Implementing a Prototype of the Security Framework for Distributed Brokering Systems

Yan Yan, Yi Huang, Geoffrey Fox, Ali Kaplan, Shrideep Pallickara, Marlon Pierce and Ahmet Topcu
(yayan,yihuan,gcf,alikapla,spallick,marpierc,atopcu}@indiana.edu
Community Grid Labs, Indiana University.

1.0 Introduction

Security is an important element of any system design. Entities supported by a system need to securely interact with each other. The problem gets even more complex in the context of a distributed system, where the underling infrastructure of messaging nodes, which we call "brokers," needs to carry these interactions securely. Furthermore, it is entirely possible that some brokers communicate with each other over communication channels that do not encrypt network traffic. Thus, even if two entities are connected to brokers, and they have secure communications channels with their brokers, when their communications traverse over insecure channels, the security of their interactions is compromised.

In [1] we presented a security framework that is appropriate for distributed brokering systems. The framework provided for secure communications over insecure links, and ensured that only authorized entities are allowed to view entity interactions. In this paper, we present our prototype implementation of this framework. The paper presents implementation details of key components within the system. We have also performed a series of experiments that would affect the design of components within the system, as well as the encryption strategies chosen by entities within the system.

This paper is organized as follows. In section 2.0 we present an overview of the related work in this area. Section 3.0 presents an overview of NaradaBrokering [2-10] and the security framework. Section 4.0 outlines the functions and issues that may affect the design of various building blocks in the system. Section 5.0 presents results pertaining to aspects of end-to-end secure interactions within the system. These results are then used to formulate a set of recommendations for secure interactions for different applications. Finally we outline our proposed future work in this area and set of conclusions, based on the work outlined in this paper.

2.0 Related Work

Peer-to-peer (P2P) [11] systems incorporate several strategies that address secure interchange while incorporating strategies to incorporate trust and reputations. The Project JXTA [12] security model is similar to the PGP "Web of trust" [13]. Current implementations provide TLS (Transport Level Security) for p2p interactions. The default cipher suite used in JXTA is RSA [14] 1024-bit for asymmetric keys, 3DES [15] for symmetric keys and SHA-1 for computing message digests. By localizing p2p interactions during search/discovery, JXTA attempts to limit bad behavior from other peers. Groove [16] provides excellent P2P security by securing shared spaces, which comprise documents, messages, etc. Incremental changes to a shared space object are transmitted to authorized peers in a secure way. In Groove the default asymmetric algorithm (modulus) for authentication and keys exchanges is ElGamal [17] (1536-bit modulus), while the default symmetric algorithm (key size) used for encryptions and decryptions is MARC4 [18] (192-bit key). Systems such as http://www.advagato.org incorporate trust metrics to support reputations while defeating scenarios where users band together to boost reputation scores. The Free Haven system [19] provides strategies for incorporating accountability while maintaining peer anonymity. Each server in Free Haven maintains values pertaining to reputation and credibility, while broadcasting referrals in some cases. Legion (http://www.cs.virginia.edu/~legion/) is a long-standing research project for building a "virtual computer" out of distributed objects running on various computing resources. Legion objects communicate within a secure messaging framework [20] with an abstract authentication/identity system that may use either PKI [21] or Kerberos [22]. Legion also defines an access control policy on objects.

There are many emerging issues pertaining to security in XML-based Web Services. Although normally presented as a client-server style system, Web Services may be equivalently viewed as an XML message-based communication framework. We may exchange Web Service messages in a distributed system just as easily as between clients and servers. WS-Security [23] from IBM and Microsoft outlines a proposed architecture to address the gaps between existing security standards and Web Services such as SOAP [24]. By abstracting security services, the WS-security model also serves to unify security technologies such as PKI and Kerberos. Security specifications for Web Services are just starting to emerge, but generally follow the same approach: the message creator adds a signed XML message containing security statements to the SOAP envelope. The message consumer must be able to check these statements and the associated signature before deciding if it can execute the request. The Security Assertion Markup Language (SAML) [25] from OASIS deals with the standard representation of security data – authentication, authorization and attribute – which would be recognized by different application security services irrespective of the security technology or policy that is deployed. SAML is designed to work with W3C specifications such as XML Signature [26], XKMS (XML Key Management Specification) [27] and SOAP.

