Data and Application Security for Distributed Application Hosting Services

Ping Lin & K. Selçuk Candan

Computer Science and Engineering Department

Arizona State University

Tempe, AZ, 85287, USA.

Phone: 480-727-3611, 480-965-2770

Fax: (480) 965-2751

Email: ping.lin@asu.edu, candan@asu.edu

Data and Application Security for Distributed Application Hosting Services

The cost of creating and maintaining software and hardware infrastructures for delivering web services led to a notable trend toward the use of application service providers (ASPs) and, more generally, distributed application hosting services (DAHSs). The emergence of enabling technologies, such as J2EE and .NET, has contributed to the acceleration of this trend. DAHSs rent out Internet presence, computation power, and data storage space to clients with infrastructural needs. Consequently, they are cheap and effective outsourcing solutions for achieving increased service availability and scalability in the face of surges in demand. However, ASPs and DAHSs operate within the complex, multi-tiered, and open Internet environment and, hence, they introduce many security challenges that have to be addressed effectively to convince customers that outsourcing their IT needs is a viable alternative to deploying complex infrastructures locally. In this chapter, we provide an overview of typical security challenges faced by DAHSs,

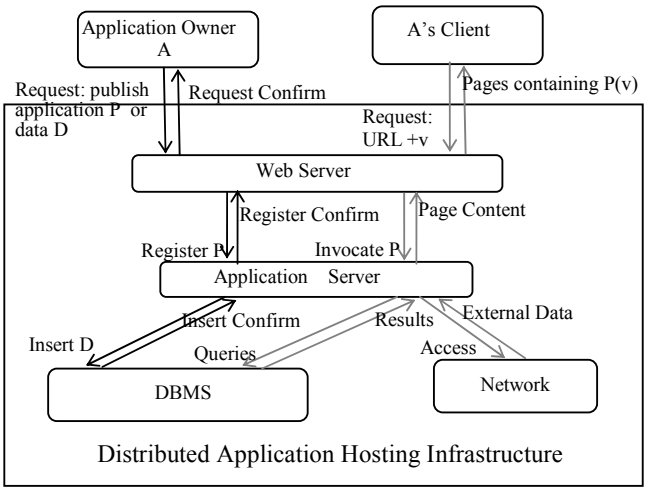
introduce dominant security mechanisms available at the different tiers in the information management hierarchy, and discuss open challenges.

Keywords: Data security, application security, XML, authentication, authorization, confidentiality, computation hiding, private information retrieval.

INTRODUCTION

In an e-business setting, distributed computation with multiple parties' participation is typical. Most business tasks, for example calculating the collective financial health of a group of independent companies, are inherently distributed (Franklin et al., 1992). Consequently, most businesses need an information technology (IT) infrastructure capable of providing such distributed services. For most businesses, investing in a local, privately owned infrastructure is not economically meaningful. For instance, an e-commerce company may find deploying an infrastructure that can handle peak demand volumes, while sitting idle most other times wasteful. Therefore, businesses are willing to pay premium prices for third-party solutions that can help them reduce their infrastructural needs while providing them appropriate quality of service guarantees. Consequently, application service providers (ASPs) and distributed application hosting services (DAHSs), which rent out storage, (Internet) presence, and computation power to clients with IT needs (but without appropriate infrastructures) are becoming popular. Especially with the emergence of enabling technologies, such as J2EE (J2EE, 2003) and .NET (.NET, 2003), there is currently a shift toward services hosted by third parties.

Figure 1. Components of a distributed application infrastructure



Most DAHSs typically deploy a large number of servers to host their customers' business logic and data. They employ hardware- or software-based load balancing components to provide quality of service guarantees to customers. In addition, DAHSs can also place or replicate applications and data in servers closer to the end-users, to eliminate network delays. Examples include Akamai and MirrorImage.

A typical application hosting infrastructure (Figure 1) consists of three major components: database management systems (DBMSs) which maintain business data, application servers (ASs) which encode business logic of the customers, and web servers (WSs) which provide the web interface between end-users and the business applications that are hosted by the DAHS. Although there are various modes of DAHS operation, a common way hosting services are used is as follows:

(1) the customer (or application owner) with an application program publishes this application along with the relevant data onto the servers of the host. (2) Whenever they need, the customers (or its clients) access this application remotely by passing appropriate parameter variables to the host using the web interface. (3) User requests invoke appropriate program scripts in the application server, which in turn issue queries to the underlying DBMS to dynamically generate and construct responses. In other words, the host runs the application with the local or provided data and sends the result back to the requesting party. (4) The host charges the application owner based on the resources (such as bandwidth, CPU, or storage) required for the processing of the request. Figure 1 shows a typical application hosting infrastructure and its operation flows.

Distributed application hosting services, on the other hand, pose various security challenges due to their inherently distributed and mostly open natures. Without proper security provisions, customers will not choose to outsource services, hence DAHS will not survive. In this chapter, we provide an overview of various security challenges faced by DAHSs, discuss the techniques developed to address these challenges, introduce the security technologies and protocols used at different tiers in the Internet information management hierarchy, and discuss still-open issues.

The structure of the chapter is as follows. In the next section, we give an overview of the basic data and application security services and security tools. We

also enumerate the security challenges faced by DAHSs. In section COMPARISON OF PROVISIONS OF VARIOUS ENABLING SYSTEMS, we discuss security mechanisms adopted by widely used information management architectures. In Sections AUTHENTICATION, AUTHORIZATION, CONFIDENTIALITY, we discuss the techniques available to DAHS to tackle these challenges and highlight the open problems.

OVERVIEW OF DATA AND APPLICATION SECURITY

Merriam-Webster dictionary gives the following definition for the term *security*: "the quality or state of being secure, freedom from danger". In data, content, and information delivery systems, security generally refers to the ability of a system to manage, protect, and distribute sensitive information so that information is free from eavesdropping, tampering, and spoofing and the service is available to all legitimate users. Therefore, security plays an **essential** role in e-commerce and e-business applications, where quality of information and service delivery means money, and military systems, where secure information and service links directly to national safety and human life.

Basic data, application, and system security services needed by such systems can be categorized as follows:

- ***Authentication***: All entities in the system should be properly identified.

- ***Authorization and confidentiality***: Access to applications or information should be restricted only to those entitled entities. Data or application code should be unintelligible to any non-entitled entity.

- ***Integrity***: Data or applications should be free from unauthorized modification and damage.

- ***Availability***: Data, application, and the system should be available to users despite attacks or damages.

- ***Auditing***: Records of security relevant events (such as authentication, authorizing decisions, or abnormal events) should be kept for assessing attacks and intrusions or for evaluating effectiveness of security policies and mechanisms.

Although they address different security challenges, at their foundations, all these services rely on basic cryptography techniques. For a comprehensive background in cryptographic protocols and tools refers to part ? of this book. In this paper, we focus on the security challenges peculiar to DAHSs.

Security Challenges and Security Mechanisms in Distributed Application Hosting Services

Distributed application hosting services (DAHSs) face a number of security challenges. In small, closed local area networks, the mutual trust between clients and hosts is high. Clients can fully trust all hosts and the communications are reliable.

However, in open, wide area networks, such as the Internet, where hosts are added and removed dynamically, it is very possible that clients and hosts have little mutual trust. In an environment, where servers may not always be honest, data security constitutes a major concern to users. Executing an application remotely exposes the application code and data to non-trusted, potentially malicious, distributed computing infrastructures. A malicious host may make use of clients' private information to gain illegal profits or cause damages to system by tempering. Certain applications, such as business transactions and military applications, do not lend themselves well to the risks involved in simple outsourcing computation tasks to third parties. How to protect application code and input data from malicious executing environments, therefore, is a critical challenge.

As discussed earlier, techniques for data security occupy a wide spectrum. Most mechanisms aim at protecting data from unauthorized accesses. On the other hand, users' queries to data servers or private inputs to outsourced applications may also be of value and, without proper protection, important information may be leaked to the untrusted or compromised servers. For example, in a stock database, *the type of stock a user is querying* is sensitive information and may need to be kept private, sometimes even from the database server. Hence, traditional network-level encryption schemes may not be enough.

The security concerns in DAHSs can be broadly categorized as follows:

- System resources may be accessed by malicious or illegal clients so that sensitive information is leaked.

- Legal clients may access more system resource than they are entitled to, hence damaging these resources or preventing other clients from accessing these resources.

- Clients' application, data, or requests may be leaked, modified, or lost when they are being transported by insecure communication channel or executed by malicious hosts.

A qualified application hosting infrastructure should provide proper mechanisms to tolerate any faulty or malicious actions listed above. Although these mechanism have already been briefly introduced earlier in this section; in the remainder of the chapter, we focus on the DAHS specific challenges in *authentication*, *authorization*, and *confidentiality*,

- ***Authentication in DAHSs:*** Authentication means verification and validation. Identity authentication enables verifying the identities of the entities participating in the application hosting services (either the clients or the hosts) to make sure that both ends at the communicating channel have the right to perform their tasks. Services will be denied to unauthorized clients.

Data authentication, on the other hand, verifies the origin and the integrity of data. In DAHSs, data owners usually outsource their data and delegate their services to untrusted third-parties. Hence DAHSs should provide mechanisms to enable clients to verify query answers. Application can be delivered across networks for remote execution. This gives rise to two authentication issues: (1) to authenticate that the application is safe and does not contain malicious code; and (2) to authenticate that the application has been correctly executed on untrusted remote sites.

- ***Authorization in DAHSs***: In order to protect resources of DAHS hosts, security policies should be specified by hosts to restrict clients' access to resources. If any violation occurs, proper action will be taken by hosts, including the termination of service.

- ***Confidentiality in DAHSs:*** Private information of DAHS clients (including outsourced application code and data) is not leaked to or modified by any third party when transported through the Internet, nor DAHS hosts when the code is executed or the data is stored. Confidentiality can be achieved by encryption, private information retrieval, computation hiding, and information hiding: noticing that in some cases, users' queries also need to be kept private, private information retrieval prevents query as well as results from being disclosed to the host; computation hiding seeks to hide users'

private input data or code from partially trusted hosts; and information hiding is used to hide not only the content but also the existence of communication from possible attackers.

Traditional security mechanisms like authentication, access control, and cryptography have been well studied and established through various industry standards. On the other hand, some of the required technologies, such as private information retrieval, computation hiding, and information hiding are new research areas and the underlying techniques are not standardized yet. In Sections AUTHENTICATION, AUTHORIZATION, and CONFIDENTIALITY, we will revisit authentication, authorization, and confidentiality challenges in DAHSs and discuss the related technologies in greater detail. Having covered the background in security technologies, however, we now proceed to compare security challenges and provisions of popular data and information management architectures that form the basis of DAHSs.

