UNIT-1

Introduction to Database Security

Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability. This article will focus primarily on confidentiality since it's the element that's compromised in most data breaches.

Database security must address and protect the following:

- The data in the database
- The database management system (DBMS)
- Any associated applications
- The physical database server and/or the virtual database server and the underlying hardware
- The computing and/or network infrastructure used to access the database

Database security is a complex and challenging endeavor that involves all aspects of information security technologies and practices. It's also naturally at odds with database usability. The more accessible and usable the database, the more vulnerable it is to security threats; the more invulnerable the database is to threats, the more difficult it is to access and use. (This paradox is sometimes referred to as Anderson's Rule).

Why is it important

By definition, a data breach is a failure to maintain the confidentiality of data in a database. How much harm a data breach inflicts on your enterprise depends on a number of consequences or factors:

- **Compromised intellectual property:** Your intellectual property—trade secrets, inventions, and proprietary practices—may be critical to your ability to maintain a competitive advantage in your market. If that intellectual property is stolen or exposed, your competitive advantage may be difficult or impossible to maintain or recover.

- **Damage to brand reputation:** Customers or partners may be unwilling to buy your products or services (or do business with your company) if they don't feel they can trust you to protect your data or theirs.

- **Business continuity (or lack thereof):** Some business cannot continue to operate until a breach is resolved.

- **Fines or penalties for non-compliance:** The financial impact for failing to comply with global regulations such as the Sarbanes-Oxley Act (SAO) or Payment Card Industry Data Security Standard (PCI DSS), industry-specific data privacy regulations such as HIPAA, or regional data privacy regulations, such as Europe's General Data Protection Regulation (GDPR) can be devastating, with fines in the worst cases exceeding several million dollars *per violation*.

- **Costs of repairing breaches and notifying customers:** In addition to the cost of communicating a breach to customer, a breached organization must pay for forensic and investigative activities, crisis management, triage, repair of the affected systems, and more.

Risk Analysis:

A security risk assessment identifies, assesses, and implements key security controls in applications. It also focuses on preventing application security defects and vulnerabilities. Carrying out a risk assessment allows an organization to view the application portfolio holistically—from an attacker's perspective. It supports managers in making informed resource allocation, tooling, and security control implementation decisions. Thus, conducting an assessment is an integral part of an organization's risk management process..

How does a security risk assessment work?

Factors such as size, growth rate, resources, and asset portfolio affect the depth of risk assessment models. Organizations can carry out generalized assessments when experiencing budget or time constraints. However, generalized assessments don't necessarily provide the detailed mappings between assets, associated threats, identified risks, impact, and mitigating controls.

If generalized assessment results don't provide enough of a correlation between these areas, a more in-depth assessment is necessary.

The 4 steps of a successful security risk assessment model:

1. **Identification**. Determine all critical assets of the technology infrastructure. Next, diagnose sensitive data that is created, stored, or transmitted by these assets. Create a risk profile for each.

2. **Assessment**. Administer an approach to assess the identified security risks for critical assets. After careful evaluation and assessment, determine how to effectively and efficiently allocate time and resources towards risk mitigation. The assessment approach or methodology must analyze the correlation between assets, threats, vulnerabilities, and mitigating controls.

3. **Mitigation**. Define a mitigation approach and enforce security controls for each risk.

4. **Prevention**. Implement tools and processes to minimize threats and vulnerabilities from occurring in your firm's resources.

A comprehensive security assessment allows an organization to:

- Identify assets (e.g., network, servers, applications, data centers, tools, etc.) within the organization.
- Create risk profiles for each asset.
- Understand what data is stored, transmitted, and generated by these assets.
- Assess asset criticality regarding business operations. This includes the overall impact to revenue, reputation, and the likelihood of a firm's exploitation.
- Measure the risk ranking for assets and prioritize them for assessment.
- Apply mitigating controls for each asset based on assessment results.

Cryptography and Web Security:

Increasingly, systems that employ cryptographic techniques are used to control access to computer systems and to sign digital messages. Cryptographic systems have also been devised to allow the anonymous exchange of digital money and even to facilitate fair and unforgivable online voting.

Roles for Cryptography:

Security professionals have identified five different roles that encryption can play in modern information systems. In the interest of sharing a common terminology, each of these different roles is identified by a specific keyword. The roles are:

Authentication:

Digital signatures can be used to identify a participant in a web transaction or the author of an email message; people who receive a message that is signed by a digital signature can use it to verify the identity of the signer. Digital signatures can be used in conjunction with passwords and biometrics or as an alternative to them.

Authorization:

Whereas authentication is used to determine the identity of a participant, authorization techniques are used to determine if that individual is authorized to engage in a particular transaction. Cryptographic techniques can be used to distribute a list of authorized users that is all but impossible to falsify.

Confidentiality:

Encryption is used to scramble information sent over networks and stored on servers so that eavesdroppers cannot access the data's content. Some people call this quality "privacy," but most professionals reserve that word for referring to the protection of personal information (whether confidential or not) from aggregation and improper use.

Integrity:

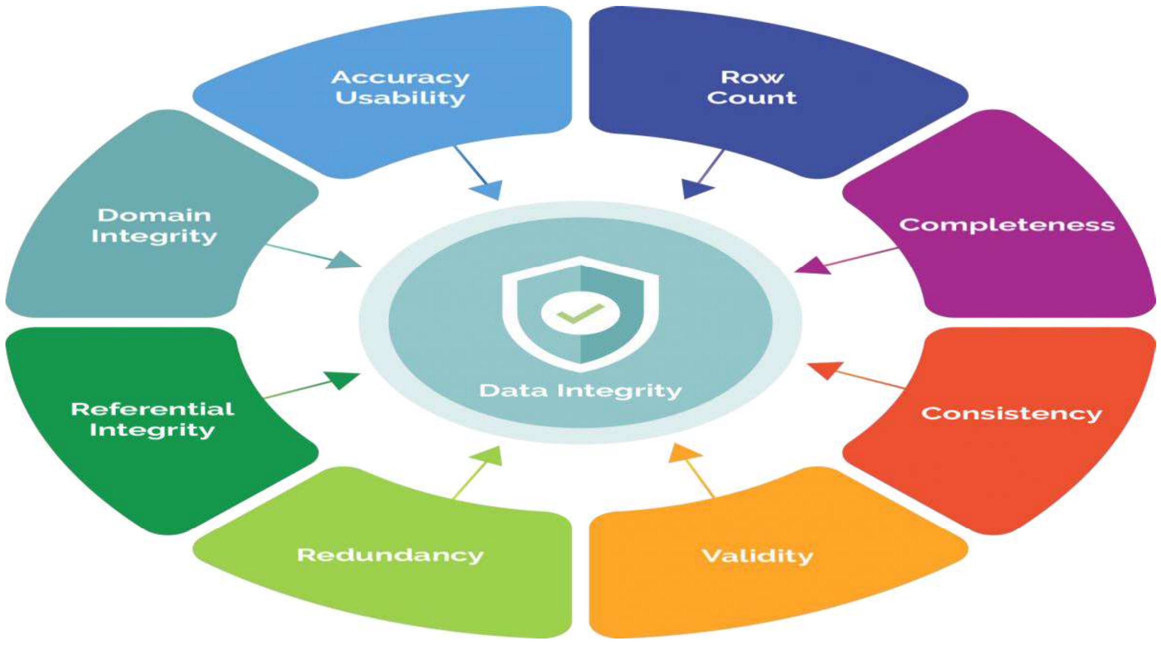
Methods that are used to verify that a message has not been modified while in transit. Often, this is done with digitally signed message digest codes.

Nonrepudiation: Cryptographic receipts are created so that an author of a message cannot realistically deny sending a message (but see the discussion later in this section).Strictly speaking, there is some overlap among these areas. For example, when a message is encrypted to provide confidentiality, an unexpected byproduct is often integrity. That's because many encrypted messages will not decrypt if they are altered.

Current practices, however, dictate that it is better to use algorithms that are specifically designed to assure integrity for this purpose, rather than relying on integrity as a byproduct of other algorithms. Using separate algorithms allows finer control of the underlying processes. Using separate algorithms for confidentiality, authentication, and integrity also minimizes the impact of any legal restrictions that apply to cryptography, because these restrictions are usually aimed at confidentiality but not other cryptographic practices.

Nonrepudiation means adding assurance mechanisms to verify the identity and intent of the user. This is needed so the user cannot claim, after the fact, that she did not actually conduct the transaction. This claim may be phrased as a denial that the activity was ever conducted, or it may be a claim that someone else was using her account. Although nonrepudiation is often listed as one of the advantages of public key technology, the "nonrepudiation" provided by this technology is not true nonrepudiation. Public key technology can prove that a certain private key was used to create a digital signature, but it cannot prove the intent of the key's user.

"Nonrepudiation," as the term is commonly used by cryptographers, analysts, and even lawmakers, is simply not possible. Even if the cryptography is perfect, the person's computer might be infected with a virus that causes it to behave in a manner other than what's intended. Smart cards and biometrics do not solve the nonrepudiation problem either—you might insert your smart card into a reader, thinking you are signing an electronic check to subscribe to a magazine, only to discover that a hostile ActiveX control has noticed the insertion and used your smart card to authorize the transfer of $1000 out of your bank account. Or a crook may force your signature at gunpoint. People can *always* repudiate something that a computer has done on their behalf.



WORKING CRYPTOGRAPHIC SYSTEMS AND PROTOCALS:

A cryptographic system is a collection of software and hardware that can encrypt or decrypt information. A typical cryptographic system is the combination of a desktop computer, a web browser, a remote web server, and the computer on which the web server is running.

Cryptographic protocols and algorithms are difficult to get right, so do not create your own. Instead, where you can, use protocols and algorithms that are widely-used, heavily analyzed, and accepted as secure. When you must create anything, give the approach wide public review and make sure that professional security analysts examine it for problems. In particular, do not create your own encryption algorithms unless you are an expert in cryptology, know what you're doing, and plan to spend years in professional review of the algorithm. Creating encryption algorithms (that are any good) is a task for experts only.

A number of algorithms are patented; even if the owners permit "free use" at the moment, without a signed contract they can always change their minds later, putting you at extreme risk later. In general, avoid all patented algorithms - in most cases there's an unpatented approach that is at least as good or better technically, and by doing so you avoid a large number of legal problems.

Another complication is that many counties regulate or restrict cryptography in some way. A survey of legal issues is available at the "Crypto Law Survey" site, http://rechten.kub.nl/koops/cryptolaw/.

Often, your software should provide a way to reject "too small" keys, and let the user set what "too small" is. For RSA keys, 512 bits is too small for use. There is increasing evidence that 1024 bits for RSA keys is not enough either; Bernstein has suggested techniques that simplify brute-forcing RSA, and other work based on it (such as Shamir and Tromer's "Factoring Large Numbers with the TWIRL device") now suggests that 1024 bit keys can be broken in a year by a $10 Million device. You may want to make 2048 bits the minimum for RSA if you really want a secure system.

Cryptographic Protocols:

When you need a security protocol, try to use standard-conforming protocols such as IPSec, SSL (soon to be TLS), SSH, S/MIME, OpenPGP/GnuPG/PGP, and Kerberos. Each has advantages and disadvantages; many of them overlap somewhat in functionality, but each tends to be used in different areas:

- **Internet Protocol Security (IPSec)**. IPSec provides encryption and/or authentication at the IP packet level. However, IPSec is often used in a way that only guarantees authenticity of two communicating hosts, not of the users. As a practical matter, IPSec usually requires low-level support from the operating system (which not all implement) and an additional keyring server that must be configured. Since IPSec can be used as a "tunnel" to secure packets belonging to multiple users and multiple hosts, it is especially useful for building a Virtual Private Network (VPN) and connecting a remote machine. As of this time, it is much less often used to secure communication from individual clients to servers. The new version of the Internet Protocol, IPv6, comes with IPSec "built in," but IPSec also works with the more common IPv4 protocol. Note that if you use IPSec, don't use the encryption mode without the authentication, because the authentication also acts as integrity protection.
- **Secure Socket Layer (SSL) / TLS**. SSL/TLS works over TCP and tunnels other protocols using TCP, adding encryption, authentication of the server, and optional authentication of the client (but authenticating clients using SSL/TLS requires that clients have configured X.509 client certificates, something rarely done). SSL version 3 is widely used; TLS is a later adjustment to SSL that strengthens its security and improves its flexibility. Currently there is a slow transition going on from SSLv3 to TLS, aided because implementations can easily try to use TLS and then back off to SSLv3 without user intervention.

Unfortunately, a few bad SSLv3 implementations cause problems with the backoff, so you may need a preferences setting to allow users to skip using TLS if necessary. Don't use SSL version 2, it has some serious security weaknesses.

SSL/TLS is the primary method for protecting http (web) transactions. Any time you use an https:// URL, you're using SSL/TLS. Other protocols that often use SSL/TLS include POP3 and IMAP. SSL/TLS usually use a separate TCP/IP port number from the unsecured port, which the IETF is a little unhappy about (because it consumes twice as many ports; there are solutions to this). SSL is relatively easy to use in programs, because most library implementations allow programmers to use operations similar to the operations on standard sockets like SSL_connect(), SSL_write(), SSL_read(), etc. A widely used OSS/FS implementation of SSL (as well as other capabilities) is OpenSSL, available at http://www.openssl.org.

- **OpenPGP and S/MIME**. There are two competing, essentially incompatible standards for securing email: OpenPGP and S/MIME. OpenPHP is based on the PGP application; an OSS/FS implementation is GNU Privacy Guard from http://www.gnupg.org. Currently, their certificates are often not interchangeable; work is ongoing to repair this.

- **SSH**. SSH is the primary method of securing "remote terminals" over an internet, and it also includes methods for tunelling X Windows sessions. However, it's been extended to support single sign-on and general secure tunelling for TCP streams, so it's often used for securing other data streams too (such as CVS accesses). The most popular implementation of SSH is OpenSSH http://www.openssh.com, which is OSS/FS. Typical uses of SSH allows the client to authenticate that the server is truly the server, and then the user enters a password to authenticate the user (the password is encrypted and sent to the other system for verification). Current versions of SSH can store private keys, allowing users to not enter the password each time. To prevent man-in-the-middle attacks, SSH records keying information about servers it talks to; that means that typical use of SSH is vulnerable to a man-in-the-middle attack during the very first connection, but it can detect problems afterwards. In contrast, SSL generally uses a certificate authority, which eliminates the first connection problem but requires special setup (and payment!) to the certificate authority.

- **Kerberos**. Kerberos is a protocol for single sign-on and authenticating users against a central authentication and key distribution server. Kerberos works by giving authenticated users "tickets", granting them access to various services on the network. When clients then contact servers, the servers can verify the tickets. Kerberos is a primary method for securing and supporting authentication on a LAN, and for establishing shared secrets (thus, it needs to be used with other algorithms for the actual protection of communication). Note that to use Kerberos, both the client and server have to include code to use it, and since not everyone has a Kerberos setup, this has to be optional - complicating the use of Kerberos in some programs. However, Kerberos is widely used.

AES Standards	**Key Size (in bits)**	**Block Size (in bits)**	**Number of Rounds**
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Digital Signature Algorithm:

Digital Signature Algorithm (DSA) is also a public key algorithm that is used only for digitally signing documents. This scheme is suitable for achieving authentication before a message or documents are shared (Forouzan, 2011). Receiving a digitally signed document, the recipient becomes confident that the sender was a genuine one and the document was not altered during the transmission. Digital signatures are applied in software distribution, financial transactions, and for documents that might be tampered with. To verify the document the receiver performs the following steps:

1. Decrypts the digital signature using the sender's public key to read the message.
2. Generates a message digest for the receiver's message using the same algorithm used by the sender.
3. If both message digests do not match, the sender's message digest is considered to be compromised.

Hash functions:

Hash functions or one-way functions are used in public-key cryptography for implementing protocols (Alawida et al., 2021). Hash functions do not need any key. They are easily computable but harder to reverse. For example, f(x) can be computed easily but the computation of x from f(x) will take many years even for all the computers of the world collectively. The value of f(x) is a fixed-length hash value computed out of x which is the plaintext. Neither the contents of the plaintext nor its length can be obtained. Hash functions are used to verify the integrity of the documents and encryption of passwords. Even a small bit of change in the contents can be easily detected because the hash values of the two versions will be absolutely different.

Cryptographic protocol:

Cryptography analyses the issues of integrity, authentication, privacy, and Nonrepudiation. Cryptographic algorithms are having academic importance (Schneier, 2007). Application of these algorithms alone cannot guarantee to achieve the goal of Cryptography. Well-defined policies and agreements between the parties involved in the communication are also required in order to make Cryptography a reliable technology for achieving its goals so that it can solve real problems in completing online tasks between trusted parties.
A cryptographic protocol is a distributed algorithm designed to precisely describe the interactions between two or more parties with the objective of implementing certain security policies. It follows some series of steps in exact sequence. Every step must be completely executed without any alteration in the agreed-upon sequence. It must be complete and able to finish a task. At least two parties are required. Any single party executing a series of steps to complete a task is not a protocol. Every party must know, understand, and follow it. They must not be able to do something beyond the specified agreement.

Arbitrated Protocols:

Arbitrated protocols use a trusted third party called an arbitrator. The arbitrator has no vested interest and cannot favor any of the involved parties. Such protocols are used to complete tasks between two or more parties not trusting each other.

Adjudicated Protocols:

The arbitrated protocols are implemented with two sub protocols to reduce the cost of third-party involvement. Some non-arbitrated protocol is used in the first level which is executed for each task. In the second level, an arbitrated protocol is used which is executed only in case of disputes occur between the involved parties during the task.

Self-Enforcing Protocols:

These protocols require no arbitrator to complete tasks or to resolve disputes. The protocol itself ensures that there is no dispute between the involved parties. One party can detect whenever the other party is trying to play smart and the task is stopped immediately. It is ideal that every protocol should be self-enforcing. Similar to the attacks on Cryptographic algorithms and techniques, protocols can also be attacked by the cheaters.

