Performance of Public-Key-Enabled Kerberos Authentication in Large Networks

Alan H. Harbitter
PEC Solutions, Inc.
Aharbitter@pec.com

Daniel A. Menascé
George Mason University
Menasce@cs.gmu.edu

Abstract

Several proposals have been made to public-key-enable various stages of the secret-key-based Kerberos network authentication protocol. The computational requirements of public key cryptography are much higher than those of secret key cryptography, and the substitution of public key encryption algorithms for secret key algorithms impacts performance.

This paper uses closed, class-switching queuing models to demonstrate the quantitative performance differences between PKCROSS and PKTAPP—two proposals for public-key-enabling Kerberos. Our analysis shows that, while PKTAPP is more efficient for authenticating to a single server, PKCROSS outperforms the simpler protocol if there are two or more remote servers per remote realm. This heuristic can be used to guide a high-level protocol that combines both methods of authentication to improve performance.

1. Introduction

Kerberos has become a mature, reliable, secure network authentication protocol. Kerberos is based on secret key encryption technology. It is the native network authentication protocol in the Microsoft Windows 2000 operating system and may be a candidate for use as a general-purpose authentication protocol for large user communities on the Internet.

Several proposals have been developed that add public key cryptography to various stages of Kerberos to make the protocol work with large user communities and Public Key Infrastructures (PKIs). But the computational requirements of public key cryptography are higher than those of secret key cryptography. As a result, the substitution of public key encryption algorithms for secret key algorithms impacts performance.

This paper examines the performance impacts of PKCROSS and PKTAPP—two proposals to substitute public key cryptography for the client-to-server authentication and KDC-to-KDC (Key Distribution Center) inter-realm authentication in Kerberos. The context for this analysis is a large network with many application hosts participating in the authentication process.

Section 2 provides background information about Kerberos and proposals to public-key-enable Kerberos. Section 3 describes the performance modeling techniques used to analyze PKCROSS and PKTAPP, and Section 4 analyzes the performance implications. Sections 5 and 6 describe related work, present the conclusions of this analysis, and suggest areas for future research.

2. Kerberos and Public Key Variants

2.1 The Basic Operation of Kerberos

Kerberos is a network authentication scheme based on the early work of Needham and Schroeder [1]. Two popular Kerberos standards exist: Version 4 and Version 5 [2]. This paper considers the more current Version 5 and its variations.

Kerberos divides the world into realms, each with a single primary Key Distribution Center, back-up KDCs, applications servers, and user workstations. A single realm corresponds to a community of interest with a single security policy.

Many good, detailed descriptions of Kerberos protocol operation exist [3,4] and will not be repeated here. Briefly, the client—Alice—engages in a multiple-step authentication to obtain access to the application server—Bob. Alice must first obtain a ticket-granting ticket (TGT) to a centralized Ticket-Granting Service (TGS) offered by the KDC. She uses the TGT to obtain a service ticket to Bob. She presents the service ticket to Bob and authenticates herself by demonstrating

Proceedings of the IEEE Symposium on Security and Privacy (S&P'01)
1081-6011/01 $10.00 © 2001 IEEE

knowledge of a secret session key securely passed to her by the KDC.

A truly large enterprise may consist of many realms, and Alice may wish to gain access to an application server in a remote realm. To support "cross-realm" authentication, Alice's KDC and the remote KDC must have either a direct or indirect trust relationship (Kerberos Version 4 requires a direct trust relationship). This trust relationship is implemented by sharing symmetric keys among the KDCs.

If such a trust relationship exists, Alice may gain access to a remote server by first requesting a ticket to the remote realm's KDC from her local KDC. She will receive a ticket (a data structure encrypted with the KDC pair's shared symmetric key) to the remote TGS. When she presents the ticket to the remote KDC, she will receive a service ticket for an application server in the remote realm.

2.2 Motivation to Add Public Key Cryptography to Kerberos

Kerberos offers at least two significant benefits as a network authentication protocol. First, the message formats are defined such that the session keys are always included in some encrypted portion of the message. As a result, Kerberos servers do not need to store session keys or maintain a security association with each client. Kerberos is stateless; state is represented through the Kerberos tickets. Statelessness is extremely valuable from the standpoint of scalability.

The second benefit is that client user Alice is only required to provide her password once. Although many Kerberos request messages contain authenticators, the encryption is performed by Alice's workstation. As a result, Kerberos implements a "single sign-on" mechanism.

A potential limitation in Kerberos scalability is its reliance on symmetric key encryption [5]. Shared secrets must be established and maintained between every user and the KDC, between every application server and the KDC, and between remote KDCs. The use of public key cryptography shifts key management from the KDC to a Certification Authority (CA) [6]. Public key cryptography eliminates the need to establish a large number of shared secrets (i.e., symmetric keys) between KDCs, servers, and users.

Although the scaling merits of public-key- versus secret-key-based systems have not been definitively proven, several current proposals suggest adding public key cryptography to Kerberos and hence changing the key management model.

2.3 Proposals to Add Public Key Encryption to Kerberos

Internet drafts exist for three alternatives: (1) Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) [7], (2) Public Key Cryptography for Cross-Realm Authentication in Kerberos (PKCROSS) [8], and (3) Public Key Utilizing Tickets for Application Servers (PKTAPP) [9]. PKINIT is the core specification. Both PKCROSS and PKTAPP use variations of PKINIT message types and data structures to apply public key cryptography to different Kerberos authentication steps.

PKINIT. The PKINIT Internet draft specifies that considerable message content must be added to the initial Kerberos Version 5 exchanges to replace the user secret key authentication with public key authentication.

Figure 1 illustrates the flow of the PKINIT protocol. The client must send a public key certificate and a chain of certificates ("userCert" in the PKINIT specification) to establish trust between the user and the KDC and relay the user's public key. The client must also send an authenticating data structure ("SignedAuthPack") signed with the client's private key. This information is included in the Kerberos pre-authentication fields—defined in the specification to support extensions to the protocol.

The KDC verifies the client's identity by verifying the digital signature in the SignedAuthPack. The KDC replies to the client with a chain of certificates for the KDC's public key ("kdcCert"), the KDC's digital signature ("SignedReplyKeyPack"), and the session key ("encTmpKeyPack") encrypted with the client's public key. The client can confirm KDC identity by verifying its digital signature.

Mandatory variations allow the substitution of a certificate serial number for the certificate chain (assuming that the KDC already has a trusted copy of the user's certificate) and the use of Diffie-Hellman to establish a session key. PKINIT drafts have included several interesting options such as storing the user's private key on the KDC and allowing the user to generate the session key. Allowing the user to generate the session key would reduce load on the KDC and potentially change scaling characteristics.

