the preemptive-resume priority discipline. *Fifth*, due to a performance trade off between these two protocols according to our scalability analysis, we propose a new hybrid technique. Our analysis has showed that the new technique has better scalability than these two protocols in most cases. Finally, we must point out that the approach of using queueing networks is relatively complicated in the case of multiple remote realms but it is necessary. As is seen, the numbers n given in Table 8 based on the approach of counting the number of operations are smaller than the crossover numbers presented in Figure 14. The preliminary results of this research was published in [40].

7. Conclusions and future work

Public-key enabled Kerberos-based techniques such as PKINIT, PKCROSS, PKDA and PKTAPP (a.k.a KX.509/KCA) give a potential effective way for cross-realm authentication. However, the authentication problem is simple to describe but hard to solve, especially for multiple realms. Their performance has been poorly understood. In this paper we presented a throughout performance evaluation of PKCROSS and PKTAPP in terms of computational and communication times, and through the use of validated analytical queueing models. Our analysis revealed that PKTAPP does not perform better than PKCROSS in a large network where there are many application servers

within either a single or multiple remote realms. Thus, we proposed a new public key cryptography-based group authentication technique. While giving the design and security of the new authentication technique in Section 5, we mainly focus on the study of its performance compared to PKCROSS and PKTAPP in this paper. Our performance analysis has showed that the proposed technique outperforms PKCROSS in terms of computational and communication times in Propositions 4.1 and 4.3, and response time in Figures 11, 12 and 13. This paper also gave the predicted minimal number of application servers so as to ensure that the proposed approach is more efficient than PKTAPP in multiple remote realms. Roughly speaking, the proposed technique performs better than PKTAPP when the number of application servers is about 50% more than the number of remote realms (i.e., $n \geq 1.5m$) in Section 4.2.

As is shown, our performance methodology based on complexity analysis and queueing theory is an effective way to analyze the performance of security protocols. In the future, we plan to apply the methodology to other protocols, such as IKEv2 in Kaufman [27] and AAA in Patel et al. [35]. Particularly, it is of very interest to apply our method to those protocols which are used in either time-sensitive or resource-limited applications, e.g., the ones in wireless sensor networks.

References

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. *ACM Transactions on Information and Systems Security (TISSEC)*, 7(3):1-32, August 2004.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton and G. Tsudik. Secure group communication using robust contributory key agreement. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):468–480, 2004.

[3] J. Altman. NIST PKI'06: Integrating PKI and Kerberos. `www.secure-endpoints.com/talks/nist-pki-06-kerberos.pdf`, 2007.

[4] J. Altman. Personal communication, 2007.

[5] D. Barry. *Web Services and Service-Oriented Architecture: Your Road Map to Emerging IT*. Morgan Kaufmann, 2003.

[6] S. Bruell and G. Balbo. *Computerational Algorithms for Closed Queueing Netowrks*. Editored by P. J. Denning, Science Library, Elsevier North Holland, Inc., New York, 1980.

[7] S. Buckley. MIT Kerberos Consortium Proposal to Sponsors. `http://www.kerberos.org/join/overview.pdf`, 2008.

[8] CITI. kx509 and KCA. `http://www.citi.umich.edu/projects/kerb_pki/`, 2006.

[9] J. Clark and J. Jacob A Survey of authentication protocol literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.

[10] W. Dai. Crypto++ 3.1 benchmarks. `http://www.eskimo.com/~weidai/benchmark.html`, 2007.

[11] D. Davis. Kerberos plus RSA for world wide web security. In *Proceedings of the First USENIX UNIX Workshop on Electronic Commerce*, New York City, New York, July 1995.

[12] D. Davis. Compliance defects in public-key cryptography. In *Proceedings of the Sixth USENIX UNIX Security Symposium (USENIX Security'96)*, San Jose, California, July 1996.

[13] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[14] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS 05)*, pages 38–49 , 2005.

[15] P. Dongara and T. N. Vijaykumar. Accelerating private-key cryptography via multithreading on symmetric multiprocessors. In *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS 03)*, pages 58–69, IEEE Press, 2003.

[16] W. Doster and M. Watts and D. Hyde. The KX.509 Protocol. `http://www.citi.umich.edu/techreports/reports/citi-tr-01-2.pdf`, 2001.