<u>Types of Protocols:</u>

Key Exchange Protocols:

A key exchange protocol is required for two parties to reach an agreement for a shared secret key. Either one party can authenticate the other or both parties can authenticate each other. The protocol can agree for the generation of a random key. One party can generate the key and send it to another party or both parties can participate in the key generation.

Diffie-Hellman key exchange:

This protocol is used by the involved parties to agree on a shared key by exchanging messages through a public channel. Therefore, the key is not revealed to any unauthorized party. This is protected only against passive attacks.

Identification and Authentication Protocols:

Identification protocols are required to ensure the identity of both parties when they are online for a task. Genuine possession of their private keys needs to be verified. The level of identification by the protocols may be judged with three levels: (1) Who is he? – Biometrics is used, (2) What he possesses? –Some hardware gadgets can be used, (3) What he knows? – Secret keys or passwords are used. Some popular protocols are zero-knowledge protocol, Schnorr Protocol, Guillou-Quisquater protocol, witness hiding identification protocols, etc.

Using Password Authentication:

In absence of any digital signature scheme the two parties can share a password that is comparatively less powerful.

Issues in Cryptography

In symmetric cryptography, if the key is lost, communication cannot be completed. This creates an issue of secure key distribution with possibly involving either the sender and the receiver to communicate directly or via a trusted third party or communicating via an existing cryptographic medium (Sharma et al., 2021). The issue of key distribution is to be dealt with delicately: keys must be stored, used, as well as destroyed securely.

Cryptography only transforms plaintext but never hides it (Rahmani et al., 2014). One weakness of Cryptography is if somehow any third party detects the presence of an encrypted message, it can make attempts to break into it out of curiosity. Sometimes curiosity feeds the cat. As a consequence, it can reveal the secrecy, modify or misuse the information.

Legal Restrictions on Cryptography:

The legal landscape of cryptography is complex and constantly changing. In recent years the legal restrictions on cryptography in the United States have largely eased, while the restrictions in other countries have increased somewhat.

Cryptography and the Patent System:

Patents applied to computer programs, frequently called *software patents*, have been accepted by the computer industry over the past thirty years-some grudgingly, and some with great zeal.

The *doctrine of equivalence* holds that if a new device operates in substantially the same way as a patented device and produces substantially the same result, then the new device infringes the original patent. As a result of this doctrine, which is one of the foundation principles of patent law, a program that implements a patented encryption technique will violate that patent, even if the original patent was on a machine built from discrete resistors, transistors, and other components. Thus, the advent of computers that were fast enough to implement basic logic circuits in software, combined with the acceptance of patent law and patents on electronic devices, assured that computer programs would also be the subject of patent law.

The outlook for patents:

Although new encryption algorithms that are protected by patents will continue to be invented, a wide variety of unpatented encryption algorithms now exist that are secure, fast, and widely accepted. Furthermore, there is no cryptographic operation in the field of Internet commerce that requires the use of a patented algorithm. As a result, it appears that the overwhelming influence of patent law in the fields of cryptography and e-commerce have finally come to an end.

Cryptography and Trade Secret Law:

Until very recently, many nontechnical business leaders mistakenly believed that they could achieve additional security for their encrypted data by keeping the encryption algorithms themselves secret. Many companies boasted that their products featured *proprietary encryption algorithms.* These companies refused to publish their algorithms, saying that publication would weaken the security enjoyed by their users.

Today, most security professionals agree that this rationale for keeping an encryption algorithm secret is largely incorrect. That is, keeping an encryption algorithm secret does not significantly improve the security that the algorithm affords. Indeed, in many cases, secrecy actually decreases the overall security of an encryption algorithm.

There is a growing trend toward academic discourse on the topic of cryptographic algorithms. Significant algorithms that are published are routinely studied, analyzed, and occasionally found to be lacking. As a result of this process, many algorithms that were once trusted have been shown to have flaws. At the same time, a few algorithms have survived the rigorous process of academic analysis. Some companies think they can short-circuit this review process by keeping their algorithms secret. Other companies have used algorithms that were secret but widely licensed as an attempt to gain market share and control. But experience has shown that it is nearly impossible to keep the details of a successful encryption algorithm secret. If the algorithm is widely used, then it will ultimately be distributed in a form that can be analyzed and reverse-engineered.

Regulation of Cryptography by International and National Law:

In the past 50 years there has been a growing consensus among many governments of the world on the need to regulate cryptographic technology. The original motivation for regulation was military. During World War II, the ability to decipher the Nazi Enigma machine gave the Allied forces a tremendous advantage-Sir Harry Hinsley estimated that the Allied "ULTRA" project shortened the war in the Atlantic, Mediterranean, and Europe "by not less than two years and probably by four years."As a result of this experience, military intelligence officials in the United States and the United Kingdom decided that they needed to control the spread of strong encryption technology-lest these countries find themselves in a future war in which they could not eavesdrop on the enemy's communications.

U.S. regulatory efforts and history:

Export controls in the United States are enforced through the Defense Trade Regulations (formerly known as the International Traffic in Arms Regulation-ITAR). In the 1980s, any company wishing to export a machine or program that included cryptography needed a license from the U.S. government. Obtaining a license could be a difficult, costly, and time consuming process.

Following the 1992 compromise, the Clinton Administration made a series of proposals designed to allow consumers the ability to use full-strength cryptography to secure their communications and stored data, while still providing government officials relatively easy access to the plaintext, or the unencrypted data. These proposals were all based on a technique called *key escrow.* The first of these proposals was the administration's Escrowed Encryption Standard (EES), more commonly known as the Clipper chip.

In 1997, an ad hoc group of technologists and cryptographers issued a report detailing a number of specific risks regarding all of the proposed key recovery, key escrow, and trusted third-party encryption schemes:

*The potential for insider abuse-*There is fundamentally no way to prevent the compromise of the system "by authorized individuals who abuse or misuse their positions. Users of a key recovery system must trust that the individuals designing, implementing, and running the key recovery operation are indeed trustworthy. An individual, or set of individuals, motivated by ideology, greed, or the threat of blackmail, may abuse the authority given to them.

The creation of new vulnerabilities and targets for attack: Securing a communications or data storage system is hard work; the key recovery systems proposed by the government would make the job of security significantly harder because more systems would need to be secured to provide the same level of security.

*Scaling might prevent the system from working at all-*The envisioned key recovery system would have to work with thousands of products from hundreds of vendors; it would have to work with key recovery agents all over the world; it would have to accommodate tens of thousands of law enforcement agencies, tens of millions of public-private key pairs, and hundreds of billions of recoverable session keys.

The difficulty of properly authenticating requests for keys- A functioning key recovery system would deal with hundreds of requests for keys every week coming from many difficult sources. How could all these requests be properly authenticated?

*The cost-*Operating a key recovery system would be incredibly expensive. These costs include the cost of designing products, engineering the key recovery center itself, actual operation costs of the center, and (we hope) government oversight costs. Invariably, these costs would be passed along to the end users, who would be further saddled with "both the expense of choosing, using, and managing key recovery systems and the losses from lessened security and mistaken or fraudulent disclosures of sensitive data."

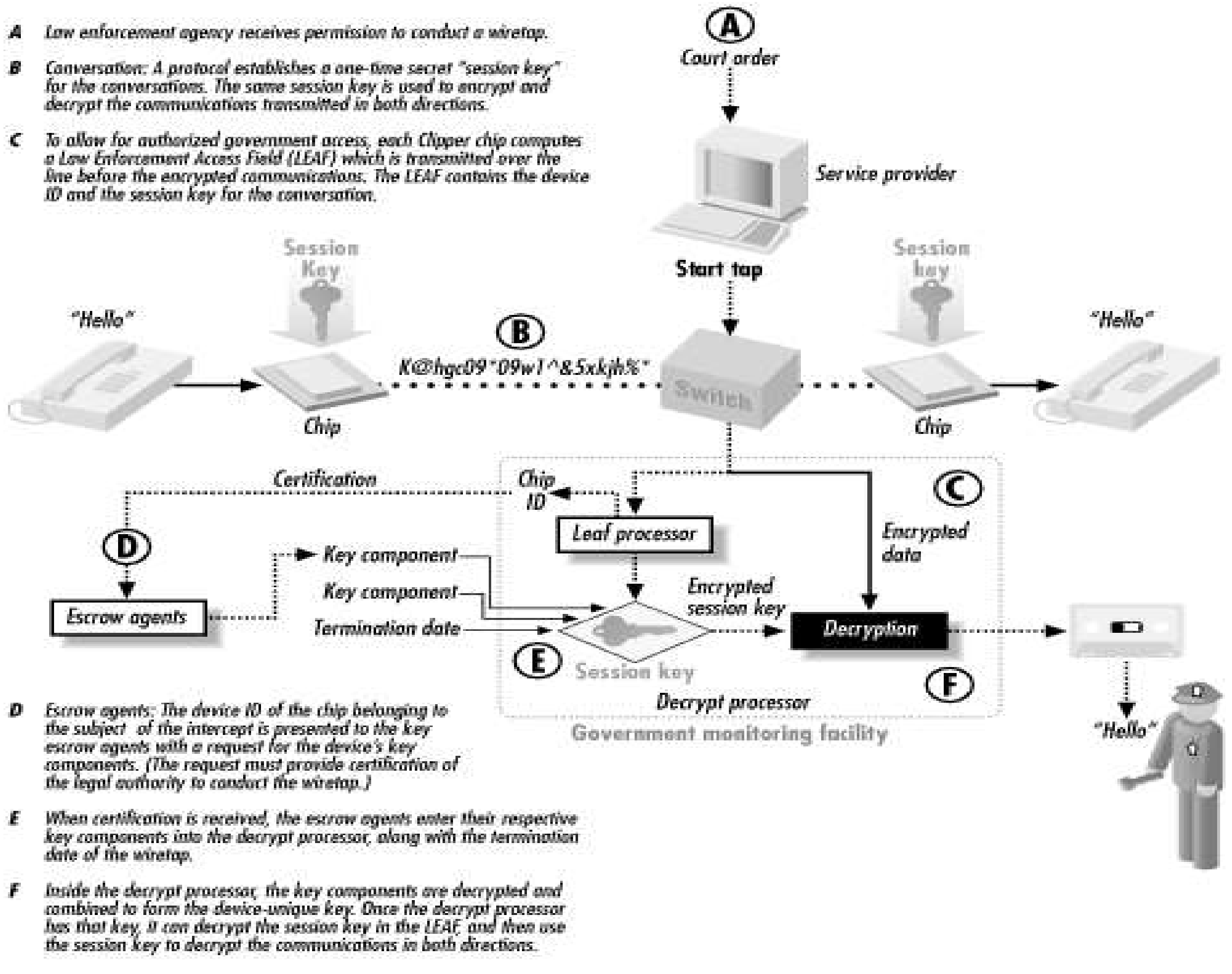
The Digital Millennium Copyright Act:

There is a huge and growing market in digital media. Pictures, e-books, music files, movies, and more represent great effort and great value. Markets for these items are projected to be in the billions of dollars per year. The Internet presents a great medium for the transmission, rental, sale, and display of this media. However, because the bits making up these items can be copied repeatedly, it is possible that fraud and theft can be committed easily by anyone with access to a copy of a digital item.

International agreements on cryptography:

International agreements on the control of cryptographic software date back to the days of COCOM (Coordinating Committee for Multilateral Export Controls), an international organization created to control the export and spread of military and dual-use products and technical data.

The Council of Europe is a 41-member intergovernmental organization that deals with policy matters in Europe not directly applicable to national law. On September 11, 1995, the Council of Europe adopted Recommendation R (95) 13 Concerning Problems of Criminal Procedure Law Connected with Information Technology, which stated, in part, that "measures should be considered to minimize the negative effects of the use of cryptography on the investigation of criminal offenses, without affecting its legitimate use more than is strictly necessary."



A *Law enforcement agency receives permission to conduct a wiretap.*

B *Conversation: A protocol establishes a one-time secret "session key" for the conversations. The same session key is used to encrypt and decrypt the communications transmitted in both directions.*

C *To allow for authorized government access, each Clipper chip computes a Law Enforcement Access Field (LEAF) which is transmitted over the line before the encrypted communications. The LEAF contains the device ID and the session key for the conversation.*

D *Escrow agents: The device ID of the chip belonging to the subject of the intercept is presented to the key escrow agents with a request for the device's key components. (The request must provide certification of the legal authority to conduct the wiretap.)*

E *When certification is received, the escrow agents enter their respective key components into the decrypt processor, along with the termination date of the wiretap.*

F *Inside the decrypt processor, the key components are decrypted and combined to form the device-unique key. Once the decrypt processor has that key, it can decrypt the session key in the LEAF, and then use the session key to decrypt the communications in both directions.*

\

UNIT-II

PRIVACY AND PRIVACY PROTECTING TECHNIQUES:

Choosing a Good Service Provider:

The first and most important technique for protecting your privacy is to pick service providers who respect your privacy. Here are some things to consider when you choose an ISP:
● Unless you take special measures to obscure the content and destinations of your Internet usage, your ISP can monitor every single web page that you visit, every email message that you send, every email message that you receive, and many other things about your Internet usage.
● If you have a dialup ISP, your ISP can also infer when you are at home, when you go on vacation, and other aspects of your schedule.
● If you check your email from work, your ISP can learn where you work.
● Many ISPs routinely monitor the actions of their subscribers for the purposes of testing equipment, learning about their user population, or collecting per-user demographics.

Picking a Great Password:

Passwords are the simplest form of authentication. Passwords are a secret that you share with the computer. When you log in, you type your password to prove to the computer that you are who you claim to be. The computer ensures that the password you type matches the account that you have specified. If they match, you are allowed to proceed. Using good passwords for your Internet services is a first line of defense for your privacy. If you pick a password that is easy to guess, then somebody who is targeting you will find it easier to gain access to your personal information. If you use the same password on a variety of different services, then a person who is able to discover the password for one of your services will be able to access other services.

Cleaning Up After Yourself:

When you use the Internet, you leave traces of the web sites that you visit and the information that you see on your computer. Another person can learn a lot about the web sites that you have visited by examining your computer for these electronic footprints. This process of computer examination is called *computer forensics*, and it has become a hot area of research in recent years. Special-purpose programs can also examine your computer and either prepare a report, or transmit the report over the Internet to someone else. Although it can be very hard to remove all traces of a web site that you have seen or an email message that you have downloaded, you can do a good job of cleaning up your computer with only a small amount of work. There are also a growing number of programs that can automatically clean up your computer at regular intervals.

Browser Cache:

Each time your web browser attempts to fetch a URL from the Internet, it first checks a special directory called the *browser cache* to see if it has a copy of the information that it is about to download. The browser cache can dramatically speed up the process of web browsing. For example, if you are clicking through a web site that has lots of little buttons, each of the buttons only needs to be downloaded once, even though the same buttons might appear on every page. But if another person can gain access to your computer, the browser cache will provide that person with copies of the web pages that you have visited. There are several ways to minimize the privacy impact of your browser's cache:
● You can usually configure your browser so that HTML pages and images downloaded by SSL are not cached on your hard disk. This way, confidential information such as bank statements and credit card numbers will not be locally stored. This is the default for many web browsers.
● You can configure your browser so that no HTML pages or images are cached. Although this approach will give you the maximum amount of privacy, it will also significantly decrease your browsing performance.

● You can manually delete the information in your browser's cache when you leave a particularly sensitive site. Alternatively, you can inspect the browser's cache and delete the specific material that you believe to be sensitive.

Avoiding Spam and Junk Email:

Unwanted electronic mail is the number one consumer complaint on the Internet today. A 1999 study by BrightMail, a company that develops antispam technology, found that 84 percent of Internet users had received spam; 42 percent loathed the time it takes to handle spam; 30 percent found it to be a "significant invasion of privacy;" 15 percent found it offensive; and ISPs suffered account churn rates as high as 7.2 percent as a direct result of spam.

Identity Theft:

In 1991, a car salesman from Orlando, Florida named Steven Shaw obtained the credit report for a journalist in Washington named, coincidently enough, Stephen Shaw. For Steven Shaw, getting Stephen Shaw's credit report was easier than you might think: for years, the consumer reporting firm Equifax had aggressively marketed its credit reporting service to car dealers. The service lets salespeople weed out the Sunday window-shoppers from the serious prospects by asking for a customer's name and then surreptitiously disappearing into the back room and running a quick credit check. In all likelihood, the Shaw in Florida had simply gone fishing for someone with a similar-sounding name and a good credit history.

Protecting Yourself From Identity Theft:

Identity theft is made possible because companies that grant credit-especially credit card companies-are always on the lookout for new customers, and they don't have a good way to verify the identity of a person who mails in an application or places an order over the telephone. So the credit-granting companies make a dangerous assumption: they take it for granted that if you know a person's name, address, telephone number, Social Security number, and mother's maiden name, *you must be that person*. And when the merchandise is bought and the bills aren't paid, that person is the one held responsible.

BACKUPS AND ANTITHEFT, WEB SERVER SECURITY:

Using Backups to Protect Your Data:

Backups are copies that you make of information that you hope you will never need. A backup can be a simple copy of a file that you put on a Zip disk and put away in the top drawer of your desk for safekeeping. If the original file is inadvertently deleted or corrupted, the backup can be retrieved after the damage is noticed. Backups can be very simple, like the Zip disk in your desk drawer, or they can be exceedingly complex. For example, many backup systems will let you copy every file on your computer onto a 30-gigabyte magnetic tape and create a special "restore floppy." In the event that your computer is lost or stolen, you can buy a new computer and a tape drive, put the tape into the tape drive, and boot the computer from the floppy disk; the backup system will automatically restore all of your files and applications to the newly purchased computer.

Make Backups:

Bugs, accidents, natural disasters, and attacks on your system cannot be predicted. Often, despite your best efforts, they can't be prevented. But if you have good backups, you at least won't lose your data-and in many cases, you'll be able to restore your system to a stable state. Even if you lose your entire computer-to fire, for instance-with a good set of backups you can restore the information after you purchase or borrow a replacement machine. Insurance can cover the cost of a new CPU and disk drive, but your data is something that in many cases can never be replaced.