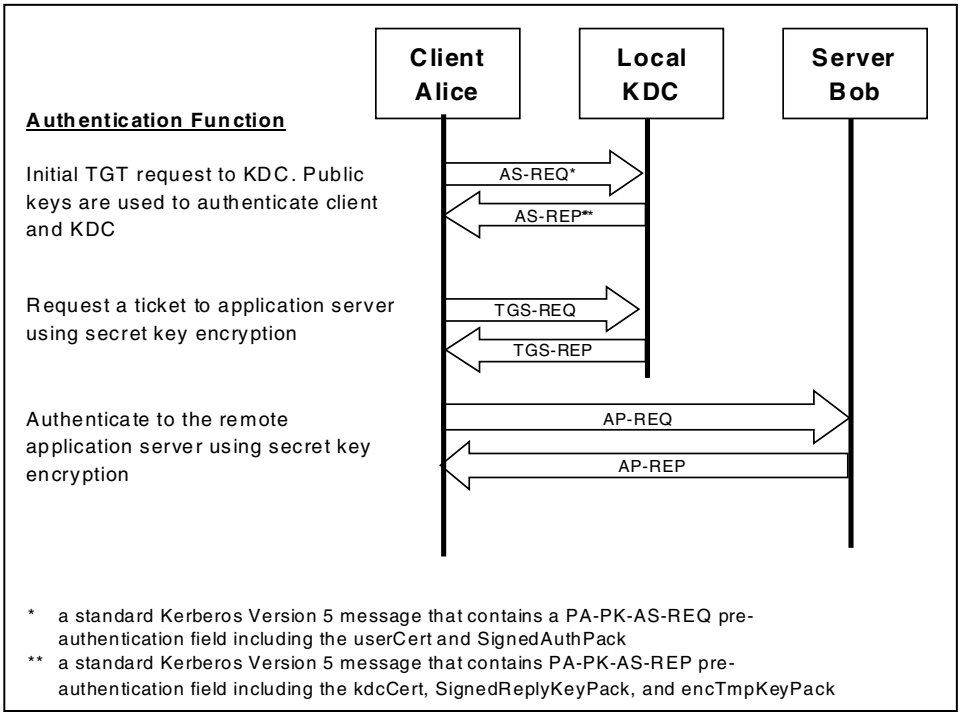

Figure 1. The PKINIT transaction flow

PKCROSS. While PKINIT addresses the issue of managing secret keys for a large number of clients, it does not address the issue of key management among a large number of realms. A logical extension of PKINIT is the use of public key encryption for multiple-realm, KDC-to-KDC authentications. This is the subject of the PKCROSS Internet draft specification.

Figure 2 illustrates a possible flow for PKCROSS authentication. PKCROSS picks up the multiple realm authentication at the point at which the client has already obtained a TGT. The client may or may not have authenticated to its local KDC using PKINIT (Figure 2 shows a secret-key-based TGT request). Assume that the client has requested access to a server in a remote realm. Its local KDC initiates a PKCROSS transaction with the appropriate remote KDC. With a few minor variations, the KDC-to-KDC authentication is performed using the PKINIT protocol.

One variation is that the remote KDC is responsible for generating a "special symmetric key it uses for PKCROSS requests" [8]. The local KDC can skip the explicit exchange with the remote KDC if it currently has an active, valid TGT sealed with a special symmetric key. Once the client possesses a remote TGT, it may request additional service tickets in the remote realm without involving the local KDC.

PKTAPP. In Kerberos, the KDC issues all TGS, remote KDC, and server tickets in its realm. Since most authentication transactions have to transit the KDC, it can become a performance bottleneck. Although secondary KDCs can be included in the system, they are typically used as backups in the event of a primary KDC failure. The PKTAPP Internet draft seeks to eliminate this potential bottleneck and reduce communications traffic by implementing the authentication exchange directly between the client and the application server.

This variation was originally introduced as the Public-key-based Kerberos for Distributed Authentication (PKDA) [10]. PKTAPP proposes to implement PKDA using the PKINIT-specified message definitions and exchanges.

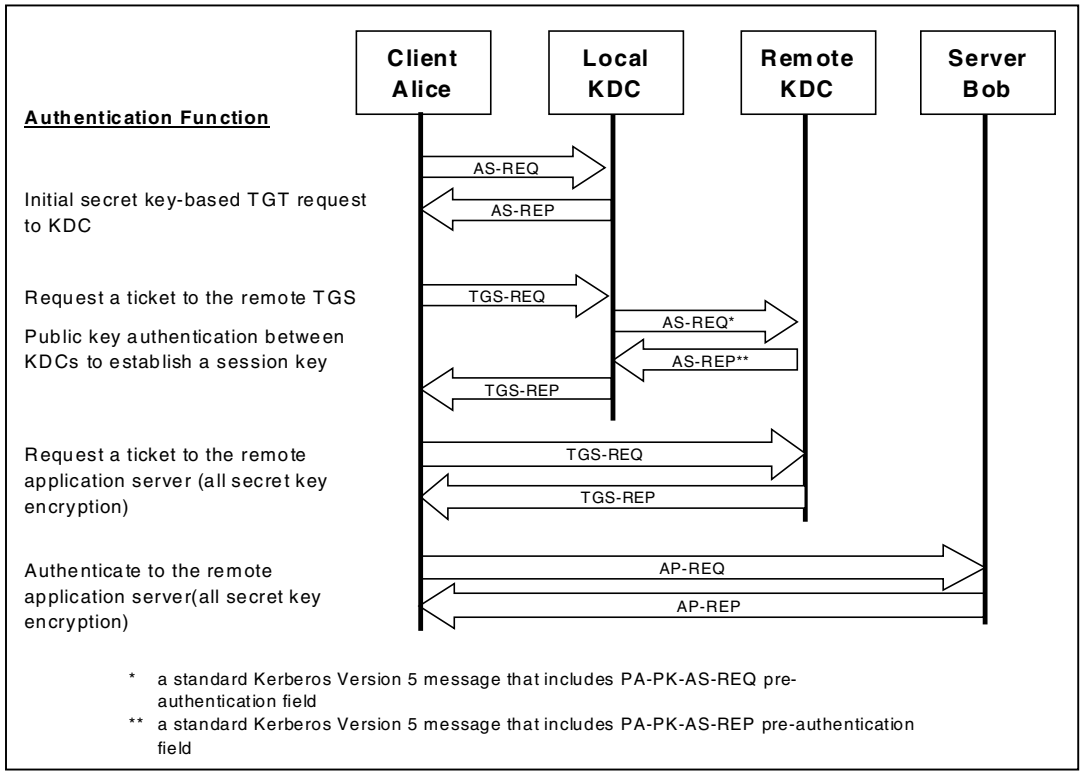

Figure 2. The PKCROSS transaction flow

PKTAPP is a more efficient protocol than traditional Kerberos from a message exchange perspective. The client may deal directly with the application server. Figure 3 illustrates the PKTAPP authentication message exchange. The AS-REQ message, the first message submitted by the client, contains the client's certificate chain and the identity of the service ticket requested. The server response, an AS-REP message, contains the server's certificate chain and the session key encrypted with the server's private key. After authentication, the client requests an application service ticket using a Kerberos Version 5 request. The entire authentication process is reduced to a total of two message pairs.