[17] J. Garman. *Kerberos: The Definitive Guide*. O'Reilly, 2003.

[18] 3GPP. Technical Specification Group Services and System Aspects; Study on the security aspects of the next generation system. In *3rd Generation Partnership Project (3GPP)*, TR 33.899, 08 2017. [Online]. Available: `http://www.3gpp.org/ftp/specs/archive/33series/33.899/`.

[19] Y. Kirsal and O. Gemikonakli. Further Improvements to the Kerberos Timed Authentication Protocol. In *Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics*, eidted by T. Sobh, K. Elleithy, A. Mahmood, and M. Karim, Springer, 2008.

[20] W. Liang and W. Wang. A Quantitative study of authentication and QoS in Wireless IP Networks. In *Proceedings of the 24th IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[21] T. Hardjono. Kerberos on the Web: Update. `http://www.kerberos.org/events/Board-3-30-09/3-hardjono-kerbweb.pdf`, MIT Kerberos Consortium, December 2005.

[22] Heimdal. PKCROSS for Heimdal. `http://www.taca.jp/krb-cross-realm/pkcross-heimdal.html`, April 2008.

[23] Heimdal. Initial version of PKCROSS Implementation. `http://www.stacken.kth.se/lists/heimdal-discuss/2008-04/msg00004.html`, Heimdal Discussion Mailing List, April 2008.

[24] A. Harbitter and D. Menasce. Perofrmance of public-key-enabled Kerberos authentication in large networks. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, Oakland, California, 2001.

[25] M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld. Public key cryptography for cross-realm authentication in Kerberos (PKCROSS). `http://tools.ietf.org/html/draft-ietf-cat-kerberos-pk-cross-07`, May 2001.

[26] R. Jover. Some key challenges in securing 5G wireless networks. *Electronic Comment Filing System*,

2017. [Online]. Available: https://www.fcc.gov/ecfs/filing/10130278051628

[27] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. http://www.ietf.org/rfc/rfc4306.txt, December 2005.

[28] C. Kaufman, R. Perlman, and M. Speciner. *Private Communication in a PUBLIC World, Second Edition*. Prentice Hall PTR, Basel-Boston-Berlin, 2002.

[29] J. Kohl and C. Neuman. RFC 1510: The Kerberos network authentication service (v5). http://rfc.net/rfc1510.html, 1993.

[30] KX.509. KX.509 Source. http://kx509.cvs.sourceforge.net/kx509/, 2007.

[31] A. Medvinsky, M. Hur, and C. Neuman. Public key utilizing tickets for application servers (PKTAPP). http://tools.ietf.org/html/draft-ietf-cat-pktapp-00, January 1997.

[32] The MIT Kerberos Consortium. Proposal for corporate sponsors. www.kerberos.org/join/proposal.pdf, 2007.

[33] R. Muntz, K. Chandy, F. Baskett, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, April 1975.

[34] B. Neuman, B. Tung, J. Way, and J. Trostle. Public key cryptography for initial authentication in Kerberos servers (PKINIT-02). http://ietf.org/internet-drafts/draft-ietf-cat-Kerberos-pk-init-02.txt, October 2002.

[35] A. Patel, K. Leung, M. Khalil, and H. Akhtar. Authentication protocol for mobile IPv6. http://www.rfc-editor.org/rfc/rfc4285.txt, 2006.

[36] D. Rupprecht, A. Dabrowski, T. Holz, E. Weippl, and C. Pppper. On security research towards future mobile network generations. https://arxiv.org/pdf/1710.08932.pdf, 2017.

[37] M. Sirbu and J. Chuang. Distributed authentication in Kerberos using public key cryptography. In *IEEE Symposium On Network and Distributed System Security (NDSS'97)*, 1997.

[38] M. Steiner, G. Tsudik and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security (CCS'96)*, 1996.

[39] C. Skalka and X. Wang. Trust but verify: authorization for web services. In *Conference on Computer and Communications Security, Proceedings of the 2004 workshop on Secure web service*, October 2004.

[40] K. Xiong. The Performance of Public Key-based Authentication Protocols. In *Proceedings of the 6th International Conference on Network and System Security*, 2012.

[41] L. Zhu and B. Tung. RFC 4556: Public key cryptography for initial authentication in Kerberos (PKINIT). http://www.ietf.org/rfc/rfc4556.txt, June 2006.