GKMP [28] outlines an architecture for the management of cryptographic keys for multicast communications. Brokering systems also may be used to implement network multicasting. Ref [29] discusses strategies for reducing the number of encryptions required to preserve confidentiality, between an end-point broker and its subscribing entities, in the context of Content based publish-subscribe systems.

The Grid Security Infrastructure (GSI) [30] approach treats secure end-to-end connections as a sequence of secure point-to-point connections. The problem GSI addresses is where a user may need to invoke a particular service through one or more proxy servers. GSI breaks this request into a chain of point-to-point invocations, with the user's initial (proxy) credential used to create a sequence of proxy key pairs, with each key pair being delegated limited authority to invoke a remote service. The Akenti system [31] addresses the important problem of authorization of resources in a distributed system with multiple stakeholders. Akenti provides an XML access policy language that is transmitted using X.509 policy certificates. Ref [39] outlines strategies for securing computational portals.

3.0 NaradaBrokering and Security Framework Overview

NaradaBrokering is a distributed brokering system, which provides support for centralized, P2P and distributed interactions. The smallest unit of the messaging infrastructure, which can run on a network of cooperating nodes, is the broker. Each broker is responsible for processing events (specialized messages with additional headers), computing destinations and making decisions to facilitate efficient routing.

In NaradaBrokering the broker nodes are organized in a cluster-based architecture. The cluster based architecture allows the system, to scale, to support clients of arbitrary size, while allowing individual broker nodes to compute alternate routes in response to node failures. NaradaBrokering provides intelligent routing of events within the system by selectively deploying brokers and communication links to aid disseminations.

In NaradaBrokering we rely on message level security. To review NaradaBrokering's security strategy briefly, all messages (or events, which are specialized messages) have a topic associated with them. Messages are encrypted with a topic-key, which is different for different topics. The system ensures that only authorized entities hold that topic-key, which will be used to decrypt encrypted messages. The framework allows strategies where the keys used for encrypting/decrypting messages could be based on symmetric or asymmetric keys. In addition to topic keys, each entity also contains a public/private personal-key pair. These are used to sign messages that allow entities/brokers to verify the source and integrity of the message.

Of course possession of a valid topic key is necessary to decrypt the messages. The framework also deals with a variety of attack scenarios such as man-in-the-middle, denial of service attacks and replay attacks. The framework also incorporates strategies to detect security compromises and deals with issues pertaining to key invalidations and generations, based on personal-key and topic-key compromises within the system.

The framework provides for end-to-end security and ensures that the only place where the messages are seen, in plain text, are at the originator of the content and authorized entities, who posses valid keys, to decrypt the encrypted content. The scheme allows interactions to traverse over insecure links and also deals with the presence of rouge brokers within the system.

4.0 Implementing components of the Security Framework

In this section, we discuss implementation and design issues pertaining to implementing the key components of the security framework. First, there is the Key Management Center (KMC) which is responsible for key management functions within the system. Then there is also the Access Control List (ACL) module, which is responsible for maintaining information pertaining to authorizations within the system. This is depicted in Figure 1. We now discuss these components in detail.