In the above public key extensions, there is no explicit requirement for pre-knowledge of identity between the client and the KDC or between the two KDCs. There is no need to pre-establish shared secrets or store a user record in a Kerberos database. The only basis for trust between these entities is the certificate chain.

These protocols substitute public key infrastructure as the management mechanism in lieu of sharing secret keys. Because of the additional processing requirements, a performance price is paid each time a public key calculation is substituted for a symmetric key calculation. Additionally, because the public key messages are larger than the corresponding Kerberos Version 5 messages and more likely to fragment, the PKINIT draft recommends the use of TCP as the underlying transport protocol. UDP, which has a significantly lower overhead, is the protocol commonly used for secret-key-based Kerberos implementations.

2.4 Using Closed Queuing Networks to Analyze Protocol Performance

The KDCs, application server, communications networks, and client workstations are finite resources that process workload while Kerberos authentication transactions are executing. One way to quantitatively analyze the performance of alternative public-key-enabled Kerberos variants is through the use of closed queuing networks [11], which represent each system resource with a queuing discipline and a stochastic service distribution.

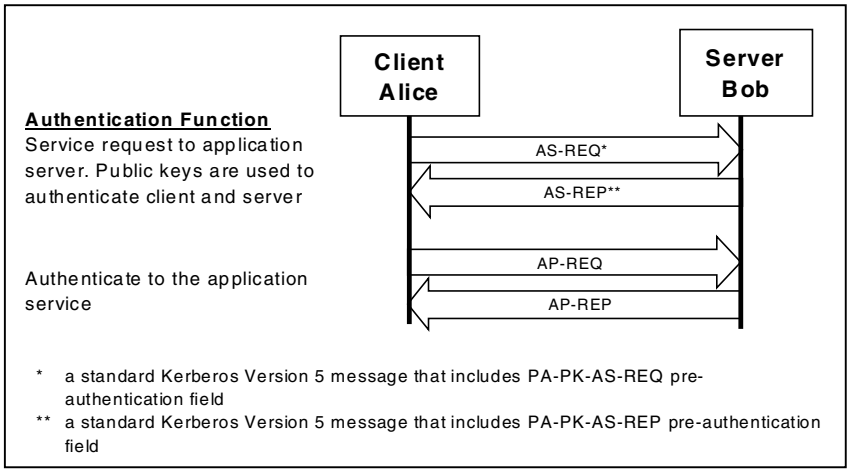

Figure 3. The PKTAPP transaction flow

Figure 4 presents the topology of a two-realm Kerberos closed queuing network model. Customers circulate among the servers in the closed network and sequentially wait for service, consume processing resources, and then proceed to the next service station.

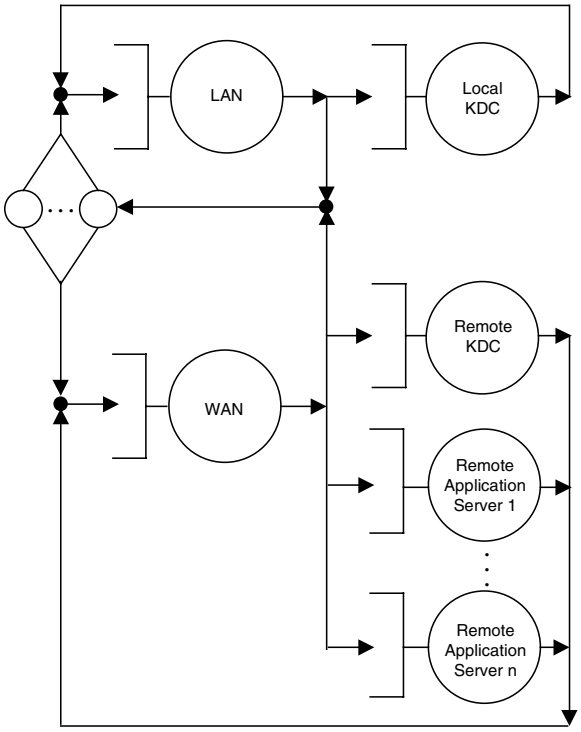

Figure 4. The topology of a two-realm Kerberos closed queuing network model

The topology envisioned in Figure 4 anticipates that the local KDC may be connected to the client by a local area network (LAN) and the remote KDC and application server may be connected by a wide area network (WAN). The mathematical solution to the queuing network produces metrics for each queuing station and the system as a whole—metrics such as the average number of customers, the average delay time, and the customer throughput. These metrics can be used to compare the performance of the alternative Kerberos proposals.

The brute-force mathematical solution to the closed queuing network can be obtained by enumerating all possible states (that is, all possible combinations of customers at each server) and solving the simultaneous set of equations representing balanced probability flow into and out of each state. This is called the global balance solution [12]. For any practical number of customers and servers, the state space is extremely large, and the global balance equations are impractical to solve.

If the closed queuing network meets a certain set of criteria—primarily concerning the independence of servers and customers—the closed queuing network can be represented by a "product form" solution [13]. In a product form solution, the probability that the system is in a given state $N = (n_1, n_2, ..., n_k)$ representing the number of customers at all k servers is proportional to the product of the marginal probabilities $P_i(n_i)$ that server i has n_i customers. The product form queuing network can be solved efficiently even for large numbers of servers and customers [11, 14].

A typical problem with this solution approach is the practicality of the assumptions required to achieve product form. For example, it is difficult to assume, when

analyzing authentication protocols, that all customers at a "first-come, first-served" resource have the same average service times. Public key encryption will be applied at specific points in the protocol. A message undergoing public key encryption will require much more processor service time than a message undergoing secret key encryption. Hence, it is impractical to assume that all messages arriving at a server will experience the same average service time. One way around this dilemma is to:

- assume that the server performing the encryption follows a processor sharing [14] queuing scheme,
- assume that messages undergoing public key encryption are in a different class [14] than messages undergoing secret key encryption, and
- implement a class-switching [11, 13] model in which a transaction will switch from one class to another as it requires different types of encryption and correspondingly different service times.

In the above construction, each closed set of customer classes forms an independent Markov chain [12], and a product form solution is maintained.