WEB SERVER SECURITY:

PHYSICAL SECURITY OF THE SERVERS:

Surprisingly, many organizations do not consider physical security to be of the utmost concern. As an example, one New York investment house was spending tens of thousands of dollars on computer security measures to prevent break-ins during the day, only to discover that its cleaning staff was propping open the doors to the computer room at night while the floor was being mopped. A magazine in San Francisco had more than $100,000 worth of computers stolen over a holiday: an employee had used his electronic key card to unlock the building and disarm the alarm system; after getting inside, the person went to the supply closet where the alarm system was located and removed the paper log from the alarm system's printer.

The Physical Security Plan:

The first step to physically securing your installation is to formulate a written plan addressing your current physical security needs and your intended future direction. Ideally, your physical plan should be part of your site's written security policy. This plan should be reviewed by others for completeness, and it should be approved by your organization's senior management. Thus, the purpose of the plan is both planning and political buy-in. Your security plan should include:

- Descriptions of the physical assets that you are protecting
- Descriptions of the physical areas where the assets are located
- A description of your *security perimeter*- the boundary between the rest of the world and your secured area-and the holes in the perimeter
- The threats (e.g., attacks, accidents, or natural disasters) that you are protecting against and their likelihood
- Your security defenses, and ways of improving them
- The estimated cost of specific improvements
- The value of the information that you are protecting

The Disaster Recovery Plan:

You should have a plan for immediately securing temporary computer equipment and for loading your backups onto new systems in case your computer is ever stolen or damaged. This plan is known as a *disaster recovery plan.* We recommend that you do the following:

- Establish a plan for rapidly acquiring new equipment in the event of theft, fire, or equipment failure.
- Test this plan by renting (or borrowing) a computer system and trying to restore your backups.

If you ask, you may discover that your computer dealer is willing to lend you a system that is faster than the original system for the purpose of evaluation. There is probably no better way to evaluate a system than to load your backup tapes onto the system and see if they work.

Protecting Computer Hardware:

Physically protecting a computer presents many of the same problems that arise when protecting typewriters, jewelry, and file cabinets. As with a typewriter, an office computer is something that many people inside the office need to access on an ongoing basis. As with jewelry, computers are valuable and generally easy for a thief to sell. But the real danger in having a computer stolen isn't the loss of the system's hardware but the loss of the data that was stored on the computer's disks. As with legal files and financial records, if you don't have a backup-or if the backup is stolen or destroyed along with the computer-the data you have lost may well be irreplaceable. Even if you do have a backup, you will still need to spend valuable time setting up a replacement system. Finally, there is always the chance that the stolen information itself, or even the mere fact that information was stolen, will be used against you.

There are several measures that you can take to protect your computer system against physical threats. Many of them will simultaneously protect the system from dangers posed by nature, outsiders, and inside saboteurs.

- Environment
- Fire
- Smoke
- Dust
- Earthquake
- Explosion
- Temperature extremes
- Bugs (Biological)
- Electrical Noise
- Lightning
- Vibration
- Humidity
- Water
- Environmental monitoring

Host Security for Servers:

Host security is the security of the computer on which your web server is running. Traditionally, host security has been a computer security topic unto itself. Whole books (including a couple of our own) have been written on it. Many organizations that run servers on the Internet simply do not secure their servers against external attack. Other problems have gotten worse: people still pick easy-to-guess passwords, and many passwords are simply "sniffed" out of the Internet using a variety of readily available packet sniffers. And people still break into computers for the thrill, except that now many of them also steal information for financial gain or to make some ideological point.

Taxonomy of Attacks:

They are now simply one of many venues for an attacker to gain access and control over a target computer system. Many of these techniques give the attacker the ability to run code on the target machine. These techniques include:

Remote Exploits: Vulnerabilities exist in many computers that make it possible for an attacker to compromise, penetrate, or simply disable the system over the network without actually logging into the system. For example, Microsoft Windows NT 4.0 was vulnerable to the *ping of death*, which allowed anybody on the Internet to crash a Windows NT 4.0 system by simply sending the computer a specially crafted "ping" packet.

Many remote exploits are based on the *buffer overflow* technique. This technique relies on the way that the C programming language lays out information inside the computer's memory. The remote system might try to store 100 bytes into a buffer that is only set up to hold 30 or 40 bytes. The resulting information overwrites the C program's stack frame and causes machine code specified by the attacker to be executed.

Malicious programs: Another way for an attacker to compromise a system is to provide the system's users with a hostile program and wait for them to run the program. Some programs when run will install hidden services that give attackers remote access capabilities to the compromised machine; these programs are called *back doors* because they offer attackers a way into the system that bypasses conventional security measures. *Trojan horses* are programs that appear to have one function but actually have another purpose that is malicious, similar to the great wooden horse that the Greeks allegedly used to trick the Trojans and end the siege of Troy.

Viruses and worms are self-replicating programs that can travel between computers as attachments on email or independently over a network. *Viruses* modify programs on your computer, adding to them their viral *payload*. *Worms* don't modify existing programs, but they can install back doors or drop viruses on the systems they visit.

Stolen usernames and passwords and social engineering: On many computer systems it is possible to exploit bugs or other vulnerabilities to parlay ordinary access granted to normal users into "super user" or "administrative" access that is granted to system operators. One of the most common ways for an attacker to get a username and password is *social engineering*. Social engineering is one of the simplest and most effective means of gaining unauthorized access to a computer system.

Phishing: Social engineering can also be automated. There are many so-called *phishing* Programs that will send social engineering emails to thousands or tens of thousands of users at a time. Some programs solicit usernames and passwords. Others try for valid credit cards. For example, one scam is to send email to users of an online service telling them that their credit cards have expired and that they need to enter a new one at the URL that is provided. Of course, the URL goes to the attacker's web server, not to the ISP's.

Understanding Your Adversaries:

Who is breaking into networked computers with the most sophisticated of attacks? It almost doesn't matter-no matter whom the attackers may be, they all need to be guarded against.

Script kiddies: As clichéd as it may sound, in many cases the attackers are children and teenagers-people who sadly have not (yet) developed the morals or sense of responsibility that is sufficient to keep their technical skills in check. It is common to refer to young people who use sophisticated attack tools as *script kiddies*. Script kiddies should be considered a serious threat and feared for the same reason that teenagers with guns should be respected and feared. We don't call gangbangers *gun kiddies* simply because youthful gang members don't have the technical acumen to design a Colt 45 revolver or cast the steel. Instead, most people realize that teenagers with handguns should be feared even more than adults, because a teenager is less likely to understand the consequences of his actions should he pull the trigger and thus more likely to pull it.

Industrial spies: There appears to be a growing black market for information stolen from computer systems. Some individuals have tried to ransom or extort the information from its rightful owners for example, by offering to help a company close its vulnerabilities in exchange for a large cash payment. There have also been reports of attackers who have tried to sell industrial secrets to competitors of the companies that they have penetrated. Such transactions are illegal in the United States and in many other countries, but not in all.

Ideologues and national agents: There is a small but growing population of "hacktivists" who break into sites for ideologic or political reasons. Often, the intent of these people is to deface web pages to make a statement of some kind. We have seen cases of defacement of law enforcement agencies, destruction of web sites by environmental groups, and destruction of research computing sites involving animal studies. Sometimes, the protesters are making a political statement; they may be advancing an ideologic cause, or they may merely be anarchists striking a blow against technology or business.

Organized crime: The apocryphal quote by Willie Sutton about why he robbed banks, "Because that's where the money is," also applies on the Net. Vast amounts of valuable information and financial data flow through the Internet. It would be naive to believe that the criminal element is unaware of this, or is uninterested in expanding into the networked world. There have been incidents of fraud, information piracy, and money laundering conducted online that officials believe are related to organized crime. Communications on the Net have been used to advance and coordinate prostitution and pornography, gambling, trafficking in illegal substances, gun running, and other activities commonly involving organized crime. Furthermore, law enforcement sites may be targeted by criminals to discover what is known about them, or to discover identities of informants and witnesses.

Rogue employees and insurance fraud: Finally, there are many cases of tactically skilled employees who have turned against their employers out of revenge, malice, or boredom. In some cases, terminated employees have planted Trojan horses or logic bombs in their employer's computers. In other cases, computer systems have been destroyed by employees as part of insurance scams.

Tools of the Attacker's Trade: Tools that are commonly used by attackers include:

Originally written by "Hobbit," *netcat* is the Swiss Army knife for IP-based networks. As such, it is a valuable diagnostic and administrative tool as well as useful to attackers. You can use *netcat* to send arbitrary data to arbitrary TCP/IP ports on remote computers, to set up local TCP/IP servers, and to perform rudimentary port scans.

Trinoo: trinoo is the attack server that was originally written by the DoS Project. *trinoo* waits for a message from a remote system and, upon receiving the message, launches a denial-of-service attack against a third party. Versions of *trinoo* are available for most Unix operating systems, including Solaris and Red Hat Linux. The presence of *trinoo* is usually hidden. A detailed analysis of *trinoo* can be found at http://staff.washington.edu/dittrich/misc/trinoo.analysis.

Back Orifice and Netbus: These Windows-based programs are Trojan horses that allow an attacker to remotely monitor keystrokes, access files, upload and download programs, and run programs on compromised systems.

Bots: Short for robots, *bots* are small programs that are typically planted by an attacker on a collection of computers scattered around the Internet. Bots are one of the primary tools for conducting distributed denial-of-service attacks and for maintaining control on Internet Relay Chat channels. Bots can be distributed by viruses or Trojan horses. They can remain dormant for days, weeks, or years until they are activated. Bots can even engage in autonomous actions.

*root kits:*A *root kit* is a program or collection of programs that simultaneously gives the attacker superuser privileges on a computer, plants back doors on the computer, anderases any trace that the attacker has been present. Originally, root kits were designed for Unix systems (hence the name "root"), but root kits have been developed for Windows systems as well. A typical root kit might attempt a dozen or so different exploits to obtain superuser privileges. Once superuser privileges are achieved, the root kit might patch the *login* program to add a back door, then modify the computer's kernel so that any attempt to read the *login* program returns the original, unmodified program, rather than the modified one. The *netstat* command might be modified so that network connections from the attacker's machine are not displayed. Finally, the root kit might then erase the last five minutes of the computer's log files.

Securing Web Applications:

Web servers are fine programs for displaying static information such as brochures, FAQs, and product catalogs. But applications that are customized for the user or that implement business logic (such as shopping carts) require that servers be extended with specialized code that executes each time the web page is fetched. This code most often takes the form of *scripts* or *programs* that are run when a particular URL is accessed. There is no limit to what a good programming team can do with a web server, a programming language, and enough time. Unfortunately, programs that provide additional functionality over the Web can have flaws that allow attackers to compromise the system on which the web server is running. These flaws are rarely evident when the program is run as intended.

A Legacy of Extensibility and Risk: There are four primary techniques that web developers can use to create web-based applications:

CGI:
The Common Gateway Interface (CGI) was the first means of extending web servers. When a URL referencing a CGI program is requested from the web server, the web server runs the CGI program in a separate process, Captures the program's output, and sends the results to the requesting web browser. Parameters to the CGI programs are encoded as environment variables and also provided to the program on standard input.

Plug-ins, loadable modules, and Application Programmer Interfaces (APIs):
The second technique developed to extend web servers involved modifying the web server with extension modules, usually written in C or C++. The extension module was then loaded into the web server at runtime. Plug-ins, modules, and APIs are a faster way to interface custom programs to web servers because they do not require that a new process be started for each web interaction. Instead, the web server process itself runs application code within its own address space that is invoked through a documented interface. But these techniques have a distinct disadvantage: the plug-in code can be very difficult to write, and a single bug can cause the entire web server to crash.

Embedded scripting languages:
Web-based scripting languages were the third technique developed for adding programmatic functionality to web pages. These systems allow developers to place small programs, usually called scripts, directly into the web page. An interpreter embedded in the web server runs the program contained on the web page before the resulting code is sent to the web browser. Embedded scripts tend to be quite fast. Microsoft's ASP, PHP, serverside JavaScript, and *mod_perl* are all examples of embedded scripting languages.

Embedded web server:
Finally, some systems do away with the web server completely and embed their own HTTP server into the web application itself.

Rules to Code By:

Most security-related bugs in computer programs are simply that: bugs. For whatever reason, these faults keep your program from operating properly.

General Principles for Writing Secure Scripts: Over the years, we have developed a list of general principles by which to code. What follows is an excerpt from that list, edited for its particular relevance to CGI and API programs:

1. Carefully design the program before you start. Be certain that you understand what you are trying to build. Carefully consider the environment in which it will run, the input and output behavior, files used, arguments recognized, signals caught, and other aspects of behavior. List all of the errors that might occur, and how your program will deal with them. Write a code specification in English (or your native language) before writing the code in the computer language of your choice.

2. Show the specification to another person. Before you start writing code, show the specification that you have written to another programmer. Make sure they can understand the specification and that they think it will work. If you can't convince another programmer that your paper design will work, you should go back to the design phase and make your specification clearer. The time you spend now will be repaid many times over in the future.

3. Write and test small sections at a time. As you start to write your program, start small and test frequently. When you test your sections, test them with both expected data and unexpected data. Where practical, functions should validate their arguments and perform reasonable actions (such as exiting with an error message or returning an error code) when presented with unreasonable data. A large number of security-related programs are simply bugs that have exploitable consequences. By writing code that is more reliable, you will also be writing code that is more secure.

4. Check all values provided by the user. An astonishing number of security-related bugs arise because an attacker sends an unexpected value or an unanticipated format to a program or a function within a program. A simple way to avoid these types of problems is by having your scripts always check and validate all of their arguments. Argument checking will not noticeably slow your scripts, but it will make them less susceptible to hostile users. As an added benefit, argument checking and error reporting will make the process of catching non security-related bugs easier.

Securely Using Fields, Hidden Fields, and Cookies: One of the reasons that it can be difficult to develop secure web applications has to do with the very architecture of web applications. When you develop an application, you generally write a body of code that runs locally on the web server and a much smaller body of code that is downloaded and run remotely on the user's web browser. You might spend a lot of time making sure that these two code bases work properly together. For example, it's very important to make sure that the field names downloaded in web forms exactly match the field names that server-side scripts are expecting. And you will probably spend time making sure that the HTML forms, JavaScript, and other codes that are downloaded to the browser work properly on a wide range of different browser programs.

Using Fields Securely: When checking arguments in your program, pay special attention to the following:

● Filter the contents of every field, selecting the characters that are appropriate for each response. For example, if a field is supposed to be a credit card number, select out the characters 0-9 and leave all other characters behind. This will also allow people to enter their credit card numbers with spaces or dashes.
● After you filter, check the length of every argument. If the length is incorrect, do not proceed, but instead generate an error.
● If you use a selection list, make certain that the value provided by the user was one of the legal values. Attackers can provide any value that they wish: they are not constrained by the allowable values in the selection list.
● Even if your forms use JavaScript to validate the contents of a form before it is submitted, be sure that you revalidate the contents on the server. An attacker can easily turn off JavaScript or bypass it entirely.

Hidden Fields and Compound URLs: A *hidden field* is a field that the web server sends to the web browser that is not displayed on the user's web page. Instead, the field merely sits in the browser's memory. When the form on the page is sent back to the server, the field and its contents are sent back. Some web developers use hidden fields to store information that is used for session tracking on e-commerce systems. For example, instead of using HTTP Basic Authentication, developers sometimes embed the username and password provided by the user as hidden fields in all future form entries:

```
<INPUT TYPE="hidden" NAME="username" VALUE="simsong">
<INPUT TYPE="hidden" NAME="password" VALUE="myauth11">
```

Hidden fields can also be used to implement a shopping cart:

```
<INPUT TYPE="hidden" NAME="items" VALUE="3">
<INPUT TYPE="hidden" NAME="item1" VALUE="Book of Secrets:$4.99">
<INPUT TYPE="hidden" NAME="item2" VALUE="Nasty Software:$45.32">
<INPUT TYPE="hidden" NAME="item3" VALUE="Helping Hand:$32.23">
```

Instead of embedding this information in hidden fields, it can be placed directly in the URL. These URLs will then be interpreted as if they were forms that were posted using the HTTP GET protocol. For example, this URL embeds a username and password:

```
http://www.vineyard.net/cgi-bin/password_tester?username=simsong&password=myauth11
```

Using Cookies: One attractive alternative to using hidden fields or URLs is to store information such as usernames, passwords, shopping cart contents, and so on, in HTTP cookies.

Users can modify their cookies, so cookies used for user tracking, shopping carts, and other types of e-commerce applications have all of the same problems described for hidden fields or compound URLs. But cookies also have problems all their own, including:

● Old cookies may continue to be used, even after they have "expired."
● Users may make long-term copies of cookies that are supposed to remain ephemeral and not ever be copied onto a hard drive.
● Some users are suspicious of cookies and simply turn off the feature.

Using Cryptography to Strengthen Hidden Fields, Compound URLs, and Cookies:

Many of the problems discussed in this section can be solved by using cryptography to protect the information in hidden fields, compound URLs, and cookies. Cryptography can:
● Prevent users from understanding the information stored on their computer.
● Allow web server applications to detect unauthorized or accidental changes to this information.

Here are the three examples from the previous sections, recoded to use cryptography.