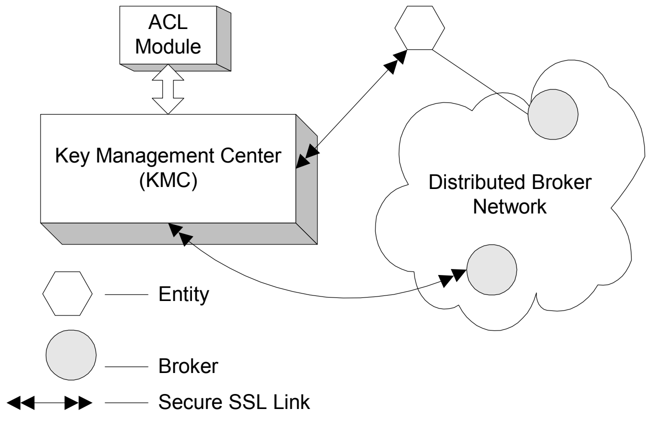

Figure 1: KMC, ACL and interactions

4.1 Key Management Center (KMC)

The KMC is responsible for functions pertaining to key management. This includes creating, storing and retrieving topic keys. Though in most practical situations entities would use symmetric keys for securing messages, the KMC also accommodates situations where the entities might use asymmetric keys for secure interactions.
Brokers and entities alike can contact the KMC to verify the signature and authorizations of message originators. All interactions that entities and brokers have with the KMC are SSL based. SSL not only secures exchanges, which include key distributions to requesting entities, but also defeats man-in-the-middle attacks during key exchanges and replay attacks on the KMC. NaradaBrokering's transport framework [32] includes support for SSL. These SSL links posses the ability to communicate over firewall, authenticating proxy and NAT boundaries. We now proceed to outline the major functions provided by the KMC

Creation of topic keys – The topic-keys generated could be either symmetric or asymmetric keys. The keys are created only after it is determined that the entity is authorized to create the topic. In the case of asymmetric keys associated with a topic, the appropriate public/private topic-keys are routed to entities depending on whether they have publish or subscribe permissions.

In their interactions with the KMC, entities can specify the type (symmetric/asymmetric) of key, the algorithm used to generate the key and the provider of the cryptographic packages. This feature is supported in our implementation for two reasons. First, supporting a large number of cryptographic package providers will allow entities to leverage implementations of cryptographic packages across providers. For example Sun's JCE [33] extension does not include an implementation of AES [35], while IAIK [34] JCE extension provides that implementation. Similarly, performance of the same cryptographic algorithm may vary across different providers.

IAIK's JCE Extensions				**Sun's JCE and JSSE Extensions**			
Symmetric Keys		**Asymmetric Keys**		**Symmetric Keys**		**Asymmetric Keys**	
Algorithm	Key Sizes	Algorithm	Key Sizes	Algorithm	Key Sizes	Algorithm	Key Sizes
DES	64	RSA	512, 1024, 2048, 4096	DES	56	RSA	512, 1024, 2048
3DES	192			3DES	112, 168		
AES	128,192,256			AES	NA		
RC2	40, 128			RC2	NA		

Table 1: Summary of different crytpgraphic providers/ functionalities supported by KMC prototype

Second, different entities and applications may sustain/accept different security/performance tradeoffs. While 4096-bit asymmetric key encryptions may be appropriate for some transactions, the performance degradations associated with supporting secure audio/video conferencing may render such a scheme unusable. Such systems may use a symmetric key algorithm such as AES (with 128-bit secret key) or DES [36] (56-bit secret key) to support encrypted communications.

Table 1 provides a list of providers, symmetric/asymmetric algorithms and corresponding key sizes supported by the KMC's prototype implementation.

Regeneration of topic keys – In response either to the detection of a security compromise or preemptive strategies mandated by the QoS associated with a given topic or entity, topic keys need to be regenerated. Of course the properties associated with the topic-key such as type, algorithm and provider would remain the same unless specified otherwise. The algorithms, key-size and provider may change due to performance or cryptographic (sometimes vulnerabilities in some algorithms might be revealed) reasons.

Verifications – Entities and brokers will interact with the KMC from time to time to confirm the veracity of a signed message's signature. This interaction is also used to verify if the entity in question is indeed authorized to issue messages to the topic contained in the message.

Key Stores and Retrievals – The KMC is also responsible for storing the generated topic keys securely onto storage and retrieving relevant keys from storage when a need arises to do so. Depending on the number of keys managed by a KMC it would not be feasible to maintain keys in memory, storage and retrieval times thus become important factors under these conditions. There are a variety of keystore implementations that we could use. For our investigations we evaluate two such keystores – one is Sun's JSK keystore, while the other is the IAIK keystore. Section 4.3 provides performance results pertaining to keystore performance, under different scenarios, which form the basis of certain design decisions pertaining to the keystore.

4.2 Access Control List (ACL) module

The KMC interacts with the ACL module to confirm permissions associated with an entity. The ACL module and the KMC reside in the same process. Associated with every entity, the ACL module keep track of the topic's the entity has subscribed to, along with the permissions it has associated with each topic, such as publish, subscribe and create. The ACL also maintains another view, where the KMC can query the ACL module to retrieve the list of valid entities authorized to retrieve the associated topic key (s).