2.5 Authentication Performance Problems in Large Networks

The context for this analysis is authentication in a large, multi-realm network. Consider the following example as an illustration.

An intelligent software agent representing a law enforcement official is collecting information for an investigation. The agent "visits" on-line town halls and virtual sheriff's offices in a nationwide search for evidence and investigative intelligence. At each stop, the agent must show well-accepted electronic credentials. The source and level of these credentials will be used to grant the agent access to records tightly controlled for reasons of confidentiality and security. Further, the agent may be requested to pay for information with electronic currency. The infrastructure to support this type of electronic investigation will require scalable, robust authentication protocols.

The intelligent agent will probably be transiting multiple security realms in the course of the cyber investigation. Both PKTAPP and PKCROSS are candidates for the authentication protocol. A quick analysis might conclude that PKTAPP would have better performance characteristics because the agent would authenticate directly at the application server with only two message pairs. However, the agent may be interested in authenticating to several servers within a single realm—as would be the case for a visit to the cyber town hall, courthouse, police station, and sheriff's office for the same township.

If the agent uses PKCROSS, an expensive public key authentication would be required only once—between the local KDC and the remote KDC. After the cross-authentication and the provision of a TGT to the remote TGS, only secret key encryption calculations would be required. At some application-server-to-realm ratio, it would be more efficient to use the PKCROSS protocol.

This is the specific performance question we will explore: Under what circumstance is it more efficient to authenticate to a central KDC than to individual applications servers?

3. Methodology

Any analysis of PKTAPP and PKCROSS should start with reference implementations of the two protocols. Unfortunately, there are no such sources for PKCROSS and PKTAPP. Consequently, we began our analysis by building "skeleton" implementations of these two protocols.

The objective of the skeleton design was to consume computing resources similar to actual implementations of the protocol (resulting from communications, encryption, and message processing), but to avoid many of the complexities presented by implementing the real protocols. For example, we excluded error processing and many optional Kerberos features from the skeleton. The implementation also included more detail in the servers than in the clients because the servers are the shared resources.

We developed the skeletons in standard C and used the RSA reference library, RSAREF [15], for public key encryption and Karn's DES library [16] for symmetric key encryption. We implemented the KDCs and application servers on Microsoft Windows NT and the clients on Microsoft Windows 98. However, we did not use any operating-system-specific extensions in the C programs.

Figure 5 presents the software architecture. In the PKCROSS transaction, the client process steps through the standard Kerberos authentication message sequence to request service from an application server in a remote realm.

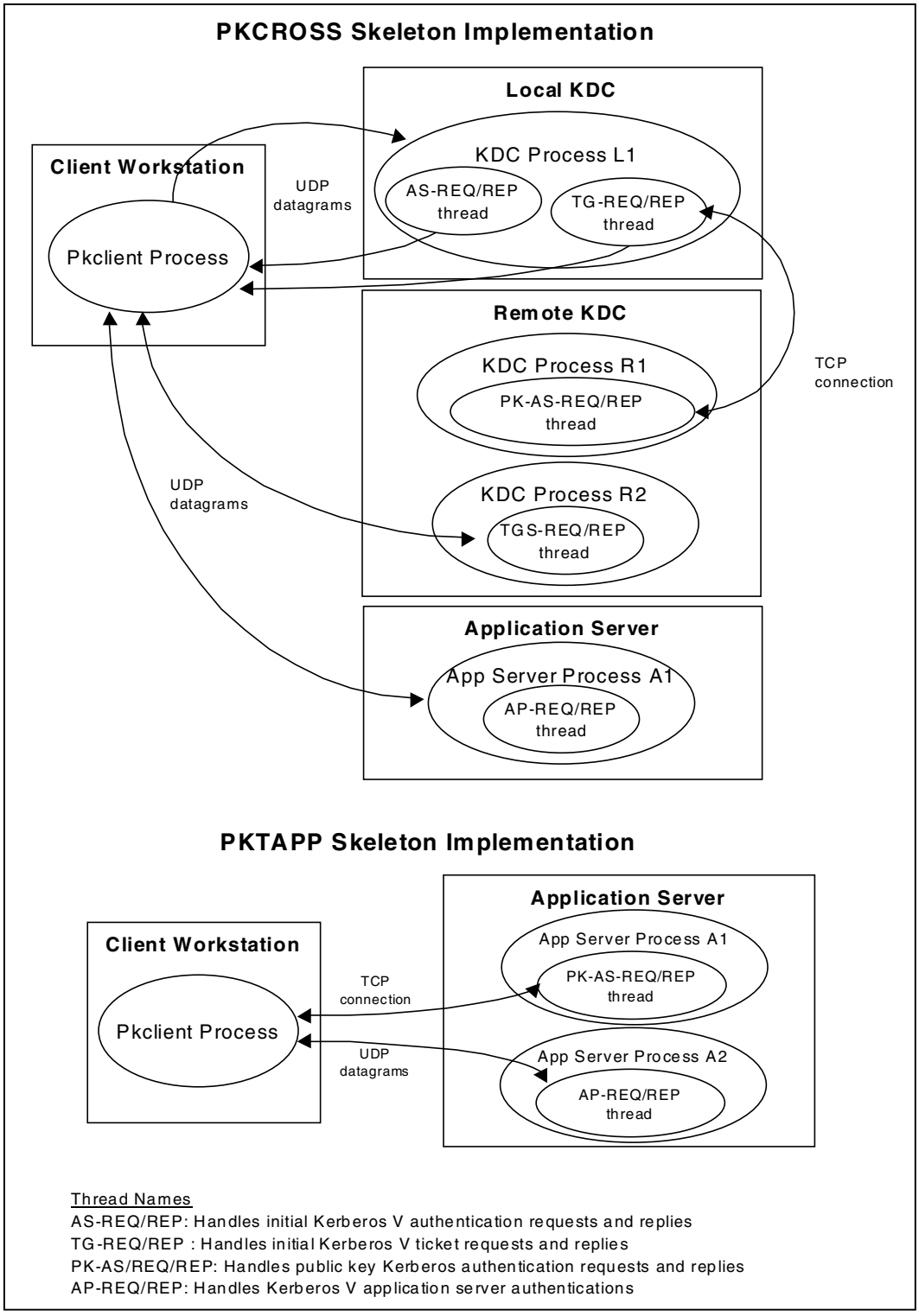

Figure 5. The PK-enabled Kerberos skeleton software architecture

The primary purpose of the skeleton software on the client is to issue requests, quickly confirm the validity of the response, and timestamp the transaction to report response time statistics. The client side processing has been simplified to focus on the shared resources: KDCs and Application Servers. The client process will loop through many transactions for the purpose of reporting average response time statistics.