Username and password authentication:
```
<INPUT TYPE="hidden" NAME="auth"
VALUE="p6e6J6FwQOk0tqLFTFYq5EXR03GQ1wYWG0ZsVnk09yv7ItIHG17ymls4UM%2F1bw
HygRhp7ECawzUm %0AKl3Q%2BKRYhlmGILFtbde8%0A:">
```
A secure shopping cart:
```
<INPUT TYPE="hidden" NAME="cart"
VALUE="fLkrNxpQ9GKv9%2FrAvnLhuLnNDAV50KhNPjPhqG6fMJoJ5kCQ5u1gh0ij8JBqphBx
dGVNOdja41XJ%0APLsT%2Bt1kydWN4Q%2BO9pW0yR9eIPLrzaDsZxauNPEe7cymPmXwd%2
B6c1L49uTwdNTKoS0XAThDzow%3D%3D%0A:">
```
A compound URL:
```
http://www.vineyard.net/cgi-bin/password_
tester?p6e6J6FwQOk0tqLFTFYq5EXR03GQ1wYWG0ZsVnk09yv7ItIHG17ymls4UM%2F1bwHyg
Rhp7ECawzUm%0AKl3Q%2BKRYhlmGILFtbde8%0A:
```

Rules for Programming Languages: This section provides rules for making programs written in specific programming languages more secure.

Rules for Perl: Here are some rules to follow to make your Perl scripts more secure:

1. Use Perl's tainting features for all CGI programs. These features are invoked by placing the "-T" option at the beginning of your Perl script. Perl's tainting features make it more suited than C to CGI programming. When enabled, tainting marks all variables that are supplied by users as tainted." Variables whose values are dependent on tainted variables are themselves tainted as well. Tainted values cannot be used to open files or for system calls. Untainted information can only be extracted from a tainted variable by the use of Perl's string match operations.

2. The tainting feature also requires that you set the PATH environment variable to a known "safe value" before allowing your program to invoke the *system( )* call.

3. Remember that Perl ignores tainting for filenames that are opened read-only. Nevertheless, be sure that you untaint all filenames, and not simply filenames that are used for writing.

Rules for C: It is substantially harder to write secure programs in C than it is in the Perl programming language; Perl has automatic memory management whereas C does not. Furthermore, because of the lack of facilities for dealing with large programs, Perl program sources tend to be smaller and more modular than their C counterparts.Nevertheless, one very important reason to write CGI programs in C remains: speed. Each time a CGI program is run, the program must be loaded into memory and executed. If your CGI program is written in Perl, the entire Perl interpreter must be loaded and the Perl program compiled before the

CGI program can run. The overhead from these two operations dwarfs the time required by the CGI program itself. The overhead of loading Perl can be eliminated by using the Apache Perl module or Microsoft's Internet Information Server *perl* script.

Rules for the Unix Shell: Don't write CGI scripts with the Unix shells (*sh*, *csh*, *ksh*, *bash*, or *tcsh*) for anything but the most trivial script. It's too easy to make a mistake, and there are many lurking security problems with these languages.

Using PHP Securely: PHP is a widely used and loved server-side scripting language for building web pages. Originally called Personal Home Page, and then PHP3, PHP now stands for PHP Hypertext Preprocessor. The web site for PHP development is http://www.php.org/. PHP is an official project of The Apache Foundation.

● It is easy to use and very fast. Even though PHP scripts are interpreted at runtime, the interpreter is built into the web server. As a result, PHP pages can run significantly faster (more than 10 times faster is not uncommon) than the equivalent Perl/CGI web pages.
Unlike CGI scripts, PHP pages do not need to be made "executable" or placed in special directories to run: if PHP is enabled in the web server, all you need to do is to give an HTML file a *.php* or *.php3* extension and the PHP system will automatically run.
● The PHP interpreter shows errors directly on the web page, not in a log file.
● PHP can cache connections to the MySQL database system. As a result, PHP pages that are fed from information in a database can display dramatically faster than database-driven pages using other systems.
● PHP is extremely powerful. Scripts written in PHP can open files, open network connections, and execute other programs.

Writing Scripts That Run with Additional Privileges: Many scripts need to run with user permissions different from those of the web server itself. On a Unix computer, the easiest way to do this is to make the script SUID or SGID. By doing this, the script runs with the permissions of the owner of the file, rather than those of the web server itself. On Macintosh, DOS, and Windows 95 systems, there is no such choice-scripts run with the same privileges and can access everything on the system.

Unfortunately, programs that run with additional privileges traditionally have been a source of security problems. This list of suggestions is based on those problems and specially tailored for the problems faced by the web developer:
1. Avoid using the superuser (SUID root or SGID *wheel*) unless it is vital that your program perform actions that can only be performed by the superuser. For example, you will need to use SUID root if you want your program to modify system databases such as */etc/passwd*. But if you merely wish to have the program access a restricted database of your own creation, create a special Unix user for that application and have your scripts SUID to that user.

2. If your program needs to perform some functions as superuser but generally does not require SUID permissions, consider putting the SUID part in a different program and constructing a carefully controlled and monitored interface between the two.

3. If you need SUID or SGID permissions, use them for their intended purpose as early in the program as possible and then revoke them by returning the effective and real UIDs and GIDs to those of the process that invoked the program.

4. Avoid writing SUID scripts in shell languages, especially in *csh* or its derivatives.

5. Consider creating a different username or group for each application to prevent unexpected interactions and amplification of abuse.

Connecting to Databases: It is common for a CGI program or script to connect to databases that are external to the web server. External databases can be used for many purposes, such as storing user preferences, implementing shopping carts, and even order processing. When the script runs, it opens a connection to the database, issues a query, gets the result, and then uses the result to formulate a response to the user. On some systems, a new database connection is created each time a new script is run. Other systems maintain a small number of persistent connections which are cached.

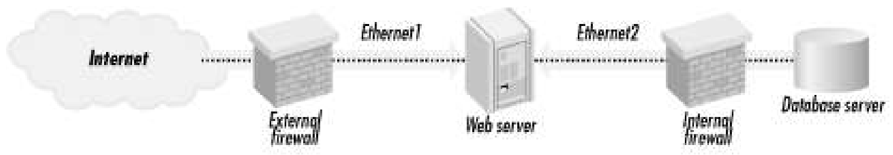
Database-backed web sites give a tremendous amount of power and flexibility to the web designer. Unfortunately, this approach can also reduce the overall security of the system: many of the security incidents mentioned earlier in this book happened because an attacker was able to execute arbitrary SQL commands on the database server and view the results. (For example, recall the story of the attacker who was able to obtain tens of thousands of credit card numbers from an e-commerce site.) If you decide to deploy a database server to supplement your web site, it is important to be sure that the server will be deployed and used securely.

Protect Account Information: Before the database server provides results to the script running on the web server, the server needs to authenticate the script to make sure it is authorized to access the information. Most databases use a simple username/password for account authentication, which means the script needs to have a valid username/password and present this information to the database server each time a request is issued.

Use Filtering and Quoting to Screen Out Raw SQL": As we mentioned earlier in this chapter, it is extremely important to filter all data from the user to make sure that it contains only allowable characters. When working with SQL servers, it is further important to properly quote data provided by the user before sending the data to the server. These procedures are used to prevent users from constructing their own SQL commands and sending that data to the SQL server.

Protect the Database Itself: Finally, it is important that you protect the database server itself:
● Configure your firewall or network topology so that it is impossible for people outside your organization to access your database server. For example, you may wish to set up your web server with two Ethernet adapters-one that connects to the Internet, and one that connects to a small firewall appliance that, in turn, connects to your database server. The firewall should be set up so that only database queries can pass between the web server and the database server.
● Make sure logins on the database server are limited. The individuals with login capabilities should be the system administrator and the database administrator.
● Make sure that the database server is backed up, physically protected, and maintained in the same way as your other secure servers.

Connecting a database server and a web server to the Internet and your internal network with multiple firewalls:



UNIT – III

Recent Advances in Access Control:

Information plays an important role in any organization and its protection against unauthorized disclosure (secrecy) and unauthorized or improper modifications (integrity), while ensuring its availability to legitimate users (no denials-of-service) is becoming of paramount importance. An important service in guaranteeing information protection is the access control service. Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. An access control system can be considered at three different abstractions of control: access control policy, access control model, and access control mechanism. A policy defines the high level rules used to verify whether an access request is to be granted or denied. A policy is then formalized through a security model and is enforced by an access control mechanism.

An example of access matrix

	Document1	Document2	Program1	Program2
Ann	read, write	read	execute	
Bob	read	read	read, execute	
Carol		read, write		read, execute
David			read, write, execute	read, write, execute

The variety and complexity of the protection requirements that may need to be imposed in today's systems makes the definition of access control policies a far from trivial process. An access control system should be simple and expressive. It should be simple to make easy the management task of specifying and maintaining the security specifications. It should be expressive to make it possible to specify in a flexible way different protection requirements that may need to be imposed on different resources and data. Moreover, an access control system should include support for the following features.

• Policy combination. Since information may not be under the control of a single authority, access control policies information may take into consideration the protection requirements of the owner, but also the requirements of the collector and of other parties. These multiple authorities' scenario should be supported from the administration point of view providing solutions for modular, large-scale, scalable policy composition and interaction.
• Anonymity. Many services do not need to know the real identity of a user. It is then necessary to make access control decisions dependent on the requester's attributes, which are usually proved by digital certificates.
• Data outsourcing. A recent trend in the information technology area is represented by data outsourcing, according to which companies shifted from fully local management to outsourcing the administration of their data by using externally service providers. Here, an interesting research challenge consists in developing an efficient mechanism for implementing selective access to the remote data.

Classical Access Control Models:

Classical access control models can be grouped into three main classes: discretionary access control (DAC), which bases access decisions on users' identity; mandatory access control (MAC), which bases access decisions on mandated regulations defined by a central authority; and role-based access control (RBAC), which bases access decisions on the roles played by users in the models. We now briefly present the main characteristics of these classical access control models.

Discretionary Access Control:

Discretionary access control is based on the identity of the user requesting access and on a set of rules, called authorizations, explicitly stating which user can perform which action on which resource. In the most basic form, an authorization is a triple (s, o, a), stating that user s can execute action a on object o. The first discretionary access control model proposed in the literature is the access matrix model [4, 5, 6]. Let S, O, and A be a set of subjects, objects, and actions, respectively. The access matrix model represents the set of authorizations through a |S|×|O| matrix A. Each entry A[s, o] contains the list of actions that subject s can execute over object o.

The access matrix model can be implemented through different mechanisms. The straightforward solution exploiting a two-dimensional array is not viable, since A is usually sparse. The mechanisms typically adopted are:
• Authorization table. The non empty entries of A are stored in a table with three attributes: user, action, and object.
• Access control list (ACL). The access matrix is stored by column, that is, each object is associated with a list of subjects together with a set of actions they can perform on the object.
• Capability. The access matrix is stored by row, that is, each subject is associated with a list indicating, for each object, the set of actions the subject can perform on it.

User	Action	Object
Ann	read	Document1
Ann	write	Document1
Ann	read	Document2
Ann	execute	Program1
Bob	read	Document1
Bob	read	Document2
Bob	read	Program1
Bob	execute	Program1
Carol	read	Document2
Carol	write	Document2
Carol	execute	Program2
David	read	Program1
David	write	Program1
David	execute	Program1
David	read	Program2
David	write	Program2
David	execute	Program2

(a)

From the access matrix model, discretionary access control systems have evolved and they include support for the following features.
• Conditions. To make authorization validity depend on the satisfaction of some specific constraints, today's access control systems typically support conditions associated with authorizations. For instance, conditions impose restrictions on the basis of: object content (content-dependent

conditions), system predicates (system-dependent conditions), or accesses previously executed (history-dependent conditions).
•Abstractions. To simplify the authorization definition process, discretionary access control supports also user groups and classes of objects, which may also be hierarchically organized. Typically, authorizations specified on an abstraction propagate to all its members according to different propagation policies. Here, for example, an authorization specified for the Nurse group applies also to Bob and Carol.
• Exceptions. The definition of abstractions naturally leads to the need of supporting exceptions in authorization definition. Suppose, for example, that all users belonging to a group but u can access resource r. If exceptions were not supported, it would be necessary to associate an authorization with each user in the group but u, therefore not exploiting the possibility of specifying the authorization of the group. This situation can be easily solved by supporting both positive and negative authorizations: the system would have a positive authorization for the group and a negative authorization for u.

The introduction of both positive and negative authorizations brings to two problems: inconsistency, when conflicting authorizations are associated with the same element in a hierarchy; and incompleteness, when some accesses are neither authorized nor denied.

Incompleteness is usually easily solved by assuming a default policy, open or closed (this latter being more common), where no authorization applies. In this case, an open policy approach allows the access, while the closed policy approach denies it.

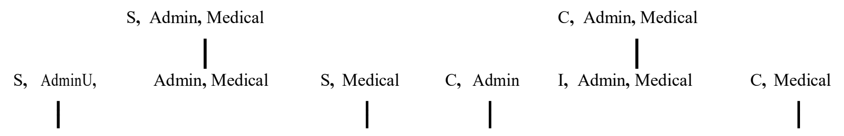
To solve the inconsistency problem, different conflict resolution policies have been proposed such as:
– *No conflict*. The presence of a conflict is considered an error.
– *Denials take precedence*. Negative authorizations take precedence.
– *Permissions take precedence*. Positive authorizations take precedence.
– *Nothing takes precedence*. Conflicts remain unsolved.
– *Most specific takes precedence*. An authorization associated with an element n overrides a contradicting authorization (i.e., an authorization with the same subject, object, and action but with a different sign) associated with an ancestor of n for all the descendants of n. For instance, consider the user-group hierarchy.

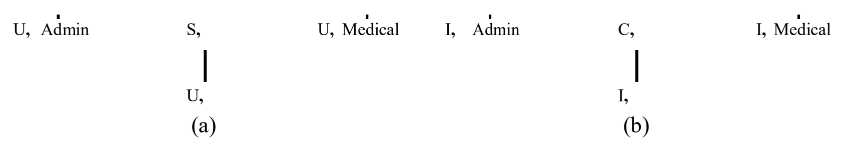
Mandatory Access Control:

Mandatory security policies enforce access control on the basis of regulations mandated by a central authority. The most common form of mandatory policy is the multilevel security policy, based on the classifications of subjects and objects in the system. Each subject and object in the system is associated with an access class, usually composed of a security level and a set of categories. Security levels in the system are characterized by a total order relation, while categories form an unordered set. As a consequence, the set of access classes is characterized by a partial order relation, denoted ≥ and called dominance.

Given two access classes c1 and c2, c1 dominates c2, denoted c1 ≥ c2, iff the security level of c1 is greater than or equal to the security level of c2 and the set of categories of c1 includes the set of categories of c2. Access classes together with their partial order dominance relationship form a lattice. Mandatory policies can be classified as secrecy-based and integrity-based, operating in a dual manner.

Mandatory policies can be classified as secrecy-based and integrity-based, operating in a dual manner.





Secrecy-Based Mandatory Policy. The main goal of secrecy based mandatory policies is to protect data confidentiality. As a consequence, the security level of the access class associated with an object reflects the sensitivity of its content, while the security level of the access class associated with a subject, called clearance, reflects the degree of trust placed in the subject not to reveal sensitive information. The set of categories associated with both subjects and objects defines the area of competence of users and data. A user can connect to the system using her clearance or any access class dominated by her clearance. A process generated by a user connected with a specific access class has the same access class as the user.

The access requests submitted by a subject are evaluated by applying the following two principles.
No-Read-Up: A subject s can read an object o if and only if the access class of the subject dominates the access class of the object.
No-Write-Down: A subject s can write an object o if and only if the access class of the object dominates the access class of the subject.

Role-Based Access Control:

A third approach for access control is represented by Role-Based Access Control (RBAC) models. A role is defined as a set of privileges that any user playing that role is associated with. When accessing the system, each user has to specify the role she wishes to play and, if she is granted to play that role, she can exploit the corresponding privileges. The access control policy is then defined through two different steps: first the administrator defines roles and the privileges related to each of them; second, each user is assigned with the set of roles she can play. Roles can be hierarchically organized to exploit the propagation of access control privileges along the hierarchy.

A user may be allowed to simultaneously play more than one role and more users may simultaneously play the same role, even if restrictions on their number may be imposed by the security administrator. It is important to note that roles and groups of users are two different concepts. A group is a named collection of users and possibly other groups, and a role is a named collection of privileges, and possibly other roles. Furthermore, while roles can be activated and deactivated directly by users at their discretion, the membership in a group cannot be deactivated.

The main advantage of RBAC, with respect to DAC and MAC, is that it better suits to commercial environments. In fact, in a company, it is not important the identity of a person for her access to the system, but her responsibilities. Also, the role-based policy tries to organize privileges mapping the organization's structure on the roles hierarchy used for access control.

Credential-Based Access Control:

In an open and dynamic scenario, parties may be unknown to each other and the traditional separation between authentication and access control cannot be applied anymore. Such parties can also play the role of both client, when requesting access to a resource, and server for the resources it makes available for other users in the system. Advanced access control solutions should then allow to decide, on one hand, which requester (client) is to be granted access to the resource, and, on the other hand, which server is qualified for providing the same resource. Trust management has been developed as a solution for supporting access control in open environments [19]. The first approaches proposing a trust management solution for access control are PolicyMaker and KeyNote.

The key idea of these proposals is to bind public keys to authorizations and to use credentials to describe specific delegations of trust among keys. The great disadvantage of these early solutions is that they assign authorizations directly to users' keys. The authorization specification is then difficult to manage and, moreover, the public key of a user may act as a pseudonym of her, thus reducing the advantages of trust management, where the identity of the users should not be considered. The problem of assigning authorizations directly to keys has been solved by the introduction of digital certificates.

A digital certificate is the on-line counterpart of paper credentials (e.g., a driver license). A digital certificate is a statement, certified by a trusted entity (the certificate authority), declaring a set of properties of the certificate's holder (e.g., identity, accreditation, or authorizations). Access control models, by exploiting digital certificates for granting or denying access to resources, make access decisions on the basis of a set of properties that the requester should have. The final user can prove to have such properties by providing one or more digital certificates

.

The development and effective use of credential-based access control models require however tackling several problems related to credential management and disclosure strategies, delegation and revocation of credentials, and establishment of credential chains. In particular, when developing an access control system based on credentials, the following issues need to be carefully considered.