These functionalities are implemented using Hashtables. The implementation included with JDK provides synchronized accesses to all functions supported by the Hashtable. This feature is useful especially when multiple read/writes are occurring concurrently.

The ACL module also needs to ensure that this information pertaining to entities, topics and accompanying access permissions, are stored securely. Only the process running the ACL module should be able to view the contents of this stored data. While storing to file, the various Hashtable contents need to be serialized and then encrypted prior to storage. During initializations, the serialized representations can then be used to construct the object.

The authorization and access control schemes can get arbitrarily complex when issues such as trust propagation and reputations are incorporated into the scheme for access controls. This feature has not yet been built into the system.

4.3 Some performance measurements from the KMC prototype

The KMC is responsible for storing symmetric/asymmetric keys. Public personal-keys reported by entities are stored by the KMC in the Certificate format. Keystores are a popular format to store both symmetric and asymmetric keys. The keystore file and every key maintained by the keystore are password protected. Keys have an alias associated with them which is used to retrieve the corresponding keys.

We compared the performance of different kinds of keystores based on different criteria such as affect on file sizes, storage time and retrieval times associated with individual keys. We compared two keystores – one was the JSK keystore from SUN JCE and the other was the IAIK keystore from IAIK. The symmetric key formats we used in our experiments included – 3DES (192 bits), AES (128 bits, 192 bits and 256 bits) and RC2 (40 bits and 128 bit). The asymmetric keys format we used were – RSA 512 bits, 1024 bits and 2048 bits. All reported results were based on an average of operations that were performed 100 times. The experiments were conducted on Pentium-4 2.4GHz CPU, 512 MB RAM machine. The OS on this machine is Windows XP. The runtime JVM for the processes involved was JRE 1.4.1.

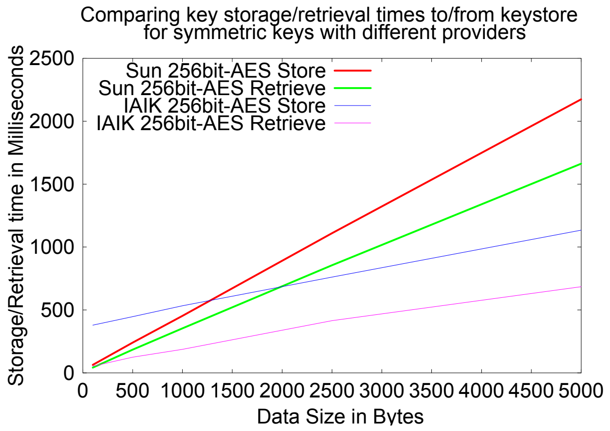

Figure 2: Storage/retrieve for symmetric keys

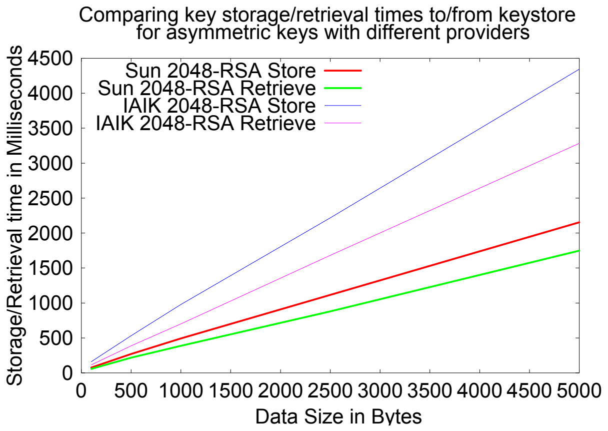

Figure 3: Storage/retrieve for asymmetric keys

Figures 2 and 3 depict the storage/retrieval times associated with symmetric and asymmetric keys, with different key sizes and different providers, respectively. Table 2 summaries the results depicted in Figures 2 and 3. We also performed similar measurements for RSA (512-bit, 1024-bit), AES (128-bit, 192-bit), 3DES (192-bit) and RC2 (40-bit, 128-bit) based keystores for each (whenever possible) of the two providers. The results followed similar patterns, and the storage/retrieval times would have made discernment of trends difficult to observe if there were to be plotted on the same graph. Table 2 summarizes the results of our benchmarks pertaining to keystores.