A single process runs on the local KDC to accept client requests in UDP datagrams and cross-authenticate with the remote KDC using PKINIT. Two processes run on the remote KDC: one waits for standard Kerberos requests arriving as UDP datagrams, and the other opens a TCP listening socket and waits for PKINIT transactions. All KDC and Application Server processes are multi-threaded—when they receive a message, they dispatch a thread to process and respond to the request. In the final step of the transaction, the client authenticates to the Application Server using a ticket received from the remote KDC.

In the PKTAPP transaction, the client process has the same role and interacts with two server processes. It conducts a PKINIT exchange over a TCP connection to a multi-threaded process and obtains a service ticket. The client completes the authentication by sending a UDP datagram (a Kerberos Version 5 AP–REQ message) to a multi-threaded process running on the same physical server.

With the availability of measurable, skeleton versions of PKCROSS and PKTAPP, we followed a five-step approach to answer the questions posed in Section 2 concerning public-key-enabled Kerberos variants:

1. Establish a test bed to measure Kerberos service times. Figure 6 illustrates a multi-domain Kerberos test bed. We used instrumented code, an IP-layer packet monitor, and software monitors to capture the service times and message sizes for a cross-domain authentication transaction. The test bed uses simple, low-level PCs for all clients and servers and connects them with a 10-Mbps Ethernet LAN. All servers used the Microsoft Windows NT operating system; all clients used the Microsoft Windows 98 operating system.

2. Develop a closed queuing network model of Kerberos multidomain authentication. We constructed a queuing model, similar in topology to that illustrated in Figure 4, to represent the operation of Kerberos Version 5 with public key extensions. The queuing model uses a class-switching formulation so the service times can be varied for each step in the protocol.

3. Calibrate the queuing model against the test bed. We ran automated authentication scripts on each client workstation in Figure 6 to generate load on the servers. We measured transaction throughput and response times for several levels of workload submission. Then we entered similar workload profiles into the queuing model to calibrate the model against the test bed emulations. This step demonstrated the predictive accuracy of the queuing model.

4. Execute the model with different numbers of realms and servers. We executed the model with varying numbers of realms and servers per realm to observe the effects on throughput and response time.

5. Analyze sensitivity to changes in service time and network delay. Any conclusions about the situations favoring PKTAPP and PKCROSS performance will depend on the capacity and the resources that support the protocols, primarily server speed and network delay. We varied these parameters in our model in order to understand the dependencies.

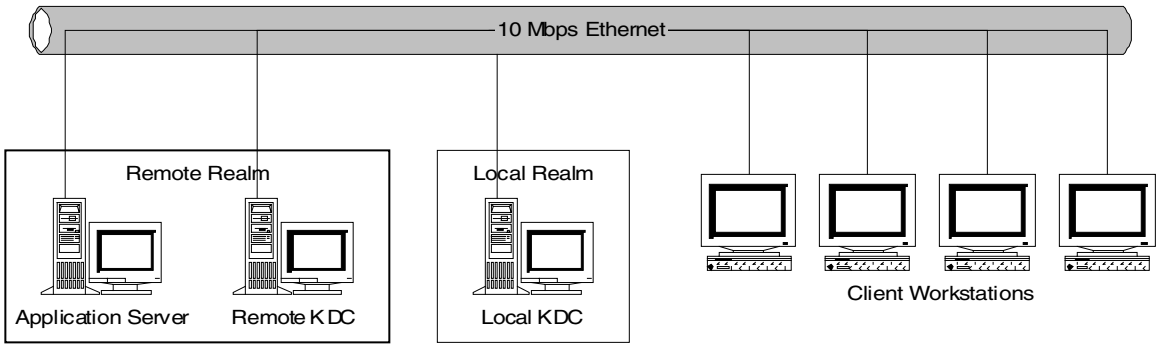

Figure 6. A multiple-realm Kerberos test bed

4. Analysis

4.1 Model Calibration

The flow of the PKCROSS and PKTAPP transactions follows the message exchanges illustrated in Figures 2 and 3. These baseline transactions are constructed with one application host in a remote realm. Because the environment under study is a large, multi-realm network, it is assumed that the client and KDCs must present certificates for authentication (that is, no parties store certificate serial numbers and local copies of certificates). Further, it is assumed that the remote server must validate two certificates in a chain corresponding to the certificate signed by the local CA and a certificate signed by the remote CA. The same assumptions are applied to the authentication of the local KDC to the remote KDC.

We configured the client, KDCs, and application server implementations to perform all encryption operations with 1024-bit RSA keys or standard DES. Table 1 summarizes the encryption operations performed. The number of public and private key operations for authentication to a single application server is identical. PKCROSS requires more secret key operations.

The model's predictive accuracy for PKCROSS and PKTAPP transactions is shown in Figures 7 and 8, respectively. Both graphs demonstrate good calibration between the model and observed test bed results. The predicted response times and throughputs are within 3 percent of measured results.

4.2 PKCROSS and PKTAPP Performance Predictions

We used the calibrated model to investigate performance with an increased number of application servers. As the number of application servers increases, the number of "visits" made to the corresponding servers in each transaction increases. The PKTAPP transaction will include an additional set of public key authentication calculations for every additional application server. In the PKCROSS transaction, there is only one public key authentication—between the local and remote realm KDCs—regardless of the number of application servers in the remote realm.

Figure 9 presents the comparative performance—transaction response time plotted as a function of throughput—for the PKCROSS and PKTAPP protocols as the number of application servers are increased. The transactions for PKCROSS represent authentication to one and sixteen application servers in the remote realm. The transactions for PKTAPP represent authentication to one, two, and four servers in the remote realm. We expect that uses similar to the example offered in Paragraph 2.5 could require authentication to four or more servers in a single realm.

The transaction rates for both protocols were increased until the overall response time became unstable and grew rapidly. In the PKCROSS transaction, the first bottleneck was the remote KDC processor, which had two processes running (one monitoring UDP traffic and the other listening for PKINIT transactions over TCP connections) and handled half the public key calculations for the KDC-to-KDC PKINIT exchange.