- Ontologies. Since there is a variety of security attributes and requirements that may need to be considered, it is important to guarantee that different parties will be able to understand each other, by defining a set of common languages, dictionaries, and Ontologies.
- Client-side and server-side restrictions. Since parties may act as either a client or a server, access control rules need to be defined both client-side and server-side.
- Credential-based access control rules. New access control languages supporting credentials need to be developed. These languages should be both expressive (to define different kinds of policies) and simple (to facilitate policy definition).
- Access control evaluation outcome. The resource requester may not be aware of the attributes she needs to gain access to the requested resource. As a consequence, access control mechanisms should not simply return a permit or deny answer, but should be able to ask the final user for the needed credentials to access the resource.
- Trust negotiation strategies. Due to the large number of possible alternative credentials that would enable an access request, a server cannot formulate a request for all these credentials, since the client may not be willing tore lease the whole set of her credentials. On the other hand, the server should not disclose too much of the underlying security policy, since it may contain sensitive information. In the following, we briefly describe some proposals that have been developed for trust negotiation and for regulating service access in open environments.

Policy Composition:

In many real word scenarios, access control enforcement needs to take into consideration different policies independently stated by different administrative subjects, which must be enforced as if they were a single policy. As an example of policy composition, consider an hospital, where the global policy may be obtained by combining together the policies of its different wards and the externally imposed constraints (e.g., privacy regulations). Policy composition is becoming of paramount importance in all those contexts in which administrative tasks are managed by different, non collaborating, entities.

Policy composition is an orthogonal aspect with respect to policy models, mechanisms, and languages. As a matter of fact, the entities expressing the policies to be composed may even not be aware of the access control system adopted by the other entities specifying access control rules. The main desiderata for a policy composition framework can be summarized as follows.

Heterogeneous policy support: The framework should support policies expressed in different languages and enforced by different mechanisms.

- Support of unknown policies. The framework should support policies that are not fully defined or are not fully known when the composition strategy is defined. Consequently, policies are to be treated as black-boxes and are supposed to return a correct and complete response when queried at access control time.
- Controlled interference. The framework cannot simply merge the sets of rules defined by the different administrative entities, since this behavior may cause side effects. For instance, the accesses granted/denied might not correctly reflect the specifications anymore.
- Expressiveness. The framework should support a number of different ways for combining the input policies, without changing the input set of rules or introducing ad-hoc extensions to authorizations.
- Support of different abstraction levels. The composition should highlight the different components and their interplay at different levels of abstraction.
- Formal semantics. The language for policy composition adopted by the framework should be declarative, implementation independent, and based on a formal semantic to avoid ambiguity.

Access Control Models for XML:

Introduction:

The amount of information that is made available and exchanged on the Web sites is continuously increasing. A large portion of this information (e.g., data exchanged during EC transactions) is sensitive and needs to be protected. However, granting security requirements through HTML-based information processing turns out to be rather awkward, due to HTML's inherent limitations. HTML provides no clean separation between the structure and the layout of a document and some of its content is only used to specify the document layout. Moreover, site designers often prepare HTML pages according to the needs of a particular browser. Therefore, HTML markup has generally little to do with data semantics.

To the aim of separating data that need to be represented from how they are displayed, the World Wide Web Consortium (W3C) has standardized a new markup language: the eXtensible Markup Language (XML) [1]. XML is a markup meta-language providing semantics-aware markup without losing the formatting and rendering capabilities of HTML. XML's tags' capability of self-description is shifting the focus of Web communication from conventional hypertext to data interchange. Although HTML was defined using only a small and basic part of SGML (Standard Generalized Markup Language: ISO 8879), XML is a sophisticated subset of SGML, designed to describe data using arbitrary tags. As its name implies, extensibility is a key feature of XML; users and applications are free to declare and use their own tags and attributes.

Therefore, XML ensures that both the logical structure and content of semantically rich information is retained. XML focuses on the description of information structure and content as opposed to its presentation. Presentation issues are addressed by a separate language, XSL [2] (XML Style sheet Language), which is also a W3C standard for expressing how XML-based data should be rendered.

Since XML documents can be used instead of traditional relational databases for data storage and organization, it is necessary to think of a security system for XML documents protection. In this chapter, we will focus on access control enforcement. Specifically, in the literature, different access control models have been proposed for protecting data stored in XML documents, exploiting the flexibility offered by the markup language. Even if traditionally access control models can be applied to XML documents, by simply treating them as files, a finer grained access control system is frequently necessary. As a matter of fact, an XML document may contain both sensitive and publicly available information, and it is necessary to distinguish between them when specifying the access control policy. The remainder of the chapter is organized as follows. Section 2 discusses the basic XML concepts, by introducing DTD, XML Schema, XPath and XQuery syntax and semantics. Section 3 introduces the problem of access control for XML documents, points out the characteristics that an access control model for XML documents should have. Section 4 illustrates in the details two of the first access control models proposed for XML documents, and briefly describes other proposals. Finally, Sect. 5 concludes the chapter.

Preliminary Concepts:

XML (eXtensible Markup Language) is a markup language developed by the World Wide Web Consortium (W3C) and used for describing semi structured information. We introduce some of the most important concepts related to XML, which are useful to define an access control system for protecting XML documents.

Well-Formed and Valid XML Documents: XML document is composed of a sequence of (possibly nested) **elements** and **attributes** associated with them. Basically, elements are delimited by a pair of start and end tags (e.g., <request> and </request>) or, if they have no content, are composed of an empty tag (e.g., <request/>). Attributes represent properties of elements and are included in the start tag of the el-ement with which they are associated (e.g., <request number="10">). An XML document is said to be **well-formed** if its syntax complies with the rules defined by the W3C consortium [1], which can be summarized as follows:

- the document must start with the **prologue** <?xml version="1.0"?>;
- the document must have a **root** element, containing all other elements in the document;
- all open tags must have a corresponding closed tag, provided it is not an empty tag;
- elements must be properly nested;
- tags are case-sensitive;
- attribute values must be quoted.

An **XML language** is a set of XML documents that are characterized by a syntax, which describes the markup tags that the language uses and how they can be combined, together with its semantics. A **schema** is a formal definition of the syntax of an XML language, and is usually expressed through a schema language. The most common schema languages, and on which we focus our attention, are **DTD** and **XML Schema**, both originating from W3C.

Document Type Definition.

A DTD document may be either internal or external to an XML document and it is not itself written in the XML notation.

A DTD schema consists of definition of elements, attributes, and other constructs. An element declaration is of the form <!ELEMENT **element name content**>, where **element name** is an element name and **content** is the de-scription of the content of an element and can assume one of the following alternatives:

o the element contains parsable character data (#PCDATA);
p the element has no content (Empty);
q the element may have any content (Any);
r the element contains a group of one or more subelements, which in turn may be composed of other subelements;
s the element contains parsable character data, interleaved with subele-ments.

When an element contains other elements (i.e., subelements or mixed con-tent), it is necessary to declare the subelements composing it and their organi-zation. Specifically, sequences of elements are separated by a comma "," and alternative elements are separated by a vertical bar " ". Declarations of se-quence and choices of subelements need to describe subelements' cardinality. With a notation inspired by extended BNF grammars, "*" indicates zero or more occurrences, "+" indicates one or more occurrences, "?" indicates zero or one occurrence, and no label indicates exactly one occurrence.

An attribute declaration is of the form <!ATTLIST **element name at-tribute def** >, where **element name** is the name of an element, and **attribute def** is a list of attribute definitions that, for each attribute, specify the at-tribute name, type, and possibly default value. Attributes can be marked as #REQUIRED, meaning that they must have an explicit value for each occur-rence of the elements with which they are associated; #IMPLIED, meaning that they are optional; #FIXED, meaning that they have a fixed value, indicated in the definition itself.

An XML document is said to be **valid** with respect to a DTD if it is syntactically correct according to the DTD. Note that, since elements and attributes defined in a DTD may appear in an XML document zero (optional elements), one, or multiple times, depending on their cardinality constraints, the structure specified by the DTD is not rigid; two distinct XML documents of the same schema may diﬀer in the number and structure of elements.

XML Schema:

An XML Schema is an XML document that, with respect to DTD, has a number of advantages. First, an XML Schema is itself an XML document, consequently it can be easily extended for future needs. Furthermore, XML Schemas are richer and more powerful than DTDs, since they provide support for data types and namespaces, which are two of the most significant issues with DTD.

An element declaration specifies an element name together with a simple or complex type. A simple type is a set of Unicode strings (e.g., decimal, string, float, and so on) and a complex type is a collection of requirements for attributes, subelements, and character data that apply to the elements assigned to that type. Such requirements specify, for example, the order in which subelements must appear, and the cardinality of each subelement (in terms of maxOccurs and minOccurs, with 1 as default value). Attribute declarations specify the attributes associated with each element and indicate attribute name and type. Attribute declarations may also specify either a default value or a fixed value. Attributes can be marked as: required, meaning that they must have an explicit value for each occurrence of the elements with which they are associated; optional, meaning that they are not necessary.

Example 1. Suppose that we need to define an XML-based language for describing bank account operations. Figure 1(a) illustrates a DTD stating that each account operation contains a request element and one or more operation elements. Each account operation is also characterized by two mandatory attributes: bankAccN, indicating the number of the bank account of the requester; and id, identifying the single update. Each request element is composed of date, means, and notes elements, where only date is required. Element operation is instead composed of: type, amount, recipient, and possibly one between notes and value.

DTDs and XML documents can be graphically represented as trees. A DTD is represented as a labeled tree containing a node for each element, attribute, and value associated with fixed attributes. To distinguish elements and attributes in the graphical representation, elements are represented as ovals, while attributes as rectangles. There is an arc in the tree connecting each element with all the elements/attributes belonging to it, and between each #FIXED attribute and its value. Arcs connecting an element with its subelements are labeled with the cardinality of the relationship. Arcs labeled or and with multiple branching are used to represent a choice in an element declaration (|). An arc with multiple branching is also used to represent a sequence with a cardinality constraint associated with the whole sequence (?, +, *). To preserve the order between elements in a sequence, for any two elements ei and ej, if ej follows ei in the element declaration, node ej appears below node ei in the tree. Each XML document is represented by a tree with a node for each element, attribute, and value in the document. There is an arc between each element and each of its subelements/attributes/values and between each attribute and each of its value(s). Each arc in the DTD tree may correspond to zero, one, or multiple arcs in the XML document, depending on the cardinality of the corresponding containment relationship. Note that arcs in XML documents are not labeled, as there is no further information that needs representation.

Elements and Attributes Identification:

The majority of the access control models for XML documents identify the objects under protection (i.e., elements and attributes) through the XPath language [3]. XPath is an expression language, where the basic building blockis the path expression.

A path expression on a document tree is a sequence of element names or predefined functions separated by character / (slash): l1/l2/ . . . /ln. Path expressions may terminate with an attribute name as the last term of the sequence. Attribute names are syntactically distinguished by preceding them with special character @.

XPath allows the association of conditions with nodes in a path; in this case the path expression identifies the set of nodes that satisfy all the conditions. Conditional expressions in XPath may operate on the "text" of elements (i.e., character data in elements) or on names and values of attributes. A condition is represented by enclosing it within square brackets, following a label li in a path expression l1/l2/ . . . /ln. The condition is composed of one or more predicates, which may be combined via and and or boolean operators. Each predicate compares the result of the evaluation of the relative path expression (evaluated at li) with a constant or with another expression. Multiple conditional expressions appearing in the same path expression are considered to be anded (i.e., all the conditions must be satisfied). In addition, conditional expressions may include functions last() and position() that permit the extraction of the children of a node that are in given positions. Function last() evaluates to true on the last child of the current node. Function position () evaluates to true on the node in the evaluation context whose position is equal to the context position.

Path expressions are also the building blocks of other languages, such as XQuery [4] that allows to make queries on XML documents through FLWOR expressions. A FLOWR expression is composed of the following clauses:

• FOR declares variables that are iteratively associated with elements in the XML documents, which are identified via path expressions;
• LET declares variables associated with the result of a path expression; • WHERE imposes conditions on tuples;
• ORDER BY orders the result obtained by FOR and LET clauses;
• RETURN generates the final result returned to the requester.
Example 2. Consider the DTD and the XML document in Example 1. Some examples of path expressions are the following.
• /account operation/operation: returns the content of the operation element, child of account operation;
• /account operation/@bankAccN: returns attribute bankAccN of element account operation;
• /account operation//notes: returns the content of the notes elements, anywhere in the sub tree rooted at account operation; in this case, it returns both /account operation/request/notes and /account operation/operation/notes;
• /account operation/operation[./type="bank transfer"]: returns the content of the operation element, child of account operation, only if the type element, child of operation, has value equal to "bank transfer".

The following XQuery extracts from the XML document in Fig. 1(b) all the account operation elements with operation type equal to "bank transfer". For the selected elements, the amount and the recipient of the operation are returned, along with all notes appearing in the selected account operation element.

```
<BankTransf>
{ FOR $r in document("update account")/account operation
WHERE $r/operation/type="bank transfer"
RETURN $r/operation/amount, $r/operation/recipient, $r//notes
}
</BankTransf>
```

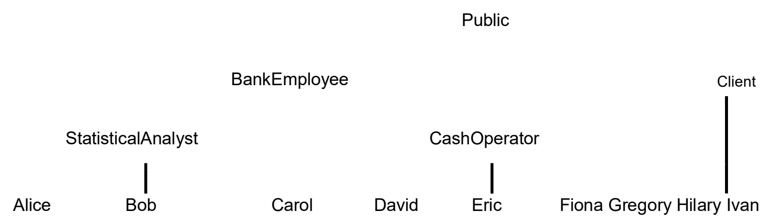
XML Access Control Requirements:

Due to the peculiar characteristics of the XML documents, they cannot be protected by simply adopting traditional access control models, and specific models need to be defined. By analyzing the existing proposals, it is easy to see that they are all based on the definition of a set of authorizations that at least specify the subjects on which they apply the objects to be protected, and the action to be executed. The existing XML-based access control models differentiate on the basis of the subjects, objects, and actions they can support for access control specification and enforcement.

Subject: Subjects are usually referred to on the basis of their identities or of the network location from which requests originate. Locations can be expressed with reference to either the numeric IP address (e.g., 150.100.30.8) or the symbolic name (e.g., bank.com) from which the request comes. It often happens that the same privilege should be granted to sets of subjects, which share common characteristics, such as the department where they work, or the role played in the company where they work. To the aim of simplifying the authorizations definition, some access control models allow the specification of authorizations having as subject:

• a group of users, which is a statically defined set of users; groups can be nested and overlapping;
• a location pattern, which is an expression identifying a set of physical locations, obtained by using the wild character * in physical or symbolic addresses;
• a role, which is a set of privileges that can be exploited by any user while playing the specific role; users can dynamically decide which role to play, among the ones they are authorized to play. Also, subjects are often organized in hierarchies, where an authorization defined for a general subject propagates to its descendants.

An example of user-group hierarchy:



A hierarchy can be pictured as a directed acyclic graph containing a node for each element in the hierarchy and an arc from element x to element y, if x directly dominates y. Dominance relationships holding in the hierarchy correspond to paths in the graph. Figure 3 shows an example of user-group hierarchy.

Object Granularity: The identification of the object involved in a specific authorization can exploit the possibility given by XML of identifying elements and attributes within a document through path expressions as defined by the XPath language.

Action: Most of the proposed XML access control models support only read operations, since there is not a standard language for XML update. Furthermore, the management of write privileges is a difficult task, which needs to take into account both the access control policy and the DTD (or XML Schema) defined for the document. In fact, the DTD may be partially hidden to the user accessing the document, as some elements/ attributes may be denied by the access control policy. For instance, when adding an element to the document, the user may even not be aware of the existence of a required attribute associated with it, as she is not entitled to access the attribute itself.

Support for Fine and Coarse Authorizations. The different protection re-quirements that different documents may have call for the support of access restrictions at the level of each specific document. However, re-quiring the specification of authorizations for each single document would make the authorization specification task too heavy. The system may then support, beside authorizations on single documents (or portions of doc-uments), authorizations on collections of documents [9]. The concept of DTD can be naturally exploited to this end, by allowing protection re-quirements to refer to DTDs or XML documents, where requirements specified at the level of DTD apply to all those documents instance of the considered DTD. Authorizations specified at DTD level are called **schema level** authorizations, while those specified at XML document level are called **instance level** authorizations.

Propagation Policy:

The structure of an XML document can be exploited by possibly applying different propagation strategies that allow the derivation of authorizations from a given set of authorizations explicitly defined over elements of DTD and/or XML documents. Some proposals therefore distinguish between two kinds of authorizations: **local**, and **recursive** [9]. Local authorizations defined on an element apply to all its attributes only. A recursive authorization defined on an element applies to its whole content (both attributes and sub elements). Recursive authorizations represent an easy way for specifying authorizations holding for the whole structured content of an element (for the whole document if the element is the root node of the document).

The models proposed assume that negative authorizations are always recursive, while positive authorizations may be either local or re-cursive. Besides downward propagation, upward propagation methods have been introduced. Here, the authorizations associated with a node in the XML tree propagate to all its parents.

Some of the most common propagation policies (which include also some resolution policies for possible conflicts) are described in the following.

o **No propagation.** Authorizations are not propagated. This is the case of local authorizations.

p **No overriding.** Authorizations of a node are propagated to its descendants, but they are all kept.

Most specific overrides. Authorizations of a node are propagated to its descendants, if not overridden. An authorization associated with a node n overrides a contradicting authorization[3] associated with any ancestor of n for all the descendants of n.

o **Path overrides.** Authorizations of a node are propagated to its descendants, if not overridden. An authorization associated with a node n overrides a contradicting authorization associated with an ancestor n' for all the descendants of n only for the paths passing from n. The overriding has no effect on other paths.

XML Access Control Models: Several access control models have been proposed in the literature for regulating access to XML documents. We start our overview of these models by presenting the first access control model for XML [9], which has then inspired.

Two authorizations (s, o, a) and (s', o', a') are contradictory if s = s', o = o', and a = a', but one of them grants access, while the other denies it.