Key Type	**File size**	**Retrieval**	**Storage**
Symmetric	IAIK is better (about 5 times smaller)	IAIK is better (about 2 times faster)	SUN is better for key numbers<1000, IAIK is better for key numbers>1000
Asymmetric	IAIK & SUN provide similar Performance	SUN is better (about 2 times faster)	SUN is better (about 2 times faster)

Table 2: Summary of results from keystore measurements

Based on the results, outlined in Table 2, we can see that SUN JCE's JSK keystore has better performance with asymmetric keys. The IAIK keystore has better performance with symmetric keys. In our final implementation we plan to separate the storage of topic keys and personal public keys into two different keystores, using the SUN JCE's JSK keystore to save the asymmetric keys and the IAIK keystore to store symmetric keys.

5.0 Performance results for End-to-End Security

End to end security is achieved in the system by ensuring that only authorized entities can issue messages while at the same time ensuring that only similarly authorized entities can view the contents of the message. Every entity can of course confirm whether the message in question has been tampered with or has been issued by an authorized entity. In this section we introduce results pertaining to the times it takes to encrypt messages, compute message digests and sign them, and finally verify the message signature and decrypt contents of encrypted message.

The experiments were performed on a Windows 2000 machine (Pentium-3, 1.5 GHz, 512 MB RAM). The runtime environment for all processes involved is JRE 1.4.1. We also used a high resolution timer for measuring certain operations. The cryptographic provider which we used in these experiments is IAIK. Figures 4 and 5 present the encryption times associated with different message sizes for different cryptographic algorithms (64 bit DES, 192 bit 3DES, and 128 bit AES). The points in the graphs represent the average value of the operation being performed 1000 times. Figure 4 outlines results that are memory optimized as a result of the key being resident in memory during computations. Figure 5 presents the similar case, when the key used for encryptions are not memory resident. In the optimized case the encryption time varies between 50-1650 microseconds, while in the un-optimized case the numbers vary between 1-17 milliseconds for the cases measured. We also measured the decryption time under identical scenarios for both the normal and memory optimized cases. For all algorithms the decryption times were almost similar to the corresponding encryption times.

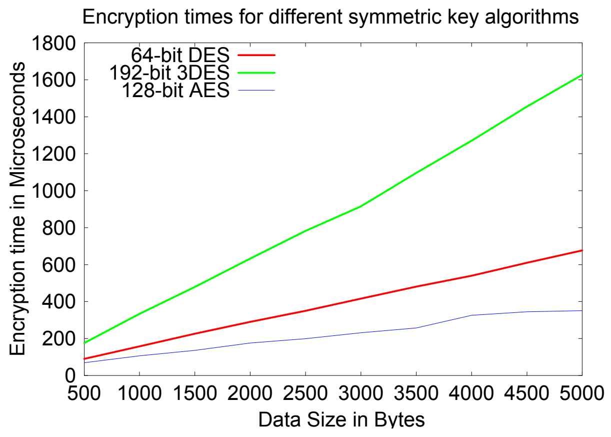

Figure 4: Encryption times for different message sizes (Memory Optimized)