Table 1. A summary of the encryption operation performed for PKCROSS and PKTAPP authentication transactions

Authentication Transaction	No. Private Key Operations	No. Public Key Operations	No. Secret Key Operations
PKCROSS			
Client	0	0	7
Local KDC	2	3	5
Remote KDC	1	4	4
Application Server	0	0	3
Totals	3	7	19
PKTAPP			
Client	2	3	3
Application Server	1	4	4
Totals	3	7	7

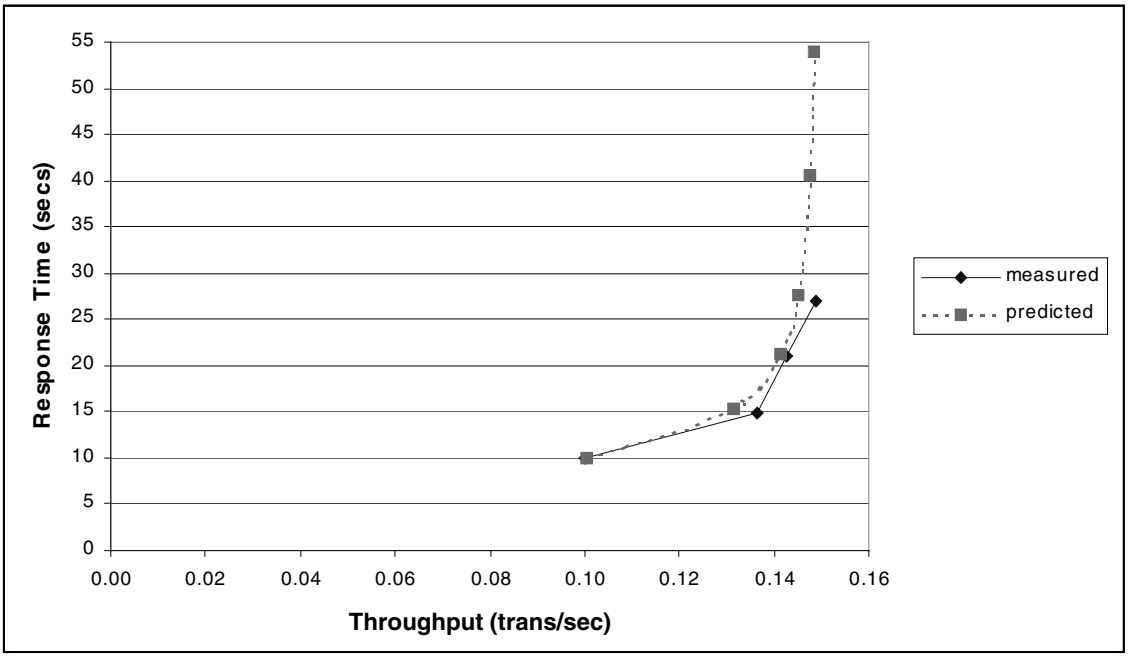

Figure 7. Calibration results for the PKCROSS transaction

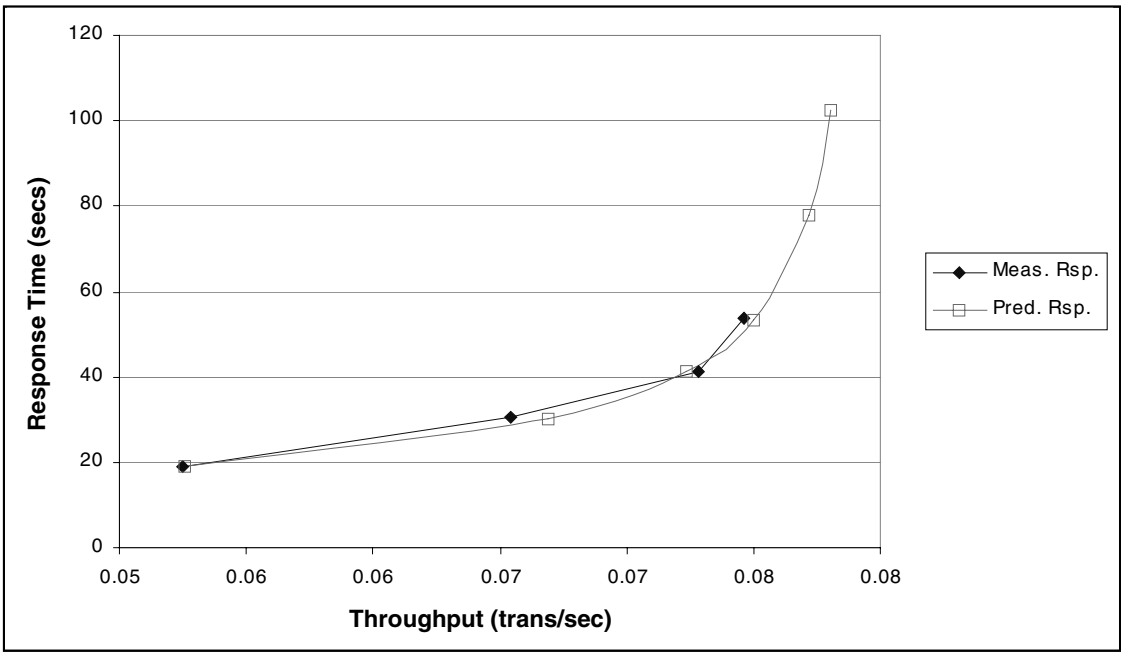

Figure 8. Calibration results for the PKTAPP transaction

Proceedings of the IEEE Symposium on Security and Privacy (S&P'01)
1081-6011/01 $10.00 © 2001 **IEEE**