Fine Grained XML Access Control System:

Damiani et al [9] propose a fine grained XML access control system, which extends the proposals in [14, 15, 16], exploiting XML's own capabilities to define and implement an authorization model for regulating access to XML documents. We now present the authorizations supported by the access control model and illustrate the authorizations enforcement process.

Authorizations Specification

Access authorization determines the accesses that the system should allow or deny. In this model, access authorizations are defined as follows.

Definition 1 ((Access Authorization)). An access authorization a ∈ Auth **is a five-tuple of the form: subject, object, action, sign, type , where:**

- subject ∈ AS **is the subject for which the authorization is intended;**
- object **is either a URI**∈Obj **or is of the form URI:PE, where URI**∈Obj **and**

PE is a path expression on the tree of document URI;

- action=read **is the action being authorized or forbidden;**
- sign ∈ +,− **is the sign of the authorization, which can be positive (allow access) or negative (forbid access);**
- type ∈ LDH, RDH, L, R, LD, RD, LS, RS **is the type of the authorization and regulates whether the authorization propagates to other objects and how it interplays with other authorizations (exception policy).**

Database Issues in Trust Management and Trust Negotiation: Trust management is the process of managing authorization decisions in a decentralized environment where many of the participants do not have pre established trust relationships, such as logins and passwords, with one another. Trust management is important for enterprise-level and cross-organizational database applications such as supply chain management, enterprise resource planning, and customer relationship management. Trust management research may also interest the database research community because of the former's affinity for a Datalog-based world, in which a query (authorization request) launches a multi-site search for a proof of authorization. To complicate the process, sites have autonomy and may not always cooperate in proof construction; it is not always obvious where to find the facts and rules needed to construct a proof; and attempts to access particular facts and rules may spawn new authorization requests.

Introduction to Trust Management: Authorization is one of the most important problems in computer security and privacy. It lies at the heart of meeting the objectives of confidentiality, integrity, and availability. Within a single organization, pre-established trust relationships are used to assign authorizations and prearranged information such as login names and passwords can serve as the basis for making authorization decisions at run time. For instance, an enterprise has pre-established trust relationships with its employees, so it is necessary only to authenticate that a certain resource request is being made by a certain employee for the request to be given appropriate authorization. On the other hand, when resource provider and resource requester belong to different organizations or have no prior relationship whatsoever, there are no pre-existing trust relationships. This problem can be mitigated slightly by using manual procedures for cross-domain authentication and authorization, such as maintaining local logins and passwords (or lists of X.509 identities) for all employees in a partner company.

Over the last 10–15 years, researchers have proposed new techniques that enable on-line parties to establish trust on the fly, as the need arises. Bina et al. proposed using characteristics other than identity, attested to by known authorities in digital certificates, as a basis for authorization on the Internet. Blaze et al. introduced a complementary approach to authorization based on delegation of privileges and coined the term trust management to describe it. Ellison et al. introduced a similar scheme called SPKI. Rivest et al. introduced a scheme called SDSI that provides an ingenious way to introduce names and bind them to public keys controlled by individuals and groups, which greatly facilitates identifying authorized principals electronically. Following these seminal works, a great deal of work has been done, much of which we will survey in this chapter. Trust management systems typically use cryptographic credentials to convey information relevant to authorization decisions. The authorization decision determines whether a given set of credentials demonstrate that a given request to access a resource, such as a web or peer-to-peer service, is authorized, which is to say that the access request complies with current policy governing that resource. This raises two additional problems that we also survey here. First, the credentials are issued in a decentralized manner, and somehow the relevant credentials need to be collected or otherwise made available to the authorization evaluation process. Second, some credentials carry sensitive, confidential information, and may need to be subject to access control themselves when dealing with an unfamiliar resource provider or requester. The same may also be true of policy: an access control policy may give clues about the nature of the resource it protects. For example, if a patient's prescription can be viewed only by their pharmacist or by their parent, then one can guess that the prescription is for a child. To preserve the privacy of the resources that they protect, policies themselves may need protection just like any other resource. In other words, access to the contents of a policy may need to be governed by another access control policy. These additional authorization decisions can also be based on credentials. Thus, there is a need for a process of credential exchange in which both parties seek to enable a positive authorization decision for the main resource request, while also supporting the additional authorization decisions that may be necessary to achieve this. This process is trust negotiation [64, 65], an automated approach to establishing bilateral trust between two parties at run time.

What is Trust Management?:

Traditional access control models base authorization decisions on the identity of the principal who requests a resource. In an operating system, this identity may be a user name that must appear on an access control list (ACL) associated with a file, if access to the file is authorized. In a large enterprise, an identity may be a distinguished name mentioned in a Public Key Infrastructure (PKI) certificate. In these and similarly closed environments, the identities of all authorized resource requesters are presumed to be known to the resource provider, or at least to some local administrator who aggregates identities into groups or roles to which the resource provider can grant access. However, the number of autonomous services that are administered in a decentralized manner (i.e., within different security domains) has increased enormously on the Internet. As a result, services are often provided to clients whose identities are not previously known to the service provider. Similarly, the participants in a peer-to-peer system need to establish mutual trust in one another. In such a decentralized environment, the traditional access control mechanisms such as ACLs cannot be used to secure the system without excluding vast numbers of valuable and well-intentioned clients and peers.

The trust management (TM) approach, first developed by Blaze et al. [11], aims to provide a basis for authorization in highly decentralized environments by enabling resource owners to delegate authority to other entities who will help them identify the appropriate requesters to authorize. In this manner, resource owners and other policy authors can enlist the assistance of appropriate authorities in determining the suitability of individual requesters for authorization.

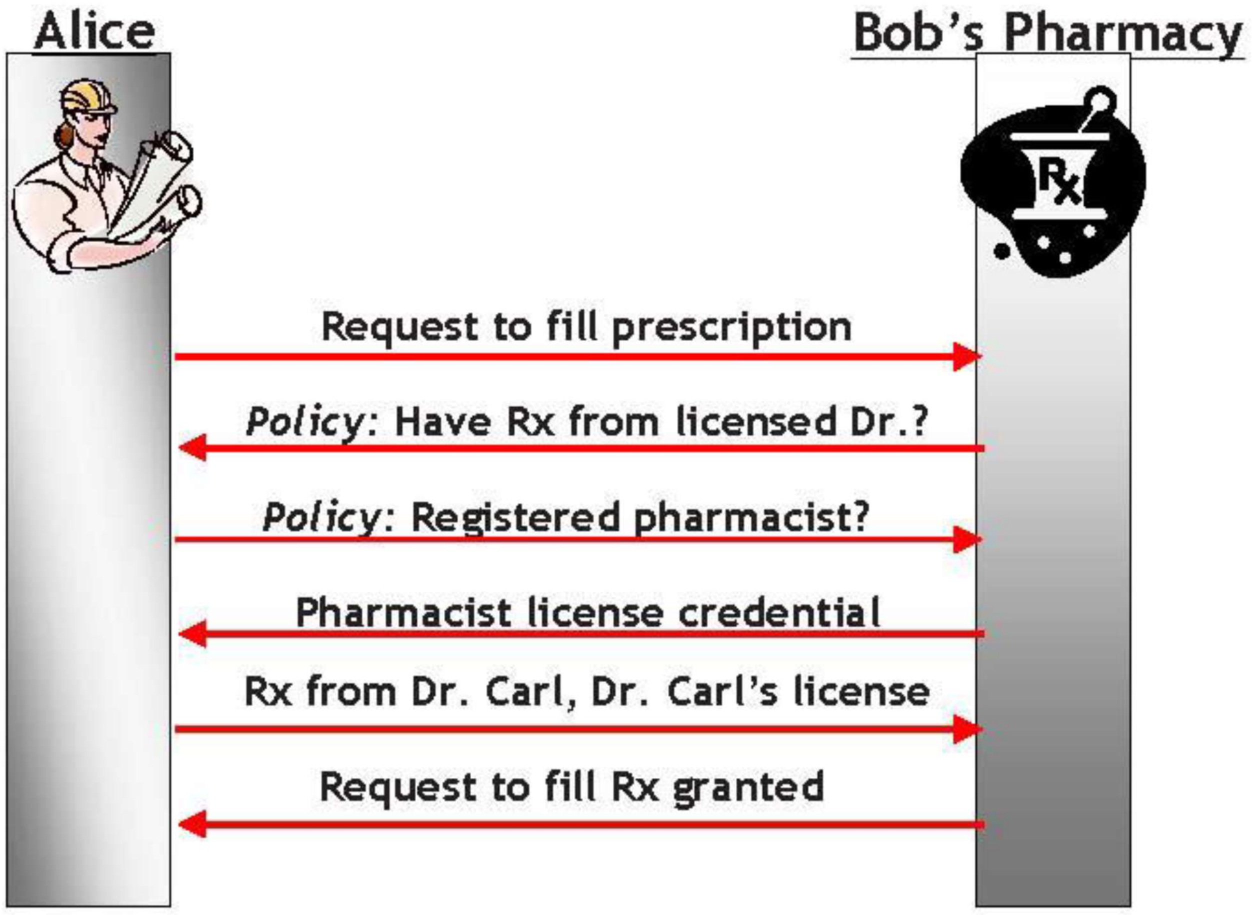
Trust management relies on digital credentials, which are unforgivable statements signed by their issuer. Typically, a digital credential contains an assertion about the properties of one or more principals mentioned in the credential. The best-known standard for digital credentials is X.509v3 [31], though many alternatives exist. Most of these schemes rely on public key cryptography: the credential issuer signs the credential using its private key, and anyone can verify the contents of the credential by obtaining the corresponding public key and checking the signature. In the US, recent legislation such as the Sarbanes-Oxley Act has forced the widespread adoption of the public key infrastructures needed to support digital credentials. Today's digital credentials are typically identity certificates, i.e., they simply say what public key is associated with a particular principal. However, current credential standards already support the inclusion of additional information describing a principal's properties, such as one would need for a digital employee ID, driver's license, or birth certificate. In TM systems, security policy is made by local administrators to specify access control rules on local resources. Blaze et al. [10] said that trust management systems combined the notion of specifying security policy with the mechanism for specifying security credentials.

The authorization semantics of most TM systems is monotonic in the sense that if any given action is approved when given a set of evidence E (i.e., policies and credentials), then it will also be approved when given any superset of E. This means that no negative evidence is allowed in the system. Monotonicity ensures fail-safe behavior in which no potentially dangerous action is allowed by default, simply because of an inability to find negative credentials. This is especially important in decentralized environments due to the many factors that can prevent one from obtaining complete information about all the credentials in the system (network interruption, uncooperative credential repositories, lost information, etc.).

Besides the basic notion of delegation in which one entity gives some of its access rights to another entity, there are two additional delegation idioms that are most often discussed in the designs of trust management systems: appointment and threshold delegation. In the case of appointment, the appointer has the (appointment) right to confer on another (the appointee) an attribute or right that the appointer may not herself have. (In general, the conferred right can itself be an appointment right.) Threshold delegation is also called k-out-of-n ($n \geq 1$ and $n \geq k \geq 1$) delegation, meaning the authority is delegated to n parties, each of which only gets a fragment of it. It is effective only if at least k of these parties issue their requests or exercise their authorities in concert.

Compliance checking (also called policy evaluation or query evaluation) answers the question: Does a set of credentials prove that a request complies with the security policy associated with a given resource? The process of evaluating a query involves finding a chain of credentials that delegate authority from the resource owner to the requester. This process is also called credential chain discovery [47]. As we shall see, it can be helpful to imagine credential chains in graphical terms. To a first approximation, a credential chain can be thought of as a path from the resource provider to the requester in which nodes are principals and edges are credentials. However, the details of such a credential graph depend on the TM system and, in general, a chain may correspond to a richer sub graph structure.

Example trust negotiation:


Alice
Bob's Pharmacy
Request to fill prescription
Policy: Have Rx from licensed Dr.?
Policy: Registered pharmacist?
Pharmacist license credential
Rx from Dr. Carl, Dr. Carl's license
Request to fill Rx granted

As mentioned earlier, trust negotiation is the process of establishing bilateral trust at run time. Trust negotiation uses verifiable, unforgivable digital credentials that describe principals' properties, together with explicit policies that describe the properties that a principal must possess in order to gain access to a particular resource. When Alice wishes to access a resource owned by Bob, she and Bob follow one of many proposed trust negotiation protocols to determine whether she has the right properties, i.e., whether her credentials satisfy the policy that Bob has specified for access to that resource.

Security in Data Warehouses and OLAP Systems:
Unlike in operational databases, aggregation and derivation play a major role in on-line analytical processing (OLAP) systems and data warehouses. Unfortunately, the process of aggregation and derivation can also pose challenging security problems. Aggregated and derived data usually look innocent to traditional security mechanisms, such as access control, and yet such data may carry enough sensitive information to cause security breaches.

Introduction:
With rapid advancements in computer and network technology, it becomes a common practice for organizations to collect, store, and analyze vast amounts of data quickly and efficiently. On-line analytical processing (OLAP) systems and data warehouses of today are used to store and analyze everything – vital or not – to an organization. The security of data warehouses and OLAP systems is crucial to the interest of both organizations and individuals. Stolen organizational secrets may cause serious and immediate damages to an organization. Indiscriminate collection and retention of data represents an extraordinary intrusion on privacy of individuals. Security breaches in governmental data warehouses may lead to losses of information that translate into financial losses or losses whose values are obviously high but difficult to quantify (for xample, national security).

Unlike in traditional databases, information stored in data warehouses is typically accessed through decision support systems, such as OLAP systems. OLAP systems help analysts to gain insights to different perspectives of large amounts of data stored in a data warehouse. Due to the sheer volume of data, OLAP systems heavily depend on aggregates of data in order to hide insignificant details and hence to accentuate global patterns and trends. As the underlying data model, a data cube [15] can nicely organize multi-dimensional aggregates formulated by dimension hierarchies. Although security breaches in a data warehouse are possible in many ways, the most challenging threat is from insiders who have legitimate accesses to data through OLAP queries. Unfortunately, most of today's OLAP systems lack effective security measures to safeguard the data accessed through them. Existing security mechanisms can at best alleviate security breaches but cannot completely remove the threat. Data sanitization has long been recognized as insufficient for protecting sensitive data by itself due to potential linking attacks [24].

Access control techniques, although mature in traditional data management systems, are usually not directly applicable to OLAP systems and data warehouses due to the difference in data models. Moreover, OLAP systems and underlying data warehouses are especially vulnerable to indirect inferences of protected data. The aggregation process used by OLAP systems does not completely destroy sensitive information. The remaining vestiges of sensitive information, together with knowledge obtained through out of bound channels, can cause disclosures of such information. Although studied since 1970´s in statistical databases, inference control for on-line systems is largely regarded as impractical due to its negative-in-tone Complexity results [7]. Most restriction-based inference control methods adopt a detecting-then-removing approach. The detection of inferences must take into accounts all combinations of answered queries, which implies complicated on-line computations and constant bookkeeping of queries. Even at such a high cost, each method usually applies to only a few unrealistically simplified cases, such as with only one aggregation type. These facts partially explain why inference control is absent in most commercial OLAP systems. On the other hand, off-line inference control methods have long been used in releasing census tables, which demonstrates that the threat of inferences is real.

Data Warehouses and OLAP Systems:

A centralized data warehouse is usually used to store enterprise data. The data are organized based on a star schema, which usually has a fact table with part of the attributes called dimensions and the rest called measures. Each dimension is associated with a dimension table indicating a dimension hierarchy. The dimension tables may contain redundancy, which can be removed by splitting each dimension table into multiple tables, one per attribute in the dimension table. The result is called a snowflake schema. A data warehouse usually stores data collected from multiple data sources, such as transactional databases throughout an organization. The data are cleaned and transformed to a common consistent format before they are stored in the data warehouse. Subsets of the data in a data warehouse can be extracted as data marts to meet the specific requirements of an organizational division. Unlike in transactional databases where data are constantly updated, typically the data stored in a data warehouse are refreshed from data sources only periodically.

Coined by Codd et. al in 1993 [9], OLAP stands for On-Line Analytical Processing. The concept has its root in earlier products such as the IRI Express, the Comshare system, and the Essbase system [29]. Unlike statistical databases which usually store census data and economic data, OLAP is mainly used for analyzing business data collected from daily transactions, such as sales data and health care data [27]. The main purpose of an OLAP system is to enable analysts to construct a mental image about the underlying data by exploring it from different perspectives, at different level of generalizations, and in an interactive manner. Popular architectures of OLAP systems include ROLAP (relational OLAP) and MOLAP (multidimensional OLAP). ROLAP provides a front-end tool that translates multidimensional queries into corresponding SQL queries to be processed by the relational backend. MOLAP does not rely on the relational model but instead materializes the multidimensional views. Using MOLAP for dense parts of the data and ROLAP for the others leads to a hybrid architecture, namely, the HOLAP or hybrid OLAP.