COMPARISON OF SECURITY PROVISIONS OF VARIOUS ENABLING SYSTEMS

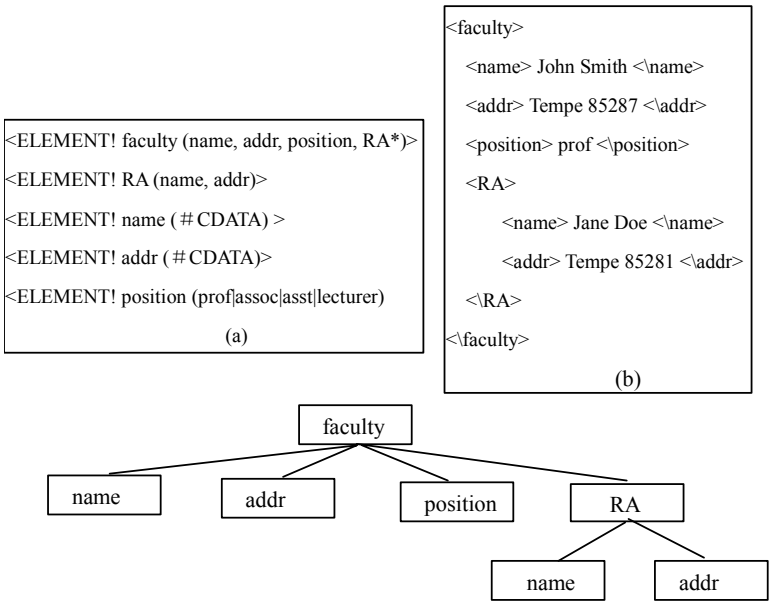
The diversity of distributed application environments and usage scenarios of computer systems contribute to the diversity of the security concerns faced by DAHSs. Since many applications, such as those involved with e-commerce, contain common modules, independent software developers can save a great deal of time by building their applications on top of existing modules that already provide

required functionalities. This calls for a distributed architecture, where different modules can locate each other through directory services and can exchange information through messaging systems. J2EE and .NET are two popular distributed application architectures, and part ? of this book provides a detailed comparison of the security measures these architecture provide. In this section, we discuss security concerns and mechanisms in various enabling software and hardware systems.

Web Services and XML

Web services are standardized ways of integrating web-based applications. The primary function of web services is to allow independent web entities to communicate with each other without considering the underlying IT systems. Web service integration is done through programmatic interfaces which are operating system independent and programming language neutral. Currently, there are various standards that enable web service integration. Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) are open enabling standards. XML is used to organize the data with tags enabling applications to understand the structures of each other's data; SOAP is a light-weight protocol, based on XML, to encode the data for exchange between web applications; WSDL is an XML-based language to describe web services so that web applications can identify services that they need; and UDDI is a directory that lists available services on the Internet and enables applications to discover each other.

Figure 2: (a) An XML DTD, (b) A matching XML document, and its (c) graph representation.



(c)

- ***Hierarchical structure***: XML data has a graph structure explicitly described by user-defined tags. The basic objects of XML data are elements, attributes, and references (idrefs). An element is a semantically whole unit. It may include subelements as components as well as attributes describing its relevant features. An idref links an element to another. Objects of an XML document are connected via element-subelement, element-attribute, element-link relationships and, therefore, form a graph (Figure 2). If references are removed, the graph is degraded into a tree structure; hence the structure of any XML object may be regarded as a hierarchy.

- ***Declarative structure***: XML data may be *validated* or *well formed*. A validated XML data conforms to a structure defined by a given Data Type Definition (DTD). A well-formed XML data follows grammar rules of XML but has no associated DTD file to define its structure. A well-formed XML data may partially conform to some DTD or not conform to any DTD.

XML encryption is the standard process specified by W3C to encrypt any XML web resource (Reagle, 2003). It supports symmetric and asymmetric encryption as well as super-encryption (i.e., encryption of data that has already been encrypted). It can also support block encryption (encryption algorithms, such as DES, that divide plaintext into fixed-length blocks and then encrypt each block respectively) and stream encryption (encryption algorithms, such as SEAL, that manipulates the plaintext in the units of bits). It supports various key exchange protocols, such as RSA-v1.5, RSA-OAEP (Fujisaki et al., 2000), and Diffie-Hellman. With XML encryption, both the schema and the content of XML documents can be hidden from non-entitled entities.

Transport layer security protocols, such as SSL/TSL or network layer security protocols (IPsec) lay the foundation for secure messaging in web services. These protocols enable client/server authentication, data integrity, and confidentiality as discussed before. XML signature (Section AUTHENTICATION) and XML encryption can be used within SOAP to provide message-level authentication and persistent confidentiality. IBM, Microsoft, and VeriSign proposed WS-Security specification to

describe how to attach signatures and encryption headers to SOAP messages (Atkinson, 2002).

At the higher levels, there is a need for service-level security models. For example, IBM provides a general security model for web services in [.NET, 2003]. In this model, a web service is accessed by requiring each requester to carry with the request messages some proof that certain specified claims are satisfied. The associated claims and relevant information constitute the policy for the web service. A security token for an entity encodes security information, such as user name or digital certificate. The requester proves required claims by attaching its security token to request messages or by asking the security token from a special web service called Security Token Service (which may ask for its own claims). IBM and Microsoft are developing WS-Policy to describe capabilities and constraints of the security policies on entities, WS-Trust to describe the trust models for Web services, WS-Privacy to enable Web services and requesters to declare privacy requirements and practices, WS-Secure Conversation to describe message exchange authentication and management, WS-Federation to describe the management of trust among heterogeneous systems, and WS-authorization for describing how to manage authorization data and security policies (.NET, 2003).

Table 1. Oracle and DB2 comparison (sources (Oracle, 2003; DB2, 2003))

	Oracle	DB2
Authentication	Provided	Provided
View	Provided	Provided
Access Control	Row level access control	Table (view) level access control
Encryption	Row level encryption	Table level encryption
Auditing	Fine grained	Less granular

Although most web applications reside in and above the application server tier that use these two technologies, data storage and management is usually left to database management systems (DBMSs). Next, we will consider security provisions of two popular DBMSs.

Database Management Systems (Oracle and DB2)

Views, private virtual databases, access control, and encryption have long been practiced by back-end database systems to achieve security. In Table 1 we compare security mechanisms used by two popular database management systems: Oracle (Oracle, 2003) and DB2 (DB2, 2003).

- Oracle's security mechanisms are embedded into the database. For DB2, there are also various tool suites to help secure DB2 data. Oracle and DB2 both provide authentication. DB2's authentication works closely

with the underlying operation system's security features to verify user IDs and passwords.

- Both of their access controls are role based. Oracle can provide access control explicitly to row level data. In DB2, access control can be explicitly granted to tables or views and row level access control is gained implicitly by defining views on rows.

- Both of them provide database encryption. Through GUI interface provided by the Encryption Wizard of Oracle, encryption can be conducted at schema, table, or column levels. With application tools such as IBM DATA Encryption, encryption for DB2 data can be conducted at table levels.

- A view is a specific way of looking at a database. Generally, a view is constructed by arranging data in some order and making only some parts of the data visible. Views can hence be used as a mechanism to grant various types of access rights on the same data. Oracle and DB2 both provide views and view-based access control.

- Oracle and DB2 both provide auditing to capture relevant data. Oracle provides session auditing at the schema, table, and column levels to trace users' encryption/decryption operations. DB2's auditing facility acts at the instance level, where an instance is a database manager that maintains a set of databases that cannot be accessed by other instances.

While application and database servers are dominant architectural components for e-businesses and web service providers, they also form the basis of *grid*s that integrate scientific as well as business applications and data.

Data and Computation Grids

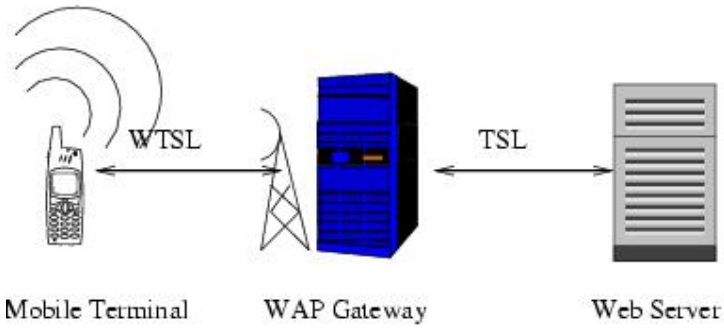
Grid computing refers to the system design approach which benefits from resources available from various devices in a network to solve a complicated problem that usually needs a huge number of processing cycles or a large amount of data from different origins. Grid computing makes all heterogeneous systems across an enterprise or organization virtually shared and always available to any legal members. Consequently, through sharing of computing capabilities and data, grid computing accelerates processing of problems that are too complicated for a single machine to handle. Hence, it enables complex business problems, computing intensive scientific problems, or large data analysis problems to be solved rapidly. The need to share resources and to execute untrusted code from any member of the grid introduces various security concerns for grid computing. In grid systems, authentication and authorization should always be present. Data grids should integrate a variety of databases regardless of which operating system they reside on. Although each database may have its own security requirements and provisions, the data grid should provide access control that respects each data source's policy while satisfies users' data needs to the greatest possible extent. Although, there are no established standards, Butt et al. (2002) proposed a technique, which uses run time monitoring and restricted shells, to enable maximum legitimate use permitted by

security policies of a shared resource. The application architectures, DBMSs, and resources on a grid are connected with each other as well as the end users through wired or wireless networks. Hence, network security is essential.

Wired and Wireless Networks

Firewalls and secure socket layer (SSL) communication are two typically used techniques to achieve network security. Firewalls isolate to-be-protected servers from the open Internet so that only messages from authenticated sources can penetrate through. Firewall authentication relies on package filtering or stateful package inspection to check the originating and destination address associated with messages. The main deficiency of the firewall technique is that it does not inspect the content of the messages. An attacker who achieves access by misrepresenting his/her identity may exploit this security hole by sending malicious content. SSL is an open protocol, developed by Netscape, to provide secure HTTP connection and data transmission between web browsers and web servers. Theoretically, it is a protocol at the transport layer of network protocol stack. Its function is to establish secure communication sessions between clients and servers. Before data communication starts, by a handshake protocol, SSL allows a client and a server to authenticate each other through asymmetric cryptography and X.509 certificates, as well as to negotiate the encryption algorithm and cryptographic keys to be used during secure data communication. From that moment on, all data is encrypted by symmetric cryptography. To check data integrity, data is transported along with keyed MAC checksum. The SSL version often used in wired networks is TLS.

Figure 3: TSL and WTSL



In wireless networks, the counterpart of TSL is WTSL. A mobile device can establish secure communication session to a wireless access protocol (WAP) gateway through WTSL. Then, on behalf of the mobile device, this WAP gateway can establish secure communication session to the target server through TSL over wired networks (Figure 3). Since WTSL is not compatible with TSL, the WAP gateway has to do translation between them. Therefore, data is encrypted all the way between the mobile client and the server except on the WAP gateway where the translation occurs. If the WAP gateway is compromised, confidential data may be leaked. Two possible ways to achieve end-to-end security are to eliminate the WAP gateway or to add application level encryption. Table 2 gives a comparison of TSL and WTSL.

Table 2. Comparison of TSL and WTSL (source (Wright, 2000)).