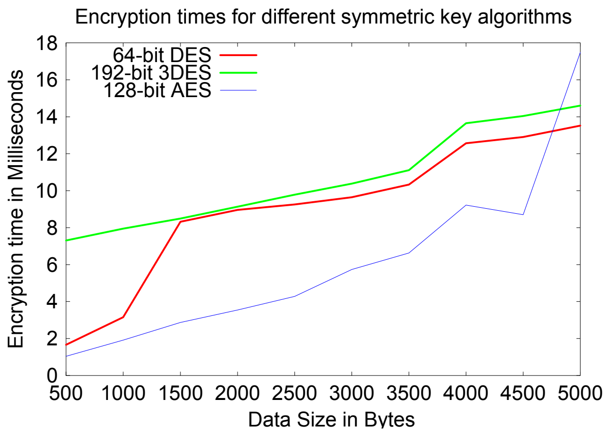

Figure 5: Encryption times for different message sizes

Figure 6 depicts the time associated with computing message digests using the MD5 [37] and SHA-1 [38] hash functions. MD5 generates a 128-bit message digest, while SHA-1 generates a 160-bit value. Figure 7 outlines the time associated with signing (encrypting message digest with personal private key) a message and verifying the message's signature. We performed the signing/verification process for 4 different RSA key sizes viz. 512-bit, 1024-bit, 2048-bit and 4096-bit. Though in the figure we report values based on signing SHA-1 hash values, we have also computed numbers for MD5 based hash values. The signing/verification times associated with MD5 based message digests are similar to those of SHA-1 based signature/verification times.

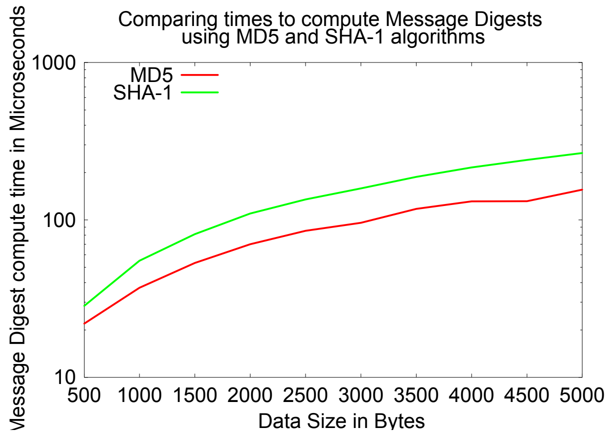

Figure 6: Comparing Message Digest compute times with different digest algorithms

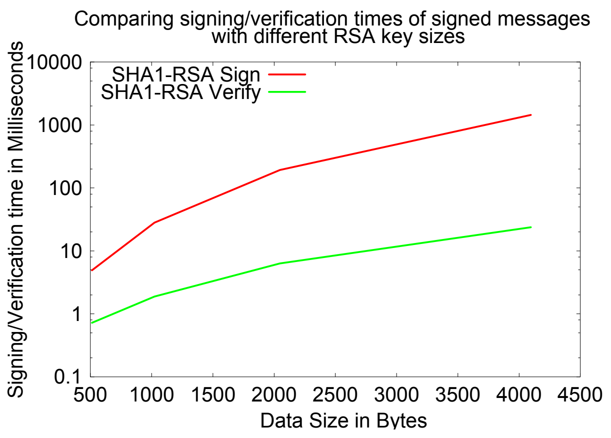

Figure 7: Signing/Verification times with different key sizes.

6.0 Conclusions and Future Work

In this paper we outlined a prototype implementation of the security framework outlined in our earlier work. We believe the results of the experiments from our prototype can be used by other researchers in the development of similar prototypes. The results also provide a precursor to testing secure communications, using various applications under different scenarios, which would help us at arriving at better heuristics as far the performance/security tradeoffs are concerned.

We have not yet implemented the Distributed KMC architecture outlined in [1]. Implementing this distributed KMC and the accompanying distributed ACL is the next implementation effort. Also under implementation are the strategies to detect a security compromise by issuing random authentication challenges or queries, and matching entity responses to stored ones.