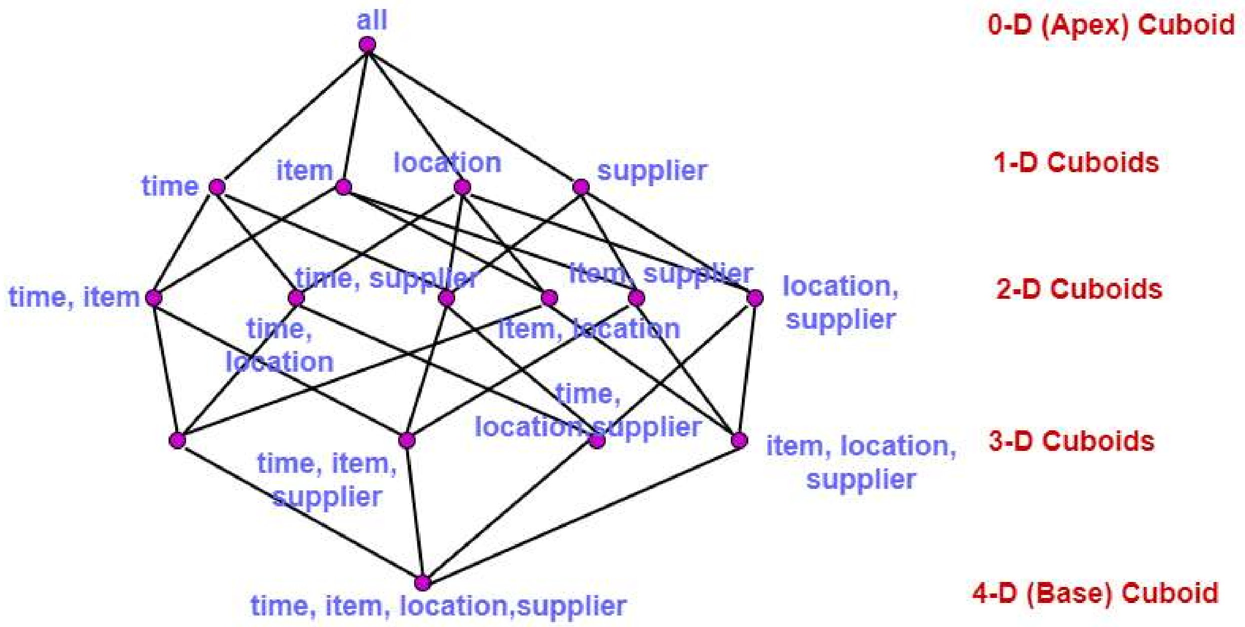
The requirements on OLAP systems have been defined differently, such as the FASMI (Fast Analysis of Shared Multidimensional Information) test [23] and the Codd rules [9]. Some of the requirements are especially relevant to this chapter. First, to make OLAP analysis an interactive process, the OLAP system must be highly efficient in answering queries. OLAP systems usually rely on extensive pre-computations, indexing, and specialized storage to improve the performance. Second, to allow analysts to explore the data from different perspectives and at different level of generalization, OLAP organizes and generalizes data along multiple dimensions and dimension hierarchies. The data cube model we shall address shortly is one of the most popular abstract models for this purpose.

Data cube was proposed as a SQL operator to support common OLAP tasks like histograms and sub-totals [15]. Even though such tasks are usually possible with standard SQL queries, the queries may become very complex. The number of needed unions is exponential in the number of dimensions of the base table. Such a complex query may result in many scans of the base table, leading to poor performance. Because sub-totals are very common in OLAP queries, it is desired to define a new operator for the collection of such sub-totals, namely, data cube.

Figure 1 depicts a fictitious data cube. It has two dimensions: time and organization with three and four attributes, respectively. We regard all as a special attribute having one attribute value ALL, which depends on all other attribute values. The attributes of each dimension are partially ordered by the dependency relation _ into a dependency lattice [18], that is, quarter _ year _ all and employee _ department _ branch _ all. The product of the two lattices gives the dependency lattice of cuboids. Each element of the dependency lattice is a tuple < T,O >, where T is an attribute of the time dimension and O is an attribute of the organization. Attached to each such tuple < T,O > is an empty two-dimensional array, namely, a cuboid. Each cell of the cuboid < T,O > is also a tuple < t,o >, where t and o are attribute values of the attribute T and O, respectively. The dependency relation extends to be among cells. For example, a cell < Y 1,Bob > depends on the cells < Q1,Bob >, < Q2,Bob >, < Q3,Bob >, and < Q4,Bob >. Hence, all cells also form a dependency lattice.

A base table with the schema (quarter, employee, commission) is used to populate the data cube with values of a measure attribute commission. Each record in the base table, a triple (q, e,m), is used to populate a cell< q,e > of the core cuboid < quarter, employee >, where q, e, and m are values of the attributes quarter, employee, and commission, respectively. Some cells of < quarter, employee > remain empty (or having the

NULL value), if corresponding records are absent in the base table. If multiple records correspond to the same cell, since the two attributes quarter and employee are not necessarily a key of the base relation, they are aggregated using the aggregation function SUM. All cuboids are then populated using the same aggregation function. For example, in the cuboid < year, employee >, a cell < Y 1,Bob > takes the value 8500, which is the total amount of the four cells it depends on, < Q1,Bob >, < Q2,Bob >,< Q3,Bob >, and < Q4.



Secure multi-party data mining allows multiple distrusted parties to cooperatively compute aggregations over each other's data [31, 14]. Cryptographic protocols enable each party to obtain the final result with minimal disclosures of their own data. This problem is different from inference control, because the threat of inferences comes from what users know, not from the way they know it. The k-anonymity model enables sensitive values to be released without threatening privacy [24, 36]. Each record is indistinguishable from at least k − 1 others because they all have exactly the same identifying attribute values. An adversary can link an individual in the physical world to at least (the sensitive values of) k records, which is considered a tolerable privacy threat. Inference control and the k-anonymity model can be considered as dual approaches. The information theoretic approach in [22] formally characterizes insecure queries as those that bring a user with more confidence in guessing possible records [22]. However, such a perfect-secrecy metric will not tolerate any partial disclosure, such as those caused by aggregations.

Security Requirements:

The Threat of Inferences:

Unlike in traditional databases where unauthorized accesses are the main security concern, an adversary using an OLAP system can more easily infer prohibited data from answers to legitimate queries. Example 1 illustrates an one dimensional (or 1-d for short) inference where the sensitive cell is inferred using exactly one of its descendants.
A multi-dimensional (or m-d) inference is the complementary case of 1-d inferences. That is, a cell is inferred using two or more of its descendants, and neither of those descendants causes 1-d inferences. Example 2 illustrates an m-d inference in a two-dimensional SUM-only data cube. Example 3 and 4 illustrate m-d inferences with MAX-only, and with SUM, MAX, and MIN.

The Requirements:

As illustrated in above examples, a security solution for OLAP systems must combine access control and inference control to remove security threats. At the same time, providing security should not adversely reduce the usefulness of data warehouses and OLAP systems. A practical solution must achieve a balance among following objectives.

- Security: Sensitive data stored in underlying data warehouses should be guarded from both unauthorized accesses and malicious inferences. Such a definition of security considers not only the information a user can directly obtain from an OLAP system, but also those that he/she can derive using answers to seemingly irrelevant queries.
- Applicability: The security provided by a solution should not rely on any unrealistic assumptions about OLAP systems. In particular, assumptions made in statistical databases are usually not unacceptable in OLAP applications. A desired solution should cover a wide range of scenarios without the need for significant modifications.
- Efficiency: The name of OLAP itself indicates the interactive nature of such systems. Most queries should be answered in a matter of seconds or minutes. A significant portion of the OLAP literature has been devoted to meeting such stringent performance requirements. A desired security must be computationally efficient, especially with respect to on-line overhead.
- Availability: Data should be readily available to legitimate users who have sufficient privileges. A solution must place security upon justifiable restrictions of accesses in the sense that removing the restrictions will either lead to security breaches or render the method computationally infeasible.
- Practicality: A practical security solution should not demand significant modifications to the existing infrastructure of an OLAP system. A solution should take advantage of any query-processing mechanisms and security mechanisms that are already in place. The main challenge, however, lies in the inherent tradeoff between above objectives. To have provable security and justifiable availability in varying settings of OLAP systems usually implies complicated on-line computations, which are expensive and hard to implement. The methods we shall describe in this chapter represent efforts towards a balance among these objectives.

A Three-Tier Security Architecture:

Security in statistical databases usually has two tiers, that is, sensitive data and aggregation queries. Inference control mechanisms check each aggregation query to decide whether answering the query, in addition to previously answered queries, may disclose any protected data through inferences. However, applying such a two-tier architecture to OLAP systems has some inherent drawbacks. First, checking queries for inferences at run time may bring unacceptable delay to query processing. The complexity of such checking is usually high due to the fact that m-d inferences must be checked against sets of queries instead of each individual query. Second, inference control methods cannot take advantage of the special characteristics of an OLAP application under the two-tier architecture.

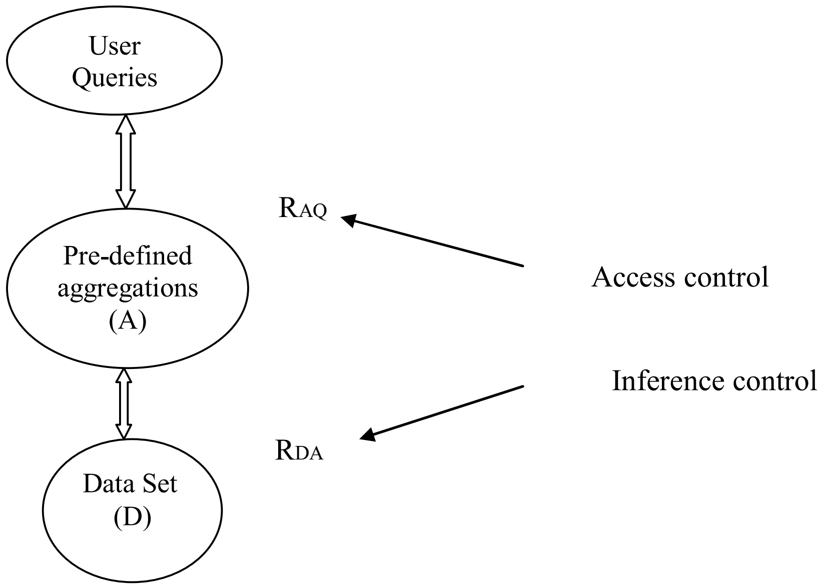


Fig :2 A Three-Tier Inference Control Architecture

The methods we shall review are based on a three-tier security architecture. As illustrated in Figure 2, this architecture introduces an intermediate aggregation tier between the data tier and the query tier. More specifically, the architecture has three tiers and three relations, and the aggregation tier must satisfy three properties. First, inference control is enforced between the aggregation tier and the data tier such that the former is secure with respect to the latter. Access control then helps to enforce that only safe aggregations will be used to compute results to queries. Second, the size of the aggregation tier must be comparable to the data tier. Third, the problem of inference control can be partitioned into blocks in the data tier and the aggregation tier such that security only need to be ensured between each corresponding pair of blocks in the two tiers.
The three-tier architecture helps to reduce the performance overhead of inference control in several aspects. The first property of the model implies that the aggregation tier can be pre-computed such that the computation intensive part of inference control can be shifted to off-line processing. The on-line part is to enforce access control based on whether a query can be rewritten using the aggregation tier (that is, security through views). Second, the last two properties both reduce the size of inputs to inference control algorithms and consequently reduce the complexity. Note that an aggregation tier can be designed to meet the second property, but the size of the query tier is inherently exponential in the size of the data tier. The third property also localizes inference control tasks to each block of the data tier so a failure in one block will not affect other blocks.

Securing OLAP Data Cubes:

The data cube is a natural data model for OLAP systems and underlying data warehouses. This section reviews several methods in safeguarding OLAP data cubes against both unauthorized accesses and indirect inferences.

SUM-only Data Cubes:

Cardinality-Based Method:

The cardinality-based method by Dobkin et. al [13] determines the existence of inferences based on the number of answered queries. In a data cube, aggregations are pre-defined based on the dimension hierarchy, and what may vary is the number of empty cells, that is previously known values. Recall that in Section 3.1, Example 1 illustrated a straightforward connection between 1-d inferences and the number of empty cells in a data cube. That is, an 1-d inference is present when an adversary can access any cell that has exactly one ancestor in the core cuboid. A similar but less straightforward connection also exists between m-d inferences and the number of empty cells, as we shall show in this here.

The model for inferences in this case is similar to that given by Chin et. al [8], but the queries are limited to data cube cells. Here we only consider one level dimension hierarchy where each dimension can only have two attributes, that is the attribute in core cuboid and the all. For each attribute of the core cuboid, we assume an arbitrary but fixed order on its domain. Although an attribute may have infinitely many values, we shall only consider the values that appear in at least one non-empty cell in the given data cube instance. The number of such values is thus fixed. From the point of view of an adversary, the value in any non-empty cell is unknown, and hence the cell is denoted by an unknown variable. The central tabulation in Table 1 rephrases part of the core cuboid in Figure 1.

Table 1 also includes cells in descendants of the core cuboid, namely, the aggregation cuboids. These are _all, employee_, _quarter, all_, and _all, all_, as we only consider one-level dimension hierarchy. For SUM-only data cubes, the dependency relation can be modeled as linear equations. At the left side of those equations are the unknown variables in the core cuboid, and at the left side the values in the aggregation cuboids. Table 2 shows a system of nine equations corresponding to the nine cells in the aggregation cuboids.

Table 1. Modeling A Core Cuboid

	Q1	Q2	Q3	Q4	ALL
Bob	x1	x2	x3		8500
Alice		x_4	x_5		10000
J im	x_6			x_7	6100
M allory	x8			x9	12400

Table 2. Modeling the Aggregation Cuboids

111000000	x_1		8500
000110000	x_2		10000
000000011	x_4		12400
100001010	x_5	=	10000
010100000	x_6		6000
001010000	x_7		11000
000000101	x_8		9000
111111111	x_9		36000

Table 3. The Reduced Row Echelon Form $M_{\mathbf{rref}}$

0010	1	0 0 0	0	
0001	1	0 0 0	0	
0000	0	100−1		
0000	0	0 1 0	1	
0000	0	0 0 1	1	
0000	0	0 0 0	0	
0000	0	0 0 0	0	

Next we obtain the reduced row echelon form (RREF) Mrref of the coefficients matrix through a sequence of elementary row operations [19], as shown in Table 3. From Mrref it can be observed that the system of linear equations in Table 2 has infinitely many solutions. This means that an adversary cannot infer the entire core cuboid from the given aggregation cuboids. However, the first row vector of Mrref being a unit vector (that is, it has a single 1) indicates that the value of x1 must remain the same among all the solutions to the system of equations. Consequently, the adversary can infer Bob's commission in Q1.

The existence of at least one unit row vectors in Mrref is indeed the necessary and sufficient condition for any unknown variable to have the same value among all the solutions [8]. We shall adopt this notion to model inferences in SUM-only data cubes. Notice that for the special case of 1-d inferences, as shown in Example 1, the coefficients matrix itself would have a unit row vector (which will certainly also appear in Mrref). It is well-known that the RREF of a m×n matrix can be obtained by a Gauss-Jordan elimination with complexity O(m2n) [19]. The number of empty cells can only determine the existence of 1-d inferences in two extreme cases. First, if the core cuboid has no empty cell, then trivially it is free of 1-d inferences as long as all the attributes have more than one value. The second straightforward result says that any data cube whose core cuboid has fewer non-empty cells than the given upper bound 2k−1.

UNIT – IV

Security Re-engineering for Databases: Concepts and Techniques

In most of today's information system infrastructures employed by e-businesses and government organizations, database management systems (DBMSs) serve as the back-end for managing and delivering often mission-critical and sensitive data. Although such infrastructures are comprised of many components, such as networks and application servers, we conjecture that the data managed in databases is often the most valuable asset to an organization. The data typically have been curated and maintained over many years, and their loss or corruption would be much more difficult (and costly) to compensate for than the failure of some other infrastructure components.

There are several reasons for this situation. First, there is a substantial time lag between the proposal of a better database security technique and its realization in a new release of a DBMS. Even then, the new technologies need to be learned and used appropriately to further secure a database. Second, and more importantly, the shortcomings of appropriately securing databases stem from circumstances that are also found all too often in other computer security settings. These include:

- Lack of clearly defined security policies. If security policies are not well understood or not clearly stated, they cannot be effectively implemented using database security mechanisms, leaving the database system open to security threats.
- Poor security design. As with many other types of computing systems, for databases too security is often an afterthought. Security policies are formulated and implemented in an ad-hoc fashion, leading to an incoherent overall database security design and thus resulting in potential vulnerabilities that can be exploited by malicious users and intruders.
- Dynamic nature of applications and user tasks/roles. Over time, database users are added or removed and applications are added, upgraded, or removed, often leaving the implementation of associated security policies at the database back-end untouched. Outdated and obsolete security policies and corresponding enforcing security mechanisms pose a critical threat to database security.

One of the most significant problem contributing to the current situation in better securing databases is that of insider misuse. That is, legitimate users of an application or database who (maliciously) tamper with the integrity and confidentiality of the data. As stated in [26, 48], insider misuse is still the biggest threat to security not only in database systems. Clearly, if security policies are not designed and implemented in a coherent and consistent fashion, intrusions and insider misuse pose a great threat to database security.

We present the fundamental concepts and techniques to support different security re-engineering tasks for relational databases. The proposed approach is data-driven, meaning that a comprehensive evaluation of the security of a given database necessitates the evaluation of the quality of the data to be protected first. Only if it is known that the mission-critical and sensitive data is of good quality (which is often not the case in practice, suitable data and user profiling techniques can be deployed.

We further present a methodological framework, called the access path model, in which administrators and security personnel can discover, annotate, and evaluate access paths. An access path represents the current (admissible) ways in which application users can operate on the data managed in a database. Correlating data accesses and user accounts (represented in the form of profiles) at the database layer and application layer is crucial in order to strengthen or replace current security policies. A feature of the access path model is that it allows to back-track accesses to and operations on database relations to application users by correlating user profiles and audit trails managed at the database and application layers. The information extracted from such access exploration and analysis tasks is then used in the re-design of existing or the implementation of new security mechanisms.

Insider Misuse and Anomaly Detection:
An overarching theme in computer security, which has been studied for more than 20 years and also motivates the need for security re-engineering concepts and techniques for databases, is intrusion detection. In general, an intrusion is considered an activity that violates the security policy of a system. Intrusion detection systems (IDSs) are based on the assumption that the behavior of an intruder is different from that of an authorized user and that respective unauthorized activities can be detected and reacted upon. One typically distinguishes among host-based IDSs, networkbased IDSs and application-based IDSs. Host-based IDS uses audit data produced by operating system calls on a local host, such as process executions, resource consumptions, and file accesses. Network-based IDS, on the other hand, are placed in a network and monitor all network traffic. They analyze packages for particular signatures and try to detect and prevent inappropriate network usages. Application-based IDSs can be considered a special class of host-based IDSs. They collect and analyze audit data specific to a particular application, application component or function realized on one or more hosts. One could argue that a database management system (DBMS) is a particular type of such applications. However, as we will discuss in the following sections, traditional application-based IDS techniques are not sufficient to realize an effective intrusion detection system for a DBMS. For this, it is important to understand the methodologies IDSs employ for the detection of security policy violations. These methods are discussed next.