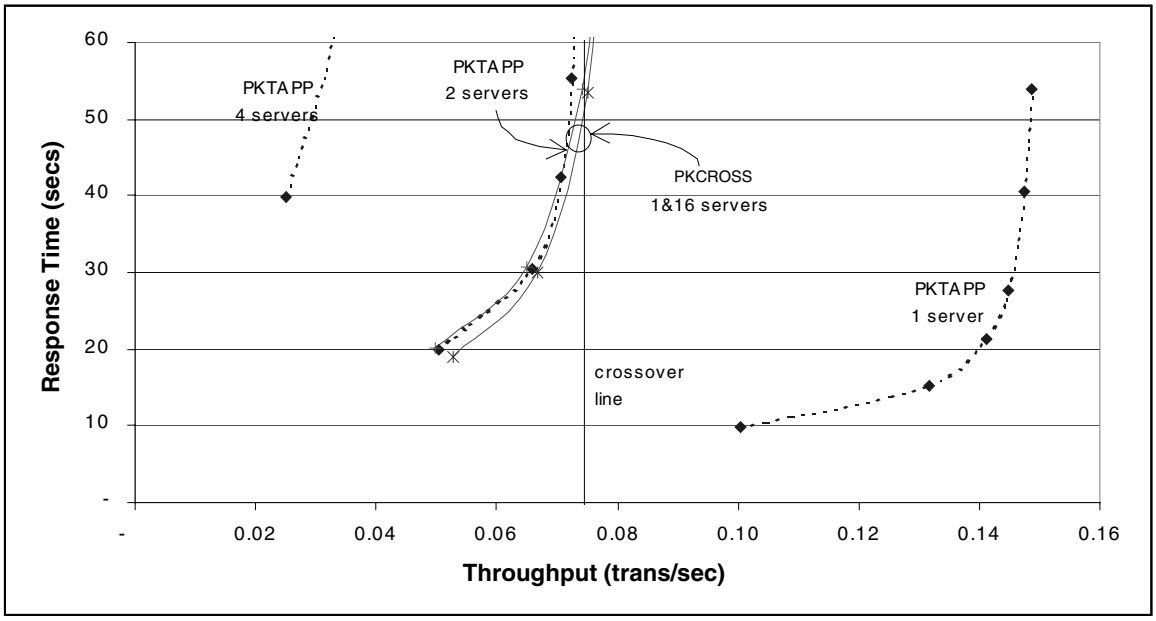

Figure 9. Comparative PKCROSS and PKTAPP performance

The next bottleneck, with very similar workload, was the local KDC. The application server was under-utilized—it conducted the final user authentication with only secret key encryption. Under PKCROSS, the KDCs remain the saturation point regardless of the number of remote realm application servers participating in the authentication.

The indifference of the PKCROSS protocol to number of servers in the remote realm is clear in Figure 9. The analysis demonstrates that, while PKTAPP is a significantly better performance choice for a single remote server, PKCROSS is significantly more stable for anything greater than two servers in the remote realm.

Our model accounts only for application server workload generated by authentications. In general, the application server will carry additional processing workload. If we had added that workload to the model, it would make PKCROSS look even more favorable.

4.3 Sensitivity to Resource Capacity

We call the number of servers per realm favoring PKCROSS performance over PKTAPP the "cross-over." The cross-over may vary with server and network capacity. The test bed was constructed with low-performance servers and a high-performance network—a 10-Mbps local area network has much better performance than the typical Internet connection.

We repeated the model variations studied in Section 4.2 over a range of server and network performance. We increased the performance of both KDCs and the application servers (that is, service times were reduced) by one and two orders of magnitude. As a result, we studied a range of processor performance that varied from a "1" to "100" SPEC CINT95 [17] rating—from a very-low-end Intel Pentium processor to a high-end server. Network performance ranged from LAN speeds to a network throughput of 12,750 bytes per second to characterize slow Internet links [18].

The results of our fast-processor/slow-network model are documented in Figure 10. This analysis indicates that performance benefits of PKCROSS when more than two application servers are accessed in the remote realm hold for increased processor capacity and reduced network throughput. It might appear that the increased network delay would favor PKTAPP because PKCROSS includes more message exchanges. The increased network delay does result in more separation between the PKCROSS response time curves for 1, 8, and 16 application server authentications. However, because PKTAPP sends large messages carrying certification chains to each application server, we observe the same cross-over result.

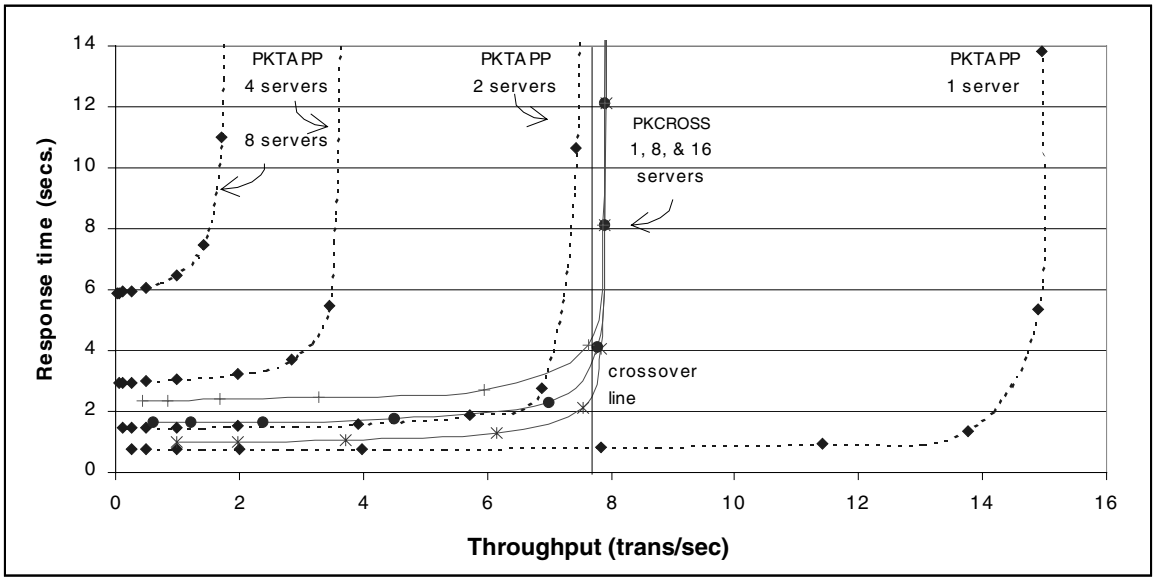

Figure 10. Comparative PKCROSS and PKTAPP performance with increased server capacity and decreased network throughput

5. Related Work

Developers of security protocols and encryption algorithms have long recognized the importance of performance. In many cases, researchers applied either complexity analysis to evaluate algorithm performance or measurement techniques to analyze system performance [19-22]. Protocol performance has become an increasingly important topic as security algorithms have become more commonly used in production and high-workload computing and networking environments.

Early work examined the impact of security protocols on network throughput, seeking to determine whether encryption calculations would put a damper on rapidly increasing data rates. In [23], Zorkadis identifies the communications performance impacts of five basic security services: authentication, access control, confidentiality, integrity, and non-repudiation. Zorkadis begins his exploration of the impacts by constructing a simple queuing model for secure communications.

Because Kerberos was the standard network authentication protocol in the Open Software Foundation's Distributed Computing Environment (DCE) [24, 25], it has been analyzed in that context. DCE security services have been benchmarked and analyzed in [26]. The performance characteristics of Kerberos have been loosely measured in some of its pilot applications [27], with good results.

The predominant role of public key cryptography in electronic commerce has motivated several performance studies. In [28], Apostolopoulos and Peris look at ways to reduce the impact of the private key encryption step in the Transport Layer Security (TLS) protocol. In [29], Menascé and Almeida use analytical modeling to assess the tradeoff between performance and security in e-commerce applications using protocols such as TLS and payment services such as SET. Lambert, in [30], performs a high-level, benchmark-supported analysis of performance improvements anticipated from the use of elliptic curve cryptography in e-commerce transactions.

Finally, in previous work, the authors analyzed public-key-enabled Kerberos [31] without the benefit of PKINIT and PKCROSS skeleton implementations. As a result, that analysis contained only speculative observations.