References

1. A Security Framework for Distributed Brokering Systems, Shrideep Pallickara, Marlon Pierce, Geoffrey Fox, Yan Yan, Yi Huang.
2. The NaradaBrokering System http://www.naradabrokering.org
3. A Middleware Framework and Architecture for Peer-to-Peer Grids. (To appear) Proceedings of ACM/ IFIP/USENIX International Middleware Conference Middleware-2003. Shrideep Pallickara and Geoffrey Fox
4. NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Geoffrey Fox and Shrideep Pallickara. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April'03.
5. The Narada Event Brokering System: Overview and Extensions. Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, June 2002. pp 353-359.
6. A Scaleable Event Infrastructure for Peer to Peer Grids. Geoffrey Fox, Shrideep Pallickara and Xi Rao. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.
7. "JMS Compliance in the Narada Event Brokering System." Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.
8. Grid Services for Earthquake Science. Fox et al. Concurrency & Computation: Practice & Experience.14(6-7):371-393.
9. "Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service". Bulut et al. Proceedings of the IASTED International Conference on Communications, Internet, and Information Technology, 2002, in US Virgin Islands.
10. An Event Service to Support Grid Computational Environments. Geoffrey Fox and Shrideep Pallickara. Journal of *Concurrency and Computation: Practice & Experience*. Volume 14(13-15) pp 1097-1129.
11. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Edited by A. Oram. O'Rielly Press, CA. March 2001.
12. Project JXTA from Sun Microsystems. Security Homepage. http://security.jxta.org/
13. Atkin, D., Stallings, W., and Zimmermann, P.,"PGP Message Exchange Formats," RFC 1991 (August 1996)
14. R.L. Rivest, A. Shamir, and L.M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, volume 21, pages 120-126, February 1978.
15. ANSI, "American National Standard for Financial Institution Key Management (wholesale)," ANSI X9.17 (1985)
16. Groove Networks Inc. Desktop Collaboration Software. http://www.groove.net/
17. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", Advances in Cryptology - Proceedings of CRYPTO'84, Springer Verlag Lecture Notes in Computer Science 196, pages 10-18, 1985.
18. MARC4 Modified-Alleged-RC4 from Groove Networks. http://www.groove.net
19. Roger Dingledine, Michael J. Freedman, David Hopwood, David Molnar. A Reputation System to Increase MIX-net Reliability. Proceedings, Information Hiding Workshop, Mar 2001 (LNCS 2137).
20. "A Flexible Security System for Metacomputing Environments" (HPCN Europe 99), April 1999.
21. "Applied Cryptography," B. Schneier. John Wiley and Sons. New York, 1996.
22. "Kerberos: An Authentication Service For Open Networked Systems". J. Steiner, C. Neuman, and J. Schiller. In Proceedings of the Winter 1988 USENIX Conference, February 1988.
23. "Web Services Security (WS-Security) Version 1.0 05 April 2002," B.Atkinson, et al. Available from http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.
24. XML based messaging and protocol specifications SOAP. http://www.w3.org/2000/xp/.
25. "Assertions and Protocol for the OASIS Security Assertion Markup Language," P. Hallam-Baker and E. Maler, eds. Available from http://www.oasis-open.org/ committees/security/docs/ cs-sstc-core-01.pdf.
26. "XML-Signature Syntax and Processing, W3C Recommendation 12 February 2002", M. Bartel, J. Boyer,B Fox, et. al. Available from http://www.w3.org/TR/xmldsig-core/
27. "XML Key Management Specification (XKMS 2.0), W3C Working Draft 18 March 2002", Edited by P Hallam-Baker, Available from http://www.w3.org/TR/xkms2/
28. H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification. IETF RFC 2093. July 97.
29. L. Opyrchal and A. Prakash. "Secure Distribution of Events in Content-Based Publish Subscribe Systems." In Proceedings of the 10th USENIX Security Symposium, pages 281--295, August 2001.
30. A Security Architecture for Computational Grids," I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998
31. "Certificate-based Access Control for Widely Distributed Resources" Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, Proceedings of the Eighth Usenix Security Symposium, Aug. `99.
32. A Transport Framework for Distributed Brokering Systems. Shrideep Pallickara, et al.
33. The JavaTM Cryptography Extension (JCE). Available from http://java.sun.com/products/jce/
34. The IAIK Java Cryptography Extension toolkit. http://jce.iaik.tugraz.at/products/01_jce/documentation/javadoc
35. "AES Proposal: Rijndael", J. Daemen, V. Rijmen, Available at http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf
36. National Institute of Standards and Technology (NIST), "Data Encryption Standard," FIPS PUB 46-2, U.S. Department of Commerce (December 1993)
37. The MD5 Message-Digest Algorithm. R. Rivest. Network Working Group. Internet RFC 1321.
38. The Secure Hash Algorithm (SHA-1) specified in FIPS 180-1. Available from http://www.itl.nist.gov/fipspubs/fip180-1.htm
39. C. Youn, M. Pierce, and G. Fox, "Developing Secure Web Services for Computational Portals". Under review.