	TSL	WTSL
Usage Environment	Wired networks	Wireless networks (WAP)
Protocols	TCP	TCP or UDP
Support for session suspend and resume?	YES	YES
Compatible with SSL	YES	NO

- TSL is a *de facto* standard to establish secure session between clients and servers in the Internet. WTSL is its counterpart in wireless networks (WAP). Both of them are transport layer protocols. They both support encryption, public key infrastructure, digital signature, certificate etc.

- TSL relies on reliable network connection (TCP). WTSL can be established on unreliable network connection (UDP).

- To adapt for mobile environments where connection is not stable and may be lost easily, WTSL supports suspended or resumable sessions. Support for resumable session is also an option for TSL.

- TSL is derived from SSL. WTSL, on the other hand, is not compatible with SSL.

Having covered the security challenges and provisions in enabling architectures, in the remainder of the chapter, we will focus on authentication, authorization, and confidentiality related challenges and provide an overview of the solutions proposed and techniques developed to address these issues. We will review the state-of-the-art as well as state-of-the-practice, highlight open challenges and directions, and discuss impact of data and code security on various applications.

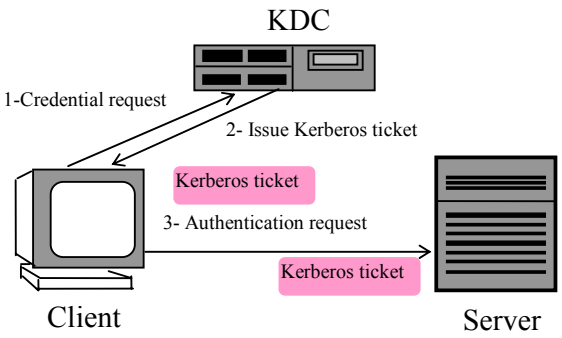
AUTHENTICATION

Authentication involves verification of the claims made by parties in a secure environment. A common authentication task is **(a)** the verification of the identity of a communicating entity: without knowing for certain the identity of a client, for example, it is not possible to decide whether to accept or deny the access request. In addition, **(b)** data exchanged within a DAHS also needs to be authenticated to verify that it is really from the claimed origin and that its content is really what the source claims to have sent. In such an untrusted environment, answers to database queries should also be verified. Finally, **(c)** application authentication involves verifying whether execution of the application on untrusted DAHS servers is correctly performed or not. In this section, we will consider various authentication tasks in DAHSs.

Identity Authentication

Identity authentication protocols may be classified into two: (1) trusted third party authentication and (2) authentication without a trusted third party.

Figure 4: Kerberos authentication



Trusted Third Party Authentication

Trusted third party authentication relies on a third party that is trusted by both communicating parties. Kerberos protocol (Kohl & Neuman, 1993), which is the *de facto* standard for network authentication, is a typical example. Kerberos relies on symmetric cryptography. In this protocol, the trusted third party is called the Key Distribution Center (KDC). Authentication protocol consists of three steps (as shown in Figure 4):

i) The client sends a request, which includes its and server's identities (c and s respectively), together with a nounce t, to the KDC. The nounce is used to prevent replay of the request and should be a value, such as a timestamp, that cannot be changed.

ii) KDC creates a session key $K_{c,s}$, and sends the session key and the nounce that are encrypted with the client's secret key K_c, back to the client. The KDC also issues credentials with which the client can access the server. A

credential is a ticket, $T_{c,s} = < c,\ K_{c,s},\ \exp iration\_time >$, that can be used to identify the client at the server. The ticket is encrypted with the server's secret key K_s, to prevent the client from tempering with it.

iii) Finally, the client sends to the server an authenticator which includes

(1) the current timestamp t_c, encrypted using the session key, and

(2) The ticket $T_{c,s}$ which was encrypted by the KDC with the server's secret key.

The server, after receiving the authenticator in Step 3, can establish the identity of the client by (1) decrypting the ticket, (2) extracting the identity of the client and the session key, then (3) using the session key to decrypt the authenticator to see if the timestamp is current. If so, under the assumption that the KDC is trustworthy, the request is known to come from the stated client. If the identity of the server is also required to be authenticated to the client, (4) the server will respond with an incremented timestamp encrypted with the session key. This will enable the client to know that the server is able to read the timestamp, which means that the server has access to the session key, thus it is indeed the target server. After the authentication is over, the client may confidently communicate with the server using the session key.

Note that, in the above protocol, each time a client communicates with a server, trust between the client and the server has to be established using server's and client's secret keys. To reduce the probability of disclosure of the secret keys,

the Kerberos protocol can be enhanced by allowing a Ticket Granting Server (TGS). In the enhanced protocol KDC has access to client's and TGS's secret keys, whereas the server's secret key is only known to the TGS.

One major advantage of the Kerberos protocol is that it only involves efficient symmetric encryption. On the other hand, it relies on the absolute security of KDC. If KDC is corrupted, the security system will be compromised. In order to prevent a corrupt third party to break the security of the system, other protocols aim to eliminate the need for a trusted third party.

Authentication without a Trusted Third Party

Public key cryptography can serve as an authentication protocol (Nace & Zmuda, 1997). Let us assume that there are two communicating parties, S_a and S_b. Each party has a key pair (Pub, $\Pr iv$) which includes a public key and a private key, respectively. Let us denote S_a's key pair as (Pub_a, $\Pr iv_a$) and S_b's key pair as (Pub_b, $\Pr iv_b$). The authentication procedure is as follows:

i) S_a and S_b exchange their public keys.

ii) S_a generates a random number R_a, sends it to S_b.

iii) S_b responds with $\Pr iv_b(R_a)$, and another random number R_b.

iv) S_a decrypts $\Pr iv_b(R_a)$ with Pub_b. If she obtains R_a, she knows the other party is S_b, for only S_b can sign it with $\Pr iv_b$.

v) S_a responds with $\Pr iv_a(R_b)$.

vi) S_b decrypts $\Pr iv_a(R_b)$ with Pub_a. If he obtains R_b, he knows the other party should be S_a, for only S_a can sign the number with her private key.

After the trust has been established, S_a and S_b can communicate with each other in this way: before sending a message, each party encrypts the message with the other party's public key. The other party, after receiving the encrypted text, decrypts it with his/her own private key to retrieve the plain text.

The main advantage of the public cryptography authentication is that its security depends only on the two communication parties, themselves. One main disadvantage of public cryptography authentication, however, is that it utilizes the inefficient asymmetric cryptography. Also, if a malicious third party intercepts the public keys being exchanged, it can replace them with different public keys and pose as one of the communication parties. A key exchange protocol, like Diffie-Hellman, may serve as a solution to this problem.

Data Authentication

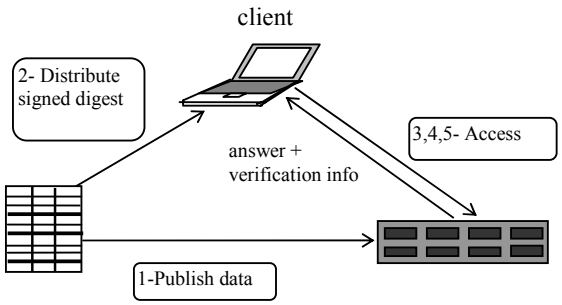
Data authentication involves verifying data's origin and integrity. Digital signatures can be used to prove the origin of a data message and hash (or digest) values can be used to check the integrity of the data being exchanged. In fact, by signing on the checksum hash value, both the origin and integrity can be verified. Basic tools for data authentication, therefore, include signature algorithms (such as

DSA and RSA/SHA-1) and digest algorithms (such as MD5, MAC, and SHA. However, different types of data have different structures or usage contexts; hence the ways to digest or sign them may vary.

In DAHSs, data owners make their database available at third party servers. Since a single database contains more than one data object and since accesses to the database are through declarative queries (instead of explicit object ids), authenticating database accesses require techniques more elaborate than simple digests and signatures.

A correct database answer to a given query should be *complete* and *inclusive*. A complete answer must include all data elements that satisfy the query and an inclusive answer should not include any data that does not satisfy the query. If the server hosting the database is trusted, then one possible authentication solution is to let the server certify answers by signing on them using a private key. However, in a DAHS, data owners may outsource their databases to untrusted third party publishers; hence, new protocols that authenticate database query results from untrusted publishers are needed. Some of these techniques are discussed next.

Figure 5: Third party data publishing

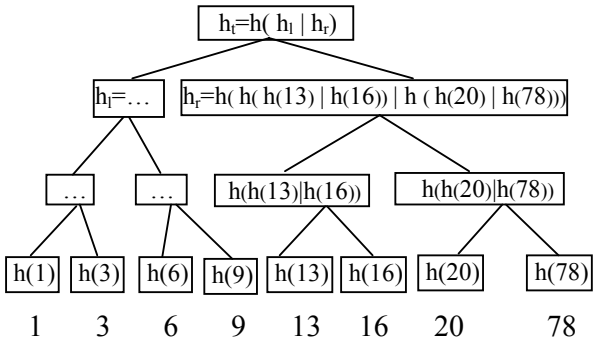


Authentic Database Publication

Devanbu et al. (1999) propose a *generic* model for authentic third party data/database publication (Figure 5). The model consists of the following steps: (1) the data owner sends the database to the third party publisher; (2) the data owner signs the database digest and sends it to its clients; (3) a client queries the database stored at the publisher; (4) the publisher processes the query, sends the answer and some verification information to the client; and (5) using the verification information and the digest, the client verifies whether the answer it received is correct or not.

Query results can always be verified by submitting the whole database as the verification information to the client. Clearly, this would be very expensive. Hence it is crucial to develop proper database digest techniques that enable exchange of minimal verification information.

Figure 6: A balanced binary Merkle hash tree



Devanbu et al. (1999) show that Merkle Hash Trees can be used to efficiently verify answers to selection, projection, join, and set operation queries that are common in relational databases. This protocol relies on the existence of database index trees, which are used for providing efficient access to the contents of the database. A trusted party (e.g., the data owner) recursively digests nodes of the index tree such that every leaf digest is a hash over the corresponding data value and every non-leaf digest is a hash over its children's digests (Figure 6). Merkle hash trees have two properties that enable authentication: Given the correct root digest,

- any modification to the tree structure can be detected; and
- the existence of a given subtree can be verified.

These two properties are fundamental for the verification of the *inclusiveness* of query answers. By requiring leaves of the tree being sorted according to some total order, it possible to enhance the Merkle hash tree with a

third property: Given the correct root digest and a sequence of leaf values, $q = <t_i ..., t_j>$,

- the completeness of the sequence can be verified.

This enhanced property is fundamental in verifying the *completeness* of query answers. Based on these results, Merkle hash trees can be used for authenticating inclusiveness and completeness of relational query results.

As discussed in Subsection Web service and XML, on the other hand, in DAHSs and the web, the *de facto* standard to organize data is XML. Hence, next we look into mechanisms for authenticating data and databases published in XML format. First, we concentrate on signing XML data and documents and then we will discuss authentication procedures for third party XML database publication.