6. Summary and Future Work

We have demonstrated, through the use of validated analytical queuing models, the quantitative performance differences between two proposals to public-key-enable Kerberos: PKCROSS and PKTAPP. Our analysis shows that, over the range of server and network capacity studied, PKCROSS outperforms the simpler protocol, PKTAPP, for authenticating to more than one application server in a remote realm. This finding can be used to

guide a high-level protocol that combines both PKTAPP and PKCROSS to improve performance.

Use of such a high-level protocol would require that each application server provide support for both PKTAPP and traditional Kerberos. It would also require that the client know, a priori, the number of servers that will be authenticated to in a given realm.

Neither requirement is onerous. The application server could support both PKTAPP and traditional Kerberos on two well-known ports. PKTAPP would be listening on a TCP socket. Kerberos would be awaiting UDP datagrams.

The client, in the process of searching for information over a large number of servers, is often presented with a "hit-list" before beginning the process of server authentications. This would allow the client to use either PKTAPP or PKCROSS based on the number of servers for each realm. This type of scenario fits well within the criminal investigation example offered in Section 2.5.

Our analysis, as shown in Figures 9 and 10, provides a small range of variation in server and network capacity. The development of a more comprehensive heuristic for selecting between PKCROSS and PKTAPP remains as future work. We would also like to use the model to evaluate the performance of a larger variety of encryption algorithms including comparison to baseline, secret-key-based Kerberos, an AES version of Kerberos, and an elliptic curve version of public-key-enabled Kerberos. In addition, the analysis we conducted does not consider the performance implications of the practical aspects of key management in a PKI, such as CRL distribution and review.

Finally, we have shown that the class switching queuing formulation is an effective way to quantitatively analyze the performance of protocols that combine secret and public key cryptography. The application of this analysis technique to a broader range of protocols will also be the topic of future research.

7. References

[1] Needham, R.M. and M.D. Schroeder, *Using encryption for authentication in large networks of computers.* Communications of the ACM, 1978. 21(December 1978): p. 993-999.

[2] Kohl, J., *The Kerberos Network Authentication Service (V5)*, C. Neuman, Editor. 1993: http://www.ietf.org/rfc/rfc1510.txt?number=1510.

[3] Kaufman, C., R. Perlman, and M. Speciner, *Network Security, Private Communication in a Public World*. 1995, Englewood Cliffs, New Jersey: PTR Prentice Hall.

[4] Schneier, B., *Applied Cryptography*. Second ed. 1996, New York: John Wiley & Sons, Inc. 758.

[5] Ashely, P. and B. Broom, *A Survey of Secure Multi-Domain Distributed Architectures*, 1997, Queensland University of Technology, Faculty of Information Technology.

[6] RSA, S., *Understanding Public Key Infrastructure (PKI)*, 1999, RSA Security Inc.

[7] Tung, B., *et al.*, *Public Key Cryptography for Initial Authentication in Kerberos*, January 15, 2001 (expiration): http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-12.txt.

[8] Tung, B., *et al.*, *Public Key Cryptography for Cross-Realm Authentication in Kerberos*, 1998: http://www.internic.net/internet-drafts/draft-ietf-cat-derberos-pk-cross-03.txt.

[9] Medvinsky, A., *et al.*, *Public Key Utilizing Tickets for Application Servers (PKTAPP)*, 1997: http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-tapp-03.txt.

[10] Sirbu, M.A. and J.C.-I. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. in Symposium on Network and Distributed System Security. 1997. San Diego, California: IEEE Computer Society Press.

[11] Bruell, S.C. and G. Balbo, *Computational Algorithms for Closed Queuing Networks*. The Computer Science Library, ed. P.J. Denning. 1980, New York: Elsevier North Holland, Inc.

[12] Gross, D. and C.M. Harris, *Fundamentals of Queuing Theory*. Third ed. 1998, New York: John Wiley & Sons, Inc. 439.

[13] Baskett, F., *et al.*, *Open, Closed, and Mixed Networks of Queues with Different Classes of Customers.* J. ACM, 1975. 22(2).

[14] Menasce, D.A., V.A.F. Almeida, and L. Dowdy, W., *Capacity Planning and Performance Modeling*. 1994, Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[15] Johnson., M., *North American Cryptography Archives*, October 26, 2000 (last accessed): http://cryptography.org/cgi-bin/noexport.cgi.

[16] Karn, P., *Index of /cryptocd/source/cyphers/des/c/karn*, October 26, 2000 (last accessed): http://www.cryptocd.org/cryptocd/source/cyphers/des/c/karn/.

[17] Standard, P.E.C., *SPEC CPU95 Benchmarks*, July 2000: http://www.spec.org/osg/cpu95/.

[18] Menasce, D.A. and V.A.F. Almeida, *Capacity Planning for Web Performance*. 1998: Prentice-Hall Inc.

[19] Bosselaers, A., *Fast Implementations on the Pentium*, in *http://www.esat.kuleuven.ac.be/~bosslae/fast.html*.

[20] Schneier, B., *et al.*, *Performance Characteristics of the AES Submission*, 1999.

[21] Bassham, L.E., *Efficiency Testing of ANSI C Implementations of Round 1 Candidate Algorithms for the Advanced Encryption Standard*, 1999, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

[22] Blaze, M., *High-Bandwidth Encryption with Low-Bandwidth Smartcards.* Lecture Notes in Computer Science, 1996. Number 1039.

[23] Zorkadis, V. *Security versus Performance Requirements in Data Communications Systems*. 1994.

[24] OpenGroup, *Distributed Computing Environment*, 1998, http://www.camb.opengroup.org/dce/.

[25] OpenGroup, *DCE 1.1: Authentication and Security Services*, 1997: http://www.opengroup.org/publications/catalog/c311.htm.

[26] Martinka, J., *et al. A Performance Study of DCE 1.0.1 Cell Directory Service: Implications for Application and Tool Programmers*. in *Lecture notes in computer science*. 1993.

[27] Stallings, W., *Kerberos Keeps the Enterprise Secure*, in *Data Communications*. 1994. p. 103-111.

[28] Apostolopoulos, G., V. Peris, and D. Saha. *Transport Layer Security: How much does it really cost?* in *IEEE INFOCOM*. 1999.

[29] Menasce, D.A. and V.A.F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. 2000: Prentice Hall.

[30] Lambert, P., *Elliptic Curve Cryptography Delivers High Performance and Security for E-commerce.* Computer Security Journal, 1998. XIV(4): p. 23-29.

[31] Harbitter, A. and D. Mensasce. *Performance Issues in Large Distributed System Security*. in *Computer Measurement Group 98*. 1998. Anaheim, CA.