XML Data Signatures

World Wide Web Consortium (W3C), which develops interoperable technologies and standards for the web, has established an XML signature standard (Eastlake, 2003). This standard includes

- a digest algorithm (SHA-1),
- a signature algorithm (DSA or RSA/SHA-1),
- a message authentication code (a hash function with a shared secret key), and

- transform algorithms to be applied to XML documents before they are digested. Transform algorithms add flexibility to the XML signature. For example, with path filtering transformations, the signer can choose to sign only nodes on a specified path in a given XML document.

W3C also specifies a progressive digesting procedure called DOMHASH (Maruyama et al., 2000) to recursively digest a DOM tree (Hégaret et al., 2003) (the native tree presentation of an XML document) from the leaf to the root, so that each node (element or attribute) has a digest value and the digest of an non-leaf element is a hash over its subelements and attributes' digests and its content. This strong version of the digest enables efficient comparison of two versions of an XML document to find the different parts.

XML signatures can be used to verify the integrity of a given XML document or some selected parts of it, but it cannot be used to verify the inclusiveness and completeness of answers to XML queries. Next, we discuss techniques for authentication of results to a query executed on an XML database published at an untrusted third party.

Authentic Third-Party XML Database Publication

Merkle hash trees work well for node selection queries (as discussed earlier), however, it not directly applicable for XML path queries, which require identification of all paths in a given XML document that match a given condition.

Devanbu et al. (2001) proposes the following approach for creating XML digests using the DTD of a given XML document:

i) The XML document owner builds an enhanced Merkle hash tree for each path type in the DTD and associates the root digest of the resulting Merkle hash tree with the corresponding path type.

ii) The owner builds another enhanced Merkle hash tree from all path type digests and associates the root digest of this second tree with the given XML document.

iii) The owner then signs the document digest and sends it to clients for verifying query results.

This enables efficient verification of the results to simple path queries in addition to the selection queries. For each path query, the publisher finds all path types that match the query and for each matching path type, it constructs a certificate. Using the XML digest provided by the owner, the client can verify whether the certificate provided by the publisher includes all and only the results of the correct path type. Furthermore, the client can pre-calculate all path types that match the query and see whether for each matching path type, a certificate is received. Hence the client can verify the completeness and inclusiveness of the answer. This protocol, however, has the following limitations:

- It is computationally costly. Besides the cost of building Merkle hash trees, the owner needs to pre-calculate all subtrees for each path type. To verify each query, the client needs to find all matching path types.

- It has a large space overhead for the publisher: for each possible path type, as a large certificate has to be maintained.

- It requires clients to know the DTD of the document to verify answers.

- It can not handle complicated path queries and queries over XML data that do not have a corresponding DTD.

Bertino et al. (2002) propose an alternative protocol that is cheaper and that does not need DTDs. This protocol utilizes DOMHASH to calculate root digest. See part ? of the book for a detailed discussion. Since this protocol is based on only one DOMHASH value for the entire XML document, it is cheaper than the previous protocol which needs hashing for every path type. Also, since the path types do not need to be known in advance, this protocol does not need the DTDs. One disadvantage of the protocol, however, is that it lacks the ability to verify answers to selection queries, hence it is less flexible.

Application Authentication

In distributed environments, application code can move among various distributed entities: (1) application code (such as Java applets) can be distributed from servers to clients for local execution; (2) application code can travel from thin

mobile clients to powerful servers for execution; (3) in DAHSs, application code can be published and outsourced to remote server by the application owners; and (4) the code can travel between DAHS servers to achieve load balancing and process migration. For application code distribution, the recipient (either the client or the server) must validate the origin and the integrity of the code before loading, installing, and executing it. Otherwise, the recipient can be subject to a malicious or tampered source which can gain unauthorized access to the recipient's resources or can receive a virus which can break down the recipient's machine and spy for sensitive information. The source also should use authentication techniques; otherwise, a malicious recipient may try to deceit the owner by providing false results. Furthermore, if the application code and the associated data visit multiples servers to conduct steps of a computation, a malicious server can modify the state of the code or the data it carries, before the code moves to the next server. For example, a malicious airline server may raise the lowest airfare a mobile airfare agent code has computed from all priory visited airlines to cheat the client.

As in data authentication, checksums and digital signatures once again constitute the set of basic tools for application code authentication. Prior to transmission, the code owner can sign the code with its digital certificates. Upon receipt, the recipient can verify the signature and decide whether the origin and integrity of the code can be trusted. If the code is verified, the recipient then has to determine the execution permissions for the code. Although checksums and digital signatures can be used to identify the owner and recipient of the code, if these

entities are themselves not trusted, we need additional mechanisms to verify the application code and the execution results. To prove that an application does not contain any malicious code as to damage internal data structure of the host, or overuse resources of the host, the application owner can provide an encoding of a proof that the code complies with the security policy of the recipient. Application code augmented as is called a *proof-carrying code*. To prove that the application is executed properly, on the other hand, the owner can benefit from *replicated execution* or *execution traces*. In a multi-agent system where there are multiple agents interacting, in order to prove that an agents is secure for interactions (i.e., the agent will not access services or data that it is not entitled to), the service provider can utilize agent's current state, records of previous interactions with other agents, and the analysis of possible consequences of the interaction. Next, we discuss proof carrying codes, replicated execution approaches, execution traces, and secure agent interactions individually.

Proof-Carrying Codes

Proof-carrying code protocol (Necula & Lee, 1998), for proving that an application does not contain any malicious code, consists of two phases:

i) The recipient of the code extracts from the untrusted application code *safety predicates* that can be proved if and only if the code conforms to the security policy defined by the recipient. Generally, a security policy defines (1) the language in which the application should be written, (2) the

conditions under which the application can be trusted, (3) the interface between the recipient and the application code, and (4) the methods for inspecting application code and discovering potential security violations. The safety predicate is constructed by inspecting the instructions of the application to find ones that may violate the security policy and generating for each such instruction a predicate that proves the safe execution of the instruction. The recipient sends the safety predicates to a *proof producer* (such as the application owner) and the proof producer returns the proof back to the recipient.

ii) The recipient, then, checks the proof via a proof checker. If the recipient verifies the correctness of the proof, it can safely install and execute the application.

This protocol is general and the recipient does not have to trust any party (either the owner or the proof producer). However, the proof size can be very large: it usually grows linearly with the size of the application code, but it can grow exponentially in worst cases.

Replication

One way to ensure the correct execution of the application code is via server (or recipient) replication and voting (Minsky et al., 1996). In this protocol, the execution of the code is divided into stages and each stage is executed at multiple servers:

i) The owner dispatches the stage one execution to multiple server replicas. Every replica sends the stage one output to all replicas of stage two.

ii) In the following stages, each replica chooses its input to be the majority of the outputs received from the previous stage. It then conducts its computation and sends the output to all replicas of the next stage. At the last stage, the replicas send their outputs back to the client.

iii) Finally, the client determines the output to be the majority of the outputs it received from the replicas corresponding to the last stage of the execution.

In addition to the high network and computation bandwidth requirements, this protocol fails if the majority of the servers at a certain stage is malicious or compromised. To prevent a third-party server from spoofing as one of the replicas, this protocol can be extended with authentication schemes to verify each replica.

Yee (1997) proposes a code replication scheme in which a copy of the code is executed by visiting the sequence of servers in the reverse order of stages. This scheme is capable of detecting certain types of tampering with the results. For example, given a mobile agent code that searches the lowest airfare by visiting airline servers in a particular order, one copy of the agent can travel the same servers in the reverse order. In this case, any inconsistency between two results implies tampering. This protocol is simple and introduces low overhead (only a copy of the original execution required). However, it is effective only when the order in which the servers are visited does not make any difference in the final result.

Cryptographic Traces

Verifying the execution trace of an application on a given server is another way to authenticate results (Vigna, 1998). An execution trace for an application on a given server records the statements executed by the server and the values of the data obtained. The traces are protected via the server's certified digital signature; hence the server cannot disown the trace. By retrieving the traces from suspected servers and using them to simulate the overall execution, the client can identify any tampering.

One main disadvantage of this protocol is that it is a post-execution method and cannot offer timely authentication. Another disadvantage is that it does not provide any mechanism to detect tampering a *priori*, so the client does not have any indication regarding when to ask for traces from a server.

Secure Agent Interactions

Bonatti et al. (2003) present a framework intended for multi-agent environments with authenticated the secure interaction between agents. The system keeps track of all agents' (1) histories, actions they performed, and messages they exchanged; (2) states which contain their current knowledge about the system; and (3) consequence operations, which describe what knowledge an agent can derive from its state and its reasoning capability. A secret is specified by the service provider agent in terms of actions and data forbidden to be accessed. Based on these, secure interaction in a multi-agent system is defined in terms of secure histories that

do not leak any information via messages that are exchanged and consequences that do not violate any secrets. Maintaining correct and accurate information about an agent's state and consequence is almost impossible in practice. Hence, Bonatti et al. (2003) suggest using approximate information and proposes a rule-based logic language with which an agent can specify how to approximate the available information about other agents.

Authorization

Even when identities of the parties in a DAHS environment have been verified through authentication, there is still possibility of other forms of attacks by malicious entities. For instance, available resources should be protected and the access should be restricted, otherwise untrusted users may break down the system purposefully or even trusted users may, without any malicious intention, cause damage to the system by improper operations. Therefore, proper access control mechanisms that can ensure that the operations that an authenticated user (or an application program on behalf of the user) can invoke on a server lie within the limits of server's security policies are essential.

Security Policies

Security policies specify what actions are allowed and what are not allowed. A policy has three dimensions: subjects, objects, and access types. Subjects are users or programs that work on be half of the users. Objects represent resources to be protected from subjects. Access types include actions, such as read or update that

subjects can execute on objects. There are various security policy models (Sandhu & Samarati, 1994). In this section, we briefly discuss discretionary, mandatory, and role based policies.

Discretionary Policies

A set of authorization rules defines the access mode for each subject and object pair. Every access is checked against this set of authorization rules. This model is simple and good for cooperative but independent environments; however, it cannot restrict what subjects will do with the information after they fetch it from the servers.

Mandatory Policies

In this model, each subject and object is assigned a security level. For example, the security level of an object may reflect the sensitivity of the information associated with the object to unauthorized disclosure. The security level of a subject is called clearance and it may reflect the trustworthiness of the subject. Access is granted only if the security levels of the target object and the subject satisfy certain relationship. Generally, to read an object, the clearance level of the subject should be higher than or equal to the security level of the object; whereas, to write an object, the object being written must have equal or higher security level than the subject (Sandhu & Samarati, 1994). In this way, the information flow is guided in a way to prevent sensitive information flowing to lower level objects. This model fits well with stricter, such as military, environments.

Role Based Policies

This model mediates subjects' access to objects according to subjects' activities (or roles). The model identifies all roles in the system and, with each role, it associates a set of operations and responsibilities. Access authorizations for objects are defined in terms of roles. Subjects are given permissions to play roles. A subject playing one role is granted all operations that are authorized for that particular role. Role based model is flexible. It does not assign access rights to subjects directly, but indirectly through roles. Hence, it avoids the cumbersome task of assigning and re-assigning access rights as the system evolves. This model is also space saving, as redundant specification of access rights assigned to users playing the same role is diminished.

Data Authorization

In a DAHS environment, the sensitive information contained in each data source, managed on the behalf of the data owners, must be protected from unauthorized user accesses. Especially in a data grid, the underlying system should provide authorization mechanisms for the databases that are being federated. Considering XML's role in distributed data and information exchange, special attention should be given to how the structure of XML data affects authorization.

Authorizing Federated and Mediated Databases

DAHSs and data grids usually host applications that integrate heterogeneous packages (including application software and associated databases) outsourced from origins with different security policies.

Therefore, they need to enforce global security policies while respecting the local security policies of each individual data source. Different security models have been developed for federated and mediated databases. Idris et al. (1994) introduce a security model for federated systems where security levels may continuously change. Jonscher and Dittrich (1994) propose a decentralized authorization security model for tightly controlled federated databases. Blaustein et al. (1995) propose a security model that relies on bilateral agreements between data sources to identify how each party protects others' data. Wiederhold (1992, 1993) introduces a centralized model, based on mediators, which integrate heterogeneous databases with different security policies. Candan et al. (1996) introduce two cooperation principles that may be implemented by a mediated system:

- ***Cautious cooperation***: If a user's query only needs to access information that the user is entitled to, the mediator will answer the query unless the global security policy directly forbids so, while each participating database's security policy is always respected.

- ***Conservative cautious cooperation***: If a user's query only needs to access information that the user is entitled to, the mediator will answer

the query unless from such query the user can *infer* information he or she is not allowed to access by global security policies, while each participating database's security policy is always respected.

Based on a rule-based mediator model that consists of a mediator M, a set of data sources $\{d_1, d_2, ..., d_n\}$ integrated by M, a global security policy G, and a set, V, of local security policies, Candan et al. (1996) further propose a formal approach for secure mediated databases. Global security constraints in G are modeled as

- a set of facts of the form $\text{sec}ret(A, i)$, denoting users with security level i has no access right to the information (atom) A and
- a set of rules of the form $\text{sec}ret(A, i) \leftarrow \text{sec}ret(A, j) \wedge i < j$, enforcing that if A can not be accessed by certain security level, it can not be accessed by lower levels either.

Local security constraints, V, on the other hand, are modeled as

- boolean functions of the form $viol_d(d : f(<\text{arg}uments>), i)$, which identify whether executing function f with specified arguments on date source d for users of security level i violates d's local security policy or not.

In order to prevent a malicious user from inferring unauthorized information through knowledge about the implemented security policy, Candan et al. (1996)

adopt *query transformation* methods to ensure that the query simply fails (without raising violation alerts) if it violates any local security constraints.

Participating databases may have different security orderings; for example, the classification labels of security levels in different databases may be different or security levels with the same label may have different security orderings in different databases. To integrate such heterogeneous databases, there is a need for mechanisms to merge heterogeneous security orderings in a way that each individual security ordering is preserved and the constraints between security orderings of different databases are satisfied, while a maximal level of global security is maintained when there are conflicts. Bonatti et al. (1996) give a formal definition of this problem and proposes two solutions: rule-based and graph-based approaches. The rule-based approach represents the semantics of security orderings and inter-database constraints using logic rules, while the graph-based approach represents them using a graph. Both methods can find a combined security ordering, for a given non-conflicting set of individual orderings, in polynomial time.

Authorization of XML Data

As discussed in Section Security Challenges and Security Mechanisms in Distributed Application Hosting Services, XML security is an essential part of web-based information architectures. Therefore, developing access control mechanisms that understands the structure and properties of data in XML form to enforce selective dissemination of information over the web is essential for DAHSs.

According to (Bertino et al., 2001), an XML access control mechanism should at least

- consider XML's rich, hierarchical structure and provide fine-grained authorization to components of a given XML data;
- provide both DTD-level and data-level authorization. DTD-level authorization applies to a set of data objects conforming to the same DTD, while data-level authorization applies to one particular document or its components;
- handle authorization to XML objects that are not conforming or partially conforming to a particular DTD;
- devise proper authorization propagation rules that refer to hierarchical relationships (such as DTD-data, element-subelement, element-attribute, and element-link) to propagate authorization policies of higher level components to lower level components. These rules should also provide mechanisms to solve propagation conflicts when a component has multiple inconsistent authorizations propagated from higher levels.

Due the rich structure of XML, a standard authorization mechanism for XML data remains an open challenge. Author-X (Bertino et al., 2001) is a tool that provides access control to XML data. Author-X satisfies the minimal requirements mentioned above. It adopts the discretionary access control model; the policies have

the following format: $< U, O, P, R, S >$. U denotes a user or a group of users to whom the authorization applies. O describes the object (DTD, XML data, or portions of them) to be protected. P denotes the access privilege (browsing or authoring) to be permitted or restricted. R provides the propagation rules (*cascade* the authorization to all descendants, limit the propagation to *first-level* descendants, or *no-prop*ogation). Finally, S , denotes whether this is positive or negative authorization. Using negative authentication, a security manager can efficiently define authentications with exceptions (e.g., defining an authorization applying to a whole document except for some few elements).

Author-X defines the following conflict-resolution rules: (1) explicit authorizations override propagated ones; (2) if there are multiple propagated authorizations, the most specific one (lowest level in the hierarchy overrides the others; (3) if there are conflicts due to propagated rules at the same level, the negative authorizations override.

The process of authorization in Author-X is as follows: For the target XML data object, Author-X

- finds the associated DTD. If the XML document does not have an associated DTD, Author-X finds the DTD that the target document mostly conforms to (hence it handles the partially conforming documents);

- propagates all possible DTD-level and document-level authorizations and resolves all conflicts;
- prunes from the document all elements that do not have required positive authorizations (explicit or implicit); and
- evaluates the user query against the pruned document and extracts the target data.

The IBM alphaWorks XML Security Suite (XML Security Suite, 1999) is another tool that provides access control mechanism for XML documents. It shares some common features with Author-X: authorization propagation based on structure hierarchy and conflicting authorization resolution, implementation of an XML-based language (XACL) for specifying security policies, and fine grained authorization. On the other hand, unlike Author-X, IBM alphaWorks utilizes role-based access control, which is more suitable for e-commerce environments. It also accommodates context and provisional actions into the access control model. Hence the extended authorization policy can specify whether a given subject, under certain context (access request time or some other conditions), is allowed to access the given protection object in a given way or not. It can also specify provisional actions (such as logging the access decisions, encrypting specified elements, verifying signatures, reading, writing, or deleting specified elements) that have to be performed whether the access is granted or not. With this extended semantics, XML Security Suite integrates authorization and non-repudiation mechanisms for

accessing XML documents and data objects; i.e., a subject cannot deny that it made an access attempt. IBM alphaWorks has two propagation directions, up and down, and when there are conflicting propagations, it arbitrarily selects one. This differs from the *most specific* based conflict resolution of Author-X.

Cho et al. (2002) propose a simple mandatory XML data security model, in which XML elements may have associated security levels or inherit them from their parents. A user is allowed to access only those elements whose security levels are no more than her clearance level. To minimize the cost of checking security levels, for each query, the system rewrites a given XML query by identifying the appropriate amount of checking. Cho et al. (2002) achieve an optimal rewriting of the query. In general, checking whether or not a user has access right to a particular object from a given set of access control rules can be inefficient. Maintaining an explicit accessibility map that lists all users that can access a given object, on the other hand, is space-inefficient. Yu et al. (2002) introduce compressed accessibility maps to efficiently enforce authorization policies over XML data. The compression is achieved by taking advantages of the feature that XML data items grouped together have similar accessibility properties.

Open networks like the Internet are inherently insecure despite authentication and authorization schemes. Sensitive data or application code may be leaked to or compromised by attackers eavesdropping on the communication link

between clients and hosts or disclosed to malicious hosts. The business logic or query results may be altered to cheat clients.

In the Section CONFIDENTIALITY, we discuss underlying confidentiality issues and related technologies.

Application Authorization

Karnik (1998) summarizes several ways to authorize accesses to server resources by outsourced application programs in mobile agent-based systems. Similar approaches can also be implemented for application authorization in DAHSs:

- ***Direct reference approach***: The server supplies the mobile agent with reference to the resource and screens all accesses via a security manager. The security manager checks against the security policies to see if each application method (or procedure) under execution is allowed to access the associated resources.

- ***Proxy approach***: The server builds a specific *resource proxy* when a mobile agent asks for some resource. The proxy provides a safe interface to the resource. This safe interface looks the same as the original interface, but certain methods are disabled to prevent the agent from accessing the resource via methods that the security policy does not permit.

- ***Capabilities approach***: This mechanism has been adopted in several distributed systems. Every agent, before accessing a resource, presents a credential containing its access rights to the server. Only after the server grants its approval can the agent access the resource. The credential is issued to the agent after its identity has been authenticated.

- ***Wrapper approach***: The resource is encapsulated with a wrapper and agents have references only to this wrapper. The wrapper maintains access control lists and decides whether an agent has the authority to access the resource or not.

- ***Protection domain approach***: There are two execution environments: one safe environment to host the agents and one trusted environment to provide access to the resource. The safe environment processes each potentially unsafe request of an agent according to its own security policy and screens unsafe requests. For safe requests, it calls methods of the trusted environment that provides access to the resource. The trusted environment can only be called by methods within this safe environment.

The *proxy* and *capabilities* approaches are flexible: an instance of a proxy or a capability can be generated dynamically and specifically to satisfy each application or agent code. The dynamicity of the *capabilities* approach, on the other hand, introduces various challenges: an application may propagate its capability to others or, if security policies change, capabilities may need to be revoked. The

wrapper approach is simple and more static: there is only one wrapper for each resource object and all applications share the same wrapper for that resource.

CONFIDENTIALITY

Confidentiality (or privacy) means that private information of DAHS customers (including outsourced application code and data) is not leaked to or modified by any party. The degree of confidentiality of security schemes can be categorized into the following two classes:

- ***Information theoretic privacy***: These schemes are built *without* any cryptographic assumptions and, hence, cannot be broken even with unlimited computation power.

- ***Computational privacy***: These schemes are built on various cryptographic assumptions, usually about certain hard to compute problems. In these schemes, the goal is to ensure that there are no efficient computations (conducted by a randomized algorithm bounded by a polynomial time in the length of inputs) that can break the scheme. Cryptographic assumptions used by computational privacy schemes are based on one-way functions that are easy to compute but hard to inverse. The most popular one-way functions, such as integer factorization, Φ-assumption, and quadratic residuosity problem, come from number theory (Goldreich, 1997).

Encryption schemes are the basic tools to achieve confidentiality. Besides the common data encryption methods that hide sensitive data, there have been some techniques for hiding other kinds of secret information from untrusted servers:

- ***Information hiding*** hides not only the communication content, but also the existence of communication between two parties, so that no suspicion arises that there is a secret communication exists.

- ***Computation hiding*** prevents host sites from gaining unauthorized information about the content of the published code, the data it executes on, or the outputs produced as a result of the computation.

- ***Private information retrieval*** aims to let users query a database without leaking to the database what data is queried.

In this section, we report on the state of the art techniques for hiding applications, data, data distribution, and user's query from application and database servers. In Subsection Computation Hiding, we focus on how to keep data or application confidential from untrusted hosts and we describe solutions to the problem of computing encrypted functions with encrypted data. In Subsection Private Information Retrieval, we give a general introduction to private information retrieval and in Subsection Private Informational Retrieval in XML Databases, we specifically focus on private information retrieval in XML databases.

Information Hiding

Cryptography deals with concealing the content of information. Information hiding, on the other hand, seeks to hide the existence of information. For some highly sensitive applications, such as military or intelligence agencies, even the existence of communication arises suspicion. Hence to provide information hiding services, a DAHS should be able to hide the communication traffic itself, which cannot be achieved with ordinary cryptography.

Information hiding technologies include *spread spectrum radio*, which is used to hide wireless transmission; *temporary mobile subscribers*, which are used in digital phones to hide a given user's location; and *digital watermarks,* which are used to imperceptibly insert copyright information in digital images, video, and audio. As they are not heavily used in DAHSs and data grids, in this chapter, we will not discuss information hiding techniques in detail.

Computation Hiding

In distributed computation environments, such as DAHSs, input data owners, application owners, and computation providers (servers) may be different parties distributed over networks. Computation hiding involves hiding secret data, propriety code, or secret outputs during a distributed computation.

Secure Distributed Computing

Secure distributed computing is also called fault-tolerant distributed computing or oblivious circuit evaluation. The basic underlying mechanism aims

evaluating a function (or a circuit) to which each party has one secret input, so that the output becomes commonly known to all parties but all inputs remain secret. If there is a trusted agent, secure distributed computing is a trivial problem: each party can securely send its private input with the help of cryptographic protocols; the agent computes the function and then distributes the result. Secure distributed computation intends to solve the problem without the assumption of any trusted agent, i.e., to simulate a trusted agent over a set of mutually untrusted parties. Secure distributed computing protocols are built on various basic protocols:

- ***Bit-commitment***: A bit commitment protocol simulates the function a sealed opaque envelope used for committing a bit of information: once the bit is sealed and committed by the committer, the content of the bit can not be changed (the envelope is closed and sealed); the receiver, upon receiving the sealed bit, cannot read it until the content is revealed by the committer (the envelope is opened and letter now can be read). Bit commitment schemes can be built on one-way functions (Naor, 1989). In secure distributed computing, bit commitment schemes are utilized to enable a party to commit to some information.

- ***Oblivious transfer***: Oblivious transfer is a protocol to transfer a bit among the involved parties in a way that the information held by the sender and the receiver about the transfer is asymmetric. A simple version of the oblivious transfer protocol resembles an undependable

post service: A wants to send a bit b to B. Let us assume that the bit is successfully transferred to B with a probability of 0.5. A does not know whether the bit is successfully transferred or not but B always knows the result of transfer; hence the overall information transfer is asymmetric. The importance of oblivious transfer is that it is a basic protocol from which more complicated secure distributed computing protocols can be built and to which all secure distributed computing protocols can be reduced (Kilian, 1988).

- ***Zero-knowledge proofs***: An interactive proof system is a two-party protocol in which a prover owns a secret and wants to convince a verifier that it really has the secret through interaction with the verifier. An interactive proof system is *computationally zero-knowledge* if from the interaction a computationally bounded verifier knows nothing about the secret except the validity of the proof. Every language in NP has a computationally zero-knowledge proof system if a one-way function exists (Goldreich, 1997). This fact is very useful in constructing multi-party secure distributed computing protocols that can tolerate active attackers.

- ***Secret sharing protocol***: A secret sharing protocol, with threshold t, enables a *dealer* to distribute a secret among several players such that each player has an individual share of the secret (called a t-share) and

coalition of any group of maximum t players cannot reconstruct the secret from their shares. A verifiable secret sharing scheme is a scheme that, despite the cheating of a dishonest dealer and some of the players, honest players can still receive valid shares and identify when the dealer cheats. The general idea is to let the dealer distribute *primary shares* of the secret to all players and each player distributes subshares (called *secondary shares*) of its primary share to all other players. Inconsistency of primary share and secondary shares of any player would invoke a challenge-response process and all challenge-response reduce to a conclusion whether the dealer is repudiated (more than a certain number of players accuse it of cheating) or upheld (less than a certain number of players accuse it from cheating). Verifiable secret sharing schemes are fundamental in constructing fault-tolerant secure computation protocols.

- ***Byzantine agreement***: In the Byzantine agreement problem, each party has an initial bit and wants to find if initial bits held by all parties have the same value. The challenge is to ensure that even when there are parties who are dishonest and act as if they have different initial values, all non-faulty parties are able to draw correct conclusions for their initial values. This challenge can be solved if no more than one-third of the parties are faulty (Lamport et al., 1982). Because of the fault-tolerance it provides, Byzantine agreement protocols are employed as backbones to

construct sophisticated fault-tolerant secure computing schemes (Bazzi & Neiger, 1991).

Secure distributed computing protocols may involve only two parties (*two-party secure computing*) or more (*multi-party secure computing*). Most two-party secure computing protocols are built on cryptographic assumptions, such as oblivious transfer, and are based on one of the following two techniques (Franklin et al., 1992): In the first approach, one party *scrambles* the code and its secret input; after scrambling, this party *interacts* with the second party to transfer the scrambled code and input; the second party then *evaluates* the scrambled code with the scrambled input values and sends the output to the first party. In the second one, the two parties *interact* for every logical component of the code in a secure and interactive fashion.

In multiparty secure computing, much attention has been given on fault tolerance, i.e. resilience against active attacks. An active attacker can cause some of the parties to behave against the protocol. Most multiparty secure computing protocols follow three stages. In the *input sharing stage*, each party distributes shares of its private input to other parties, in the *computation stage* each party performs the required computation on shares of private inputs and generates the shares of the ultimate result, and finally in the *output reconstruction stage* shares of the final result are combined by individual parties to recover the result. Zero-knowledge proofs and verifiable secret sharing are essential tools for

multiparty secure computing. Zero-knowledge proofs enable the misconduct of any party to be detected. Verifiable secret sharing (which enables each party to *verifiably* share its secret input with all other parties) guarantees that, if a party is caught cheating, the remaining honest parties can reconstruct its private input and simulate messages it would have sent in the later stages of the protocol. Collectively, these two protocols prevent honest parties involved in a multiparty secure computing from suffering.

Secure computing protocols that can withstand passive attackers are called private secure computing protocols. Secure computing protocols that can withstand active attackers are called resilient secure computing protocols. Two-party secure computing protocols are generally private protocols, as some basic fault-tolerance techniques, such as Byzantine agreement and verifiable secret sharing, used for handling active attacks requires more than two parties' engagements (Franklin et al., 1992).

Instance Hiding

Instance hiding deals with the problem of computing with encrypted data; therefore, it can be regarded as a special case of secure distributed computing. In instance hiding, there is a weak party (e.g. a DAHS client) who has some secret input and wants to compute a function that requires a large amount of computation resources, with the help of strong parties (e.g. DAHS servers): the weak party sends each strong party its encrypted data, and from the partial results computed by the

strong parties it reconstructs the actual output with minimal effort. Instance hiding is especially required in DAHSs, where a computation center enables weak, private computing devices, such as mobile terminals, compute hard problems.

Like many cryptography techniques, most instance hiding schemes are built on hard number theory problems, such as *primitive root*, *quadratic residuosity*, and *discrete logarithm* problems (Abadi et al., 1987).

Not all functions have information theoretic one-server (or one-oracle) instance hiding schemes. For instance, Abadi et al. (1987) prove that no NP-Hard problem has such a scheme. This result is also consistent with the related results that "there exists no information theoretic two-party secure distributed computing protocols for some functions" and that "there exists no information theoretic one-server private information hiding scheme (Subsection Private Information Retrieval) without sending the whole database."

It has been proven, on the other hand, that for any function, even for NP-Hard functions, there always exists information theoretic multi-server (multi-oracle) instance hiding schemes, as long as servers are forbidden from communicating with each other.

Function Hiding

Function hiding deals with the problem of computing with an encrypted function. Function hiding has crucial DAHS applications: the owner of a secret

algorithm can make its code available at a DAHS server in a way that the algorithm of the program and its execution are prevented from disclosure despite intensive code analysis and reverse engineering. In this subsection, we describe various techniques for function hiding:

- ***Matrix modification method***: This method is specifically for hiding polynomial functions that have matrix representations. Hiding is achieved by modifying the function matrix in a randomized way such that the real output can be decrypted from the *randomized* output. Typical examples include *Error Correcting Codes (ECCs) modification* and *similarity transform modification*.

  ECCs modification technique uses an ECC based cryptosystem, such as McEliece public key cryptosystem (Loureiro & Molva, 1999), to hide matrix of polynomial functions. ECC-based security relies on the difficulty of decoding a large linear code with no visible structure.

  Similarity transform modification technique transforms a function matrix, F, into its similarity matrix KFK^{-1} and the input x into Kx, where K is a random invertible matrix serving as the secret key. With similarity transform modification technique, the owner can hide a function with a loop structure within which there is a polynomial calculation. The security of this approach relies on the difficulty of finding the secret key (Badin et al., 2003).

- ***Decomposition method***: This method hides polynomial functions. By asking the server to evaluate all possible terms of a given polynomial, the client can locally construct the result by selecting and merging the relevant components. This way, the server can not learn the polynomial.

- ***Homomorphism encryption***: This method uses a homomorphism encryption function E such that, it is possible to compute $E(x+y)$ directly from $E(x)$ and $E(y)$ (Sander & Tschudin, 1998).

- ***Redundant computation***: This method uses dummy computation and dummy data to hide the real computation and real data.

In general, the types of functions that can be protected via function hiding are very limited, for it is very hard to find encryption schemes for general functions such that the encrypted functions remain executable. Up to now, most function hiding protocols are restricted to hiding polynomial functions.

Note that instance hiding and function hiding are closely related. Theoretically, instance hiding and function hiding problems are equivalent. In some instance hiding schemes, in order to compute with encrypted data, the function is also scrambled in a matching way. If the scrambling hides the original function, function hiding is achieved at the same time. Similarity transform modification technique mentioned above is a good example to this type of hiding. Such symmetric hiding schemes have very important applications in DAHSs: to hide both the private data and code from remote hostile executing environments the owner can publish

secret algorithms on untrusted servers and let those servers provide computation service with the privacy of secret input data also preserved.

A more practical and hence more promising way to protect code and data from leakage and tempering is to provide a certified safe executing environment to the secret code (Smith et al., 1999). Such safe executing environments cannot be inspected or tampered with and, hence, are called temper proof environments (TPEs). A TPE is provided and certified by a trusted manufacturer (TM). The security of TPE cryptographic protocol relies on the trust between the code owner and the trusted manufacturer as well as the security of TPE. To alleviate these requirements, some independent authority can periodically verify and certify the TPE. The TPE is generally a temper-proof hardware; hence, its private keys cannot be leaked without actually destroying it. IBM has a product called "secure co-processor" that can serve as a TPE (Dyer et al., 2001). When it is turned on, the secure co-processor generates public/private key pairs and outputs the public key that can be used for encrypting the code and the data. Since the host does not know the private key to decrypt the code or the data, the only thing it can do is to upload the hidden messages to the TPE, which internally decrypts the code and data before execution.

Private Information Retrieval

Private Information Retrieval (PIR) is a family of encryption protocols seeking to protect users' private queries from malicious data hosts. This is a

relatively new research area in database security. Traditional database security mainly deals with preventing, detecting, and deterring improper disclosure or modification of information in databases. When the database services are made available through third party hosts that cannot be fully trusted, users' queries may be improperly utilized to achieve certain malicious goals. For example, users may not trust their queries being executed on a third party stock database server. Through PIR, users can query the untrusted data source while protecting the confidentiality of their queries and the answers.

The basic idea behind any information theoretic PIR scheme is to replicate the database to several non-communicating servers and ask from each copy of the database a subset of the data that is independent of the target data in a way that the user can reconstruct the target data from query results. Chor et al. (1995) show that PIR can be translated into an oblivious transfer problem and that if one copy of database is used, the only way to hide the query in the information theoretic sense is to send the whole database to the user. In order to reduce even one bit in communication between the server and the user, replication of the whole database is required.

In PIR schemes, communication is the major cost; hence, in order to achieve practical use, it is crucial to reduce the communication cost as well as the number of replicas required. The best known k -server scheme requires $O(n^{2k-1})$ communication (Chor et al., 1995) (where n is the database size). However, to

achieve communication to a subpolynomial in the size of the database, more than a constant number of servers are needed (Chor et al., 1995).

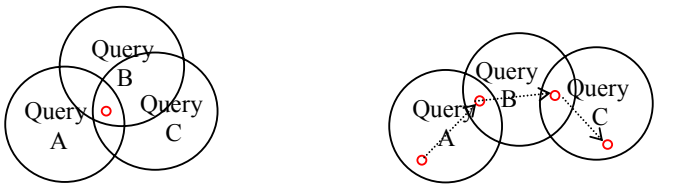
In DAHSs, replication is not a preferable solution to reduce communications in PIR. It is very likely that database owners are reluctant to replicate databases to other servers that they can not keep in contact. Moreover, it may not be possible to prevent third party servers from communicating with each other. In computationally private PIR schemes, therefore, the user privacy requirement is relaxed so that what the servers can see with respect to any two retrieval indexes are indistinguishable for any polynomial time server. PIR schemes built on cryptographic assumptions can further reduce the communication and the number of replicas. If a one-way function exists, then there is a 2-server scheme, such that the communication is $O(n^{\varepsilon})$ for any $\varepsilon > 0$ (Chor et al., 1997). Under the quadratic residuosity assumption, a one-server scheme can be constructed with sub-polynomial communication (Kushilevitz & Ostrovsky, 1997). Under the Φ-hiding assumption, a one-server scheme with a poly-logarithmic communication is possible (Beimel et al., 1999).

Private Informational Retrieval in XML Databases

As discussed earlier, XML has an inherent tree-like structure. Queries over trees are generally expressed in the form of tree path descriptions. XQuery, the standard for XML query language proposed by W3C, is a typical example. Its core is Xpath, the W3C standard for addressing parts of an XML document by describing path patterns.

Figure 7: The intersection property and the movement of the target node.

Big circles represent retrieval of a redundancy sets. Small circles denote the target nodes.



(a) Repeated accesses for a node reveal its location due to intersection

(b) Movement of the node reduces information leakage by intersection.

XML documents can be queried in a navigational manner, i.e. traversing documents along paths described by these patterns and performing join operations on the resulting paths when necessary. However, navigation is often inefficient. Using index mechanisms on the XML structures (i.e., path information) or element values and traversing index trees instead of traversing XML document can reduce query processing time.

On the other hand, if the tree (XML tree or index tree) is traversed in plain to identify the matching paths, the query as well as the result is revealed to the server. Thus hiding the traversal of the tree is necessary for hiding data and queries from untrusted XML servers.

Lin & Candan (2003) propose a protocol to hide tree traversal paths. This protocol allows clients to outsource their sensitive data on servers without any prior trust. In this protocol, tree nodes are encrypted before being outsourced; hence, their

contents (if the nodes are XML elements, also the element types) are hidden from the untrusted data store. Furthermore, to hide the XML tree structure, each time a user wants to retrieve a node, he or she asks for a set of nodes called the *redundancy set* including the target node and additional random nodes. The redundancy set hides the requested node from the server. However, repeated accesses for a particular node can reveal its location, since the node is always within the intersection of the redundant sets of these queries. Figure 7(a) demonstrates this situation. To prevent the server from inferring the locations of the nodes based on repeated node accesses, after each node is retrieved, the node is swapped with a randomly chosen empty node. Thus the node moves with respect to each retrieval, making any correct guessing of its location temporary and of no permanent use. Figure 7(b) depicts the movement of target node. In this figure, there are three consecutive queries A, B, C all of which wants to retrieve the same node. As shown in the figure, the protocol moves the location of the target node after each retrieval. It does so without violating the XML tree structure.

For a reasonable security level, the size of the redundancy set need not be large to hide long paths. If the size of the redundancy set is m, then the probability of finding the target node from a given set is $1/m$, the probability of finding the parent-child relationships in a tree is $1/m^2$, and the probability of finding a given path from root to a leaf is $1/m^{path\ length}$. Hence this protocol is sufficient to hide tree-structured data and queries from any polynomial computation-bounded servers. To enable multiple users to query a tree simultaneously, which is mandatory for an

open and public data store, we also devised deadlock free concurrency control mechanisms (Lin & Candan, 2003). Unlike the information theoretic private information retrieval schemes, the proposed technique requires no replication of the database and the communication cost is $O(m \times \textit{tree depth})$ which is adaptable and generally much less than the size of the database. Compared with general computationally private information retrieval schemes, the proposed technique is much simpler and does not rely on any cryptographic assumptions except for the ones on which the underlying encryption schemes are built.

A detailed study of the security guarantees provided by this protocol requires proper modeling of the user queries as well as the interaction between clients and the server. A request for a redundancy set constitutes a *call*. Each query path retrieval then can be represented as an ordered set of calls. If there are multiple users accessing the system, calls of concurrent queries can be intermixed. Supposing that there is a transport layer security mechanism that hides the identity of owner of each call, we can model DAHS server's view of data accesses as a stream of calls from unknown origins. The server might still be able to infer the tree structure by (a) observing the call stream it receives, (b) guessing which calls in the stream belong to the a single query, (c) guessing which queries it observes are identical or similar, and then (d) looking at the intersection of the redundancy sets of the corresponding calls in each identical query. That is, the server can analyze the calls and intersections of their redundant sets to learn about the tree structure.

Figure 8: Consecutive queries for the same data reveal the path.

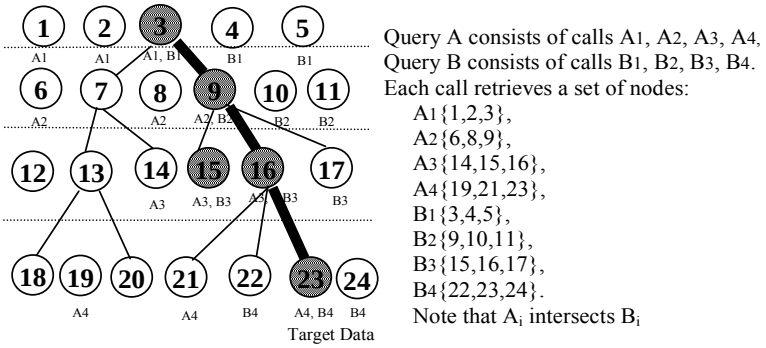
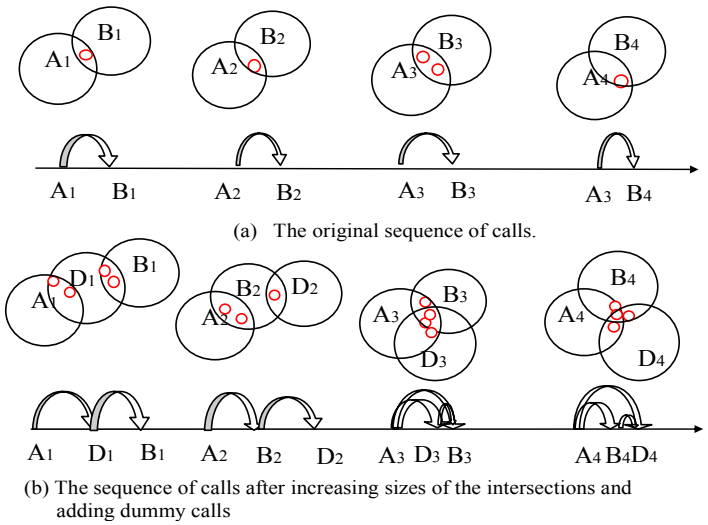


Figure 8 depicts a possible attack. This figure shows the hypothetical case where a query is posed twice consecutively; i.e., without interference from any other queries. In this case, the corresponding calls (e.g., A_2 and B_2) of the two queries intersect. Hence the query path (depicted by bold black line) is revealed.

In order to measure the degree of security that can be provided by the private information retrieval system, it is necessary to understand the probabilities with which the server can use correlations between queries to break the information retrieval security. The server can use the following analyses to attack the system:

- Given two sequences of calls, the server can try to calculate the likelihood of these two sequences containing identical (or similar) queries.
- Given two sequences of calls that are known to contain two identical queries, the server can try to identify the individual calls of them.
- Given the calls of two identical queries, the server can try to discover the query path.

Figure 9: Increased size of intersection and dummy /interfering calls reduce information leaked through the intersection of redundant sets.



These attacks by the server would rely on the intersection property mentioned above. Intuitively, such an attack by the server can be prevented by ensuring that intersections do not reveal much information. This is achieved by modifying the client/server protocols such that the redundant sets intersect at multiple nodes as well as by inserting appropriate dummy/interfering calls. As shown in Figure 9, these methods destroy the relationship between queries and intersecting calls, preventing attackers from exploiting the intersection property.

In Figure 9, each axis tracks the time when calls are posed. An arrow from one call to another represents intersection between them. Figure 9(a) shows that, when there are no interfering calls, the intersection property can be used by a malicious server to identify the existence of identical queries in a given sequence of calls. In addition, if the sizes of the intersections are small, the server can also learn

the actual data nodes. Figure 9(b), on the other hand, shows how by adding dummy calls (D_1,.., D_4) and by increasing the sizes of the intersections, one can limit the information the server can learn studying the intersection of redundant sets. Intuitively, each D_i call adds ambiguity and reduces the probability with which the server can identify the calls that correspond to identical queries. Note that in order to provide efficient and provable security, the process of introducing dummy calls have to follow a strategy which randomizes the intersections with minimal overhead.

This private information retrieval scheme requires legal clients to have access to encryption/decryption keys and be able to perform encryption and decryption operations. Where encryption and decryption constitute heavy computation costs for clients with very limited computation power and memory, we suggest the use of assistant hardware, such as smartcards, to reduce the encryption/decryption execution costs as well as to securely disseminate keys. Part ? of this book details smartcard applications.

CONCLUSION

In this chapter, we provided an overview of the security mechanisms and tools for Distributed Application Host Services (DAHSs) that rent out Internet presence, computation power, and data storage space to clients with infrastructural needs. ASPs and DAHSs operate within the complex, multi-tiered, and open Internet environment and, hence, they introduce many security challenges that have to be addressed effectively. In this chapter, we discussed security challenges in DAHS

from three main aspects: authentication, authorization, and confidentiality. For each aspect, we surveyed current techniques and tools to address these security challenges and discussed open research challenges.

REFERENCES

.NET. (2003). .NET relevant information. Retrieved September 15, 2003, from Microsoft website: www.microsoft.com/net/technical/security.asp, www.microsoft.com/net/business/security_net.asp.

Abadi, M., Feigenbaum, J., & Kilian, J. (1987). On hiding information from an oracle. Proc. of 19th ACM Symposium on Theory of Computing, 195-203.

Atkinson, B., Della-Libera, G., Hada, S. et al. (2002) Specification: Web Service Security (WS-Security) version1.0.05 Retrieved September 15, 2003, from IBM website: http://www-106.ibm.com/developerworks/library/ws-secure/

Badin, R., Bazzi, R. A., Candan, K. S. & Fajri, A. (2003), Provably secure data hiding and tamper resistance for a simple loop program. AeroSense Technologies and Systems for Defense and Security, April 2003, 21-25.

Bazzi, R.A., & Neiger, G. (1991). Optimally simulating crash failures in a Byzantine environment. Proc. of the Fifth International Workshop on Distributed Algorithms, 108-128.

Beimel, A., Ishai, Y., Kushilevitz, E., & Marlkin, T. (1999). One way functions are essential for single-server private information retrieval. Proc. of the 31st Annual ACM Symposium on Theory of Computing, 89-98.

Bertino, E., Castano, S., & Ferrari, E. (2001). Securing XML documents with Author-X. IEEE Internet Computing, 5(3), 21-31.

Bertino, E., Catania, B., Ferrari, E., Thuraisingham, B. M., & Gupta, A.(2002).Selective and authentic third-party distribution of XML documents, MIT Sloan working paper No. 4343-02.

Blaustein, B. T., McCollum, C. D., Notargiacomo, L., Smith, K. P., & Graubart, R. D. (1995). Autonomy and confidentiality: Secure Federated Data Management. The 2nd International Workshop on Next Generation Information Technologies and Systems.

Bonatti, P. A., Sapino, M. L. & Subrahmanian, V. S. (1996). Merging heterogeneous security orderings. European Symposium on Research in Computer Security 1996, 183-197.

Bonatti, P. A., Kraus, S., & Subrahmanian, V. S. (2003). Secure agents. Annuals of Mathematic and Artificial Intelligence, 37(1-2) 169-235.

Butt, A. R., Adabala, S., Kapadia, N. H., Figueiredo, R., & Fortes, J. A. B. (2002). Fine-grain access control for securing shared resources in computational

grids. Retrieved September15, 2003, from IEEE Computer Society website: http://computer.org/proceedings/ipdps/1573/symposium/15730022babs.htm

Candan, K. S., Jajodia, S., & Subrahmanian, V. S. (1996). Secure mediated databases. IEEE Conference on Data Engineering, 1996, 28-37.

Cho, S., Amer-Yahia, S., Lakshmanan, L. V. S., & Srivastava, D. (2002). Optimizing the secure evaluation of twig queries. Proc of the 28th Very Large Data Bases Conference, 490-501.

Chor, B., & Gilboa, N. (1997). Computationally private information retrieval. Proc. of the 29th Annual ACM Symposium on the Theory of Computing, 304-313.

Chor, B., Goldreich, O., Kushilevitz, E., & Sudan, M. (1995). Private information retrieval. Proc. of 36th IEEE Conference on the Foundations of Computer Sciences, 41-50.

DB2. (2003). DB2 relevant information, retrieved May 15, 2003, from IBM website: http://www7b.boulder.ibm.com/dmdd/.

Devanbu, P. T., Gertz, M., Martel, C. U., & Stubblebine, S. G. (1999). Authentic third-party data publication, 14th IFIP 11.3 Conference on Database Security.

Devanbu, P. T., Gertz, M., Kwong, A., Martel, C. U., Nuckolls, G., & Stubblebine, S. G. (2001). Flexible authentication of XML documents. ACM Conference on Computer and Communications Security, 2001, 136-145.

Dyer, J. G., Lindemann, M., Perez, R., Sailer, R., Doorn, L. V., Smith, S. W., & Weingart, S. (2001). Building the IBM 4758 secure coprocessor. IEEE Computer, 34(10), 57-66.

Eastlake, D., Reagle, J. et al. (2003). W3C XML Signature WG. Retrieved on September 15, 2003, from W3C website: http://www.w3.org/Signature.

Franklin, M., Galil, Z., & Yung, M. (1992). An overview of secure distributed computing. Technical Report, TR CUCS-008-92, Columbia University, 1992.

Fujisaki, E., Okamoto, T., Pointcheval, D., & Stern, T. (2000) RSA-OAEP is still alive! Retrieved September 15, 2003, from Cryptology ePrint Archive website: http://eprint.iacr.org/.

Goldreich, O. (1997). On the foundations of modern cryptography. In B. Kaliski (Ed.), Advances in Cryptology Crypto '97 (pp. 46-74). Spring-Verlag.

Hégaret, P. L. et al. (2002). Document Object Model (DOM). Retrieved September 15, 2003, from W3C website: http://www.w3.org/DOM/.

Idris, N. B., Gray, W. A. & Churchhouse, R. F. (1994). Providing dynamic security control in a federated database. Proc of the 20th Very Large Data Bases Conference, 13-23.

J2EE. (2003). J2EE relevant information. Retrieved September 15, 2003 from Sun java website: http://java.sun.com on Java security APIs, such as JAAS, JSSE, JCE.

Jonscher, D., & Dittrich, K. R. (1994). An approach for building secure database federations. Proc of the 20th Very Large Data Bases Conference, 24-35.

Karnik, N. M. (1998). Security in mobile agent systems. Ph.D. Thesis, University of Minnesota. Retrieved September 15, 2003, from University of Minnesota website: http://www.cs.umn.edu/Ajanta.

Kilian, J. (1988). Founding cryptography on oblivious transfer. Proc. of 20th ACM Symposium on Theory of Computing, 20-31.

Kohl, J., & Neuman, C. (1993). The Kerberos network authentication service. Retrieved September 15, 2003, from Internet RFC/STD/FYI/BCP Archives website: http://www.faqs.org/rfcs/rfc1510.html.

Kushilevitz, E., & Ostrovsky, R. (1997). Replication is not needed: single database, computationally-private information retrieval. Proc.38th IEEE Symposium on Foundations of Computer Sciences, 365-373.

Lamport, L., Shostak, R., & Pease, M. C. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4, 382-401.

Lin, P., & Candan, K. S. (2003). Hiding traversal of tree structured data from untrusted data stores. In H. Chen et al (Ed.), Intelligence and Security Informatics (pp 385). Springer-Verlag.

Loureiro, S., & Molva, R. (1999). Function hiding based on error correcting codes. Proc. of Cryptec'99 International Workshop on Cryptographic techniques and Electronic Commerce, 92-98.

Maruyama, H., Tamura, & K., Uramoto, N. (2000). Digest Values for DOM (DOMHASH), RFC2803. Retrieved September 15, 2003, from Internet RFC/STD/FYI/BCP Archives website: http://www.faqs.org/rfcs/rfc2803.html.

Minsky, Y., Renesse, R. V., Schneider, F. B., & Stoller, S. D. (1996). Cryptographic support for fault-tolerant distributed computing**.** Proc of 7th ACM Special Interest Group on Operating Systems European Workshop*,* 109-114**.**

Nace, W. A., & Zmuda, J. E. (1997). PPP EAP DSS public key authentication protocol. Retrieved September 15, 2003, from GlobeCom IETF library.

Naor, M. (1989). Bit commitment using pseudo-randomness, G. Brassard (Ed.), Advances in Cryptology - Crypto '89 (pp. 128-136). Springer-Verlag.

Necula, G. C., & Lee, P. (1998). Safe, untrusted agents using proof-carrying code, In G. Vigna (Ed.), Mobile Agents and Security (pp. 61-91). Springer-Verlag.

Oracle. (2003). Oracle relevant information. Retrieved September 15, 2003, from Oracle website: http://otn.oracle.com/deploy/security/content.html.

Reagle, J. et al. (2003) W3C XML Encryption WG. Retrieved September 15, 2003, from W3C website: http://www.w3.org/Encryption/2001/.

Sander, T., & Tschudin, C. F. (1998). Protecting mobile agents against malicious hosts. In G. Vigna (Ed.), Mobile agent security (pp 44-61). Springer-Verlag.

Sandhu, R. S., & Samarati, P. (1994). Access control: principles and practice, IEEE Communications Magazine, 32(9), 40-48.

Smith, S. W., Perez, R., Weingart, S., & Austel, V. (1999). Validating a high-performance, programmable secure coprocessor. 22nd National Information Systems Security Conference.

Vigna, G. (1998). Cryptographic traces for mobile agents. In G. Vigna (Ed.), Mobile Agents and Security (pp.137-153). Springer-Verlag.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. IEEE Computer, 25(3), 38-49.

Wiederhold, G. (1993). Intelligent integration of information. Proc. of the ACM Special Interest Group on Management of Data International Conference on Management of Data, 22(2), 434-437.

Wright, T. (2000). Proposal for WAP-IETF co-operation on a wireless friendly TLS. Retrieved September 15, 2003, from the Internet Engineering Task Force website: http://www.ietf.org/proceedings/00jul/SLIDES/tls-wtls/.

XML Security Suite. (1999). Retrieved September 15, 2003, from IBM website: http://www.alphaworks.ibm.com/tech/xmlsecuritysuite.

Yee, B. S. (1999). A sanctuary for mobile agents. In J. Vitek and C. Jensen (Ed.), Secure Internet Programming: Security Issues for Distributed and Mobile Objects (pp.261–273). Springer-Verlag.

Yu, T., Srivastava, D., Lakshmanan, L. V. S., & Jagadish, H. V. (2002). Compressed accessibility map: efficient access control for XML. Proc of the 28th Very Large Data Bases Conference, 478-489.