Misuse Detection:
Misuse detection is one of two classes of intrusion detection approaches. Misuse detection is based on signatures that describe the characteristics of known system attacks and vulnerabilities. The signatures are typically derived from security policies. Mechanisms implementing misuse detection approach monitor the system, network, or application for any activities that match the specified signatures, e.g., a specific sequence of system calls or a particular type of packet traffic between two hosts. Although misuse detection approaches work very well for known attacks and misuse patterns, they fail in dealing with new attacks and security threats. These misuse detection approaches need to continuously update the signatures of security threats and vulnerabilities in order to effectively prevent intrusions.

Anomaly Detection:
Most of the approaches to intrusion detection typically combine misuse detection with anomaly detection. Anomaly detection a popular and effective ones, as they are based on the normal behavior of a subject, e.g., a user, system component, or application. In anomaly detection, information about repetitive and usual behavior is collected and suitably represented as statistical models of normal behavior, e.g., in the form of profiles. Current user or system activities are then compared to such profiles. If the activities significantly deviate from the profile, the activities are considered intrusive. Deviations from the normal behavior indicate potential violations of a security policy or an intrusion and thus might trigger respective responses. The advantage of anomaly detection over misuse detection is that previously unseen attacks and activities have a better chance of being detected. Clearly, the effectiveness of an anomaly-based IDS depends on how well normal behavior is modeled and how tight thresholds are set to indicate a deviation from the normal behavior, aiming to reduce false positives (activities that are not normal but do not violate a security policy) and avoiding false negatives (suspicious activities that are considered normal).

Insider Misuse:
The notion of intrusion intuitively refers to subjects that gain access to a system to which they have no legitimate access otherwise. Such intrusions occur by exploiting system vulnerabilities or by simply cracking or stealing accounts of legitimate users. Once the subject has access to a system, the subject then is considered a legitimate user by the system and has all the privileges and rights associated with that user; the intruder is now considered an insider. Thus, one objective of a security re-engineering approach to database systems can be framed as “effectively detecting and preventing insider misuse”. In particular, the CSI/FBI reports state that “The threat from inside the organization is far greater than the threat from outside the organization” and “Inside jobs occur about as often as external attacks”. Clearly, the problem of insider misuse is aggravated in the context of database systems that manage large collections of sensitive and often mission-critical data. There are many sources for potential insider misuse, ranging from the frequently mentioned “disgruntled employee” who maliciously tampers with the integrity of the data, to outside hackers, criminals, and spies that gain unauthorized access to the data.

Data and User Profiling:

The basic technique underlying the detection of intrusions and insider misuse, and subsequently the re-engineering of security mechanisms, is to monitor what types of actions users perform on a database system. In the following, we take a data-centric view on this and detail different profiling approaches. We first elaborate on some standard database auditing techniques as important prerequisite for profiling.

Auditing:

In the context of database systems, auditing is the process of monitoring and recording selected database event and activities [25, 42]. Auditing is primarily used to provide for accountability, the validation of security policies, and to capture and review the observed behavior of applications, users and database objects. Auditing is also often a requirement for organizations that have to comply with federal laws and regulations such as the Health Insurance Portability and Accountability Act (HIPAA) of 1996, Sarbanes-Oxley Act of 2002, and the Graham-Leach-Bliley Act (GLBA) of 1999. In the latter cases, auditing primarily serves the purpose of establishing accountability. NIST has established six components of security audit criteria [3], which can directly be adapted to database systems. The activities are the (1) selection of security audit events, (2) security audit data generation, (3) security audit event storage, (4) audit review, (5) security audit analysis, and (6) automatic response. Figure 1 illustrates the sequence of these criteria as an approach to using auditing techniques for evaluation activities, which is preceded by an extra step, the analysis of application specific security requirements.

Most of today's commercial and open-source DBMS provide audit mechanisms and architectures that support such activities and which mostly vary in terms of the granularity of access information that can be recorded in audit trails. For example, database triggers provide a convenient means to record information about SQL insert, update, and delete statements against database relations, including the recording of old and new values of updated tuples. Activities related to the creation, modification, and deletion of database objects resulting from SQL data definition language (DDL) statements can be audited as well, often referred to as the auditing of accesses with respect to system privileges. Some database systems even offer mechanisms that go beyond simple SQL statement level auditing by monitoring data accesses based on content, such as Fine-Grained Auditing (FGA) introduced with Oracle 9i. This is in particular useful when access information about SQL select statements needs to be recorded.

In addition to database triggers and the SQL audit command supported by many DBMS, another useful technique are stored procedures. Instead of individual SQL insert, update and delete statements issued from an application, stored procedures are called. A stored procedure then executes the data modification statements and also records (in auxiliary relations) some extra context information about the modifications such the current user role, other users currently running database sessions, or the number of records that have been modified through these statements.

Data Profiling:

Most approaches to misuse detection are user-centric. Their objective is to determine how users typically behave in terms of operations on the database and how their current behavior deviates from previous behavior. We conjecture that for the security re-engineering of databases, the evaluation and strengthening of security has to take a data-centric view. That is, first the behavior of the data being managed in the database needs to be evaluated before techniques are employed to detect and evaluate the (potentially anomalous) behavior of users. There are several reasons for such an approach. First, accidental or malicious tampering with the security and integrity of data is often only detected at the data level, that is, when erroneous, missing, extra, or anomalous data are discovered. This aspect leads to the second argument, where one would like to back-trace anomalous data to users who operate on the data, something that is not trivial in particular in the case of shared database user accounts, which is common in many application settings and will be discussed in Section 4. Finally, and most importantly, approaches to establishing some normal user behavior typically assume that the data users operate on are normal, i.e., the data are correct and of good quality, something that is often not the case for production databases where evaluating and maintaining the quality of data is a major concern.

Snapshot Profiles:

The first step in security re-engineering for databases is to determine and evaluate the properties of data that is to be protected from misuse. Given a database schema consisting of a set of relations R, those relations are investigated that manage mission-critical or sensitive data. Similar to the approaches suggested in [25] and [56], for a given relation R ∈ R withA1,..., An, the values for individual attributes Ai are analyzed. This analysis includes determining the distribution and frequency of attribute values, e.g., in the form of histograms, and the minimum and maximum values and lengths for numerical and alpha-numerical attributes, respectively.

Such a simple analysis, which can be done using SQL statements at the relation level, can reveal quite a lot of useful information, for example, outliers that do not conform to certain properties the data are assumed to have. The analysis can also be extended to sets of tuples from one or more relations where now tuples and combinations thereof are analyzed and correlations among attribute values are determined using standard association analysis techniques (see, e.g., [57]). Assume, for example, two relations R1 (A, C) and R2(B,D) with a foreign key dependency R2.B → R1.A. For any two matching tuples t1 ∈ R1, t2 ∈ R2, an association can describe that the value of t1.C always determines the range of the attribute value t2.D. The association rules discovered are then evaluated and compared to what is expected from the data. In principle, many of the analysis tasks resemble standard actions employed in the evaluation of the quality of the data [9, 18]. It should also be noted that in production databases, most of the above information is typically readily available. This is in particular the case where statistics for relations are periodically maintained to provide cost-based query optimizer with information for choosing efficient query evaluation plans. Statistical information about the relations is then available in the database's data dictionary.

For data profiling, it is assumed that the above analysis tasks are performed on a snapshot of the database, that is, at a particular point in time. This can be done in a batch-mode when the workload of the database is low (e.g., during night), or by using a stand-by or recent backup database. For large-scale databases with hundreds of relations, clearly not all relations are analyzed but only those that are relevant to specific security policies or those that contain sensitive or mission-critical data as indicated in the initial step in Figure 1. The snapshot profiles for these individual relations or parts thereof (e.g., sets of tuples that have some specific properties or contain particularly sensitive information) are managed in separate relations, specifically designed for access to database security mechanisms (see Section 5). In the simplest case, a snapshot profile for a relation R ∈ R determined at a particular point in time t, denoted DataProf (R, t), is a collection of measure-value pairs that describe properties of attributes of R. The measure is related to the data, e.g., number or frequencies of different attribute values, and the measure value denotes what has been computed for a measure.

Temporal Profiles and Access Properties:

Snapshot profiles describe properties of some relations' data at a single point in time, that is, for a given database instance. In order to further evaluate the security of the database and to develop respective enforcing security mechanisms, however, it is important to get a good understanding of how the data behave and evolve over time. For this, we distinguish two objectives: 1. determining the behavior of the data over time, and 2. determining the behavior of accesses to the data over time.

timestamp	user	operation	new tuple	old tuple
09-01-07 09:12:14	scott	insert	(3,8,12)	–
09-01-07 09:12:15	scott	insert	(4,9,7)	–
09-01-07 09:13:01	smith	update	(3,8,12)	(3,8,13)
09-01-07 09:13:02	jones	delete	–	(4,9,7)
. . .	. . .	. . .	. . .	. . .

The first objective can be realized by periodically taking snapshot profiles and analyzing sequences of snapshot profiles for certain trends. Key to such an approach is the appropriate choice of time-granularities, that is, instants in time when to perform the snapshot profiling, an aspect that heavily depends on the particular application setting of the database. Assume for a relation R ∈ R, snapshot profiles DataProf(R, t1), . . . DataProf(R, tk) have been determined at times t1,...,tk. The goal of the analysis of these profiles then is to discover trends in the behavior of the data. These trends, managed in temporal profiles with a measure/value pair structure similar to snapshot profiles, can include coarse grained properties such the increase or decrease ratio of the number of tuples in R between consecutive timestamps ti and ti+1 as well as more fine-grained properties such as the significant variations in the frequency and/or distribution of attribute values. The outcome of this analysis is again evaluated and verified with respect to the expected behavior of the data. The purpose of this type of trend analysis is less to derive further security mechanisms but to gain confidence in individual snapshot profiles and the properties of relations and data at respective points in time. Again, these profiling tasks can go hand in hand with the process of managing statistics for relations for query optimization purposes (here now, such statistics are maintained over time, again in auxiliary relations).

User Profiling:

Following the data profiling tasks presented in the previous sections, the next step is to associate users with the behavior of the data and eventually determine models describing the normal behavior of users. Profiling of users or,

more precisely, their behavior over time, has been the focus of several related work in the context of relational database.
In order to suitably approach the user profiling task, it is important to understand the notion of a user in a database system. Underlying access control models in databases is the notion of authorization identifier (AuthID), which is either the identifier of a database user or a database role name. According to the SQL: 1999 standard [39], when an SQL session is initiated (e.g., an application connects to the database using a valid database user account), the authorization identifier is then determined by the DBMS. In the following, whenever we refer to a user, we mean a valid database user account used by either a person or an application. There are typically different types of users:

- Database Administrators (DBAs). These types of users possess various system privileges to manage physical database objects (e.g., the creation of database files), logical objects (e.g., the creation and deletion of relations, views, triggers etc.), user accounts, database roles and privileges, and system parameters and settings.
- Application Developers. These users are primarily responsible for the design, implementation, and maintenance of the database schemas (including structures such as indexes and stored procedures) underlying different applications.
- Application Users. Unlike the other two classes of users, application users do not have any system privileges, i.e., they are not allowed to manage database objects, but are only allowed to operate on the data of an application schema using insert, update, delete, and select statements or the execution of stored procedures.

Although DBAs have the most comprehensive and powerful set of privileges to perform operations on any database structure and object, profiling their behavior in terms of SQL DML statements against individual relations is not that meaningful, as they will rarely perform such operations. In general, profiling the behavior of DBAs is complicated as they are typically responsible for setting up an audit and profiling framework. Also, there is rarely a fine-grained typical behavior that can be derived from auditing all operations a DBA issues against a database as operations exclusively should comprise the management of logical and physical database objects, and not individual tuples in relations associated with application schemas. We will revisit this aspect again in a later section. A similar argument can be made for application developers. Applications are typically develop not on a production database serving mission-critical applications, but on a development database where the security is likely of less concern. Only in rare circumstances, application developers should perform management related operations on application schemas hosted at the production database, e.g., when structures are transitioned from the development database to the production database.

In the following, we are primarily concerned with profiling techniques for application users with respect to individual relations. Assume a set Udb of database users and a relation R with data access profile AccessProf(R, ti , tj) for a time window [ti , tj]. In order to analyze and evaluate what operations users in Udb performed on R, one can start off with a simple SQL group by statement to see the types and frequencies of operations each user performed on R. Clearly, not all users in Udb will have issued operations against R during the time window, also because they simply do not have any privileges to access R.

Following this kind of aggregating user-centric access information, one now focuses on the behavior of a particular user u ∈ Udb who operated on R. There are basically two meaningful ways by which entries in AccessProf(R, ti , tj) can be organized for u: by session and/or by database role. Typically, every operation occurs as part of a user session, which starts when a user connects to the database and has some kind of id. In a database session, the user can enable different database roles, depending on whether roles are supported by the DBMS and, if supported, have been assigned to the user. Both types of information is recorded in audit trails that keep track of accesses to relations. Using this information, it is then possible to extract sequences of operations from Access(R, ti , tj) that are delimited by role changes, that is, in a session, the user switched roles. For example, the first part of a particular user session may contain 10 operations the user executed with role ro1 and then 20 operations with role ro2, all operations against relation R. Extracting all information relevant to a user u can easily be done using SQL query statements against Access Prof (R, ti , tj).

In general, the above techniques show that it is important to determine precise metrics of interest for the user profiling approach. That is, one has to establish clear objectives that can be computed from a data access profile. For example, if the time window underlying a data access profile AccessProf(R, ti , tj) covers a whole week from Monday to Sunday, then the profile of a user u with respect to the relation R, denoted, UserProf(R, u, ti , tj) may include information about the following measures: (1) number and duration of sessions, (2) names of roles that have been enabled during a session, including timestamps of when roles where enabled, (3) number of operations against R per session and role setting, (4) typical sequence of operations in a session (per role), (5) typical values of attributes inserted or modified, and typical properties of tuples deleted. As done for the other types of profiles introduced in the previous sections, a user profile UserProf(R, u, ti , tj) can be managed as measure-value pairs (in auxiliary relations) for easy inspection and use by security mechanisms. It should also be noted that the above tasks can be extended to capture information about the execution of stored procedures by a user.

Access Path Model:

A typical production-type database setup may contain thousands of database objects, hundreds of users and roles and, consequently, many complex access privilege structures. Furthermore, several applications may operate on a single database and its objects using different accounts and privileges. In order to apply a security re-engineering approach to such a complex setting, it is essential to have a good methodology that helps administrators and security personnel to suitably approach the tasks of data and user profiling, analysis and correlation of profiles, and re-design of security policies and mechanisms.

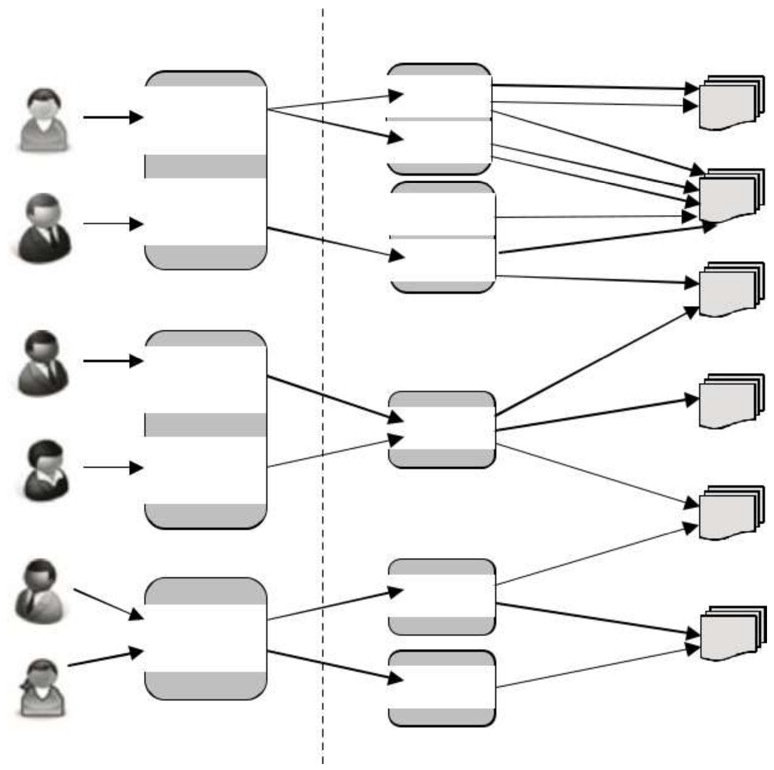
Problem Setting and Objectives:

As discussed in the previous sections, if erroneous or anomalous data have been discovered, one would like to identify the user(s) who operated on these data. There are several aspects that make it very difficult to establish such correlations. First, a complex information system infrastructure can consist of multiple layers, typically several applications on top of a single database, with numerous users at both the application and database layer, including persons, application users, and database users. Most approaches to anomaly and misuse detection assume that the notion of a user is well-defined, typically a user directly operating on the data. However, what precisely constitutes a user in a more complex setting such as outlined above? A person, an application account, or a database account (possibly having several database roles)? What if users and/or applications share accounts? How can accesses to the data be traced back to users? As accesses to the data occur through several layers, starting with a person or application, which, in turn, performs operations on the database, correlating anomalous data and data behavior with a person is not a trivial task. However, in order to adhere to the principle of accountability, being able to determine such correlations is a must.

To address these problems, we propose the access path model. The objective of the access path model is to help administrators and security personnel in a focused re-engineering approach to database systems. This is accomplished by a methodology to describe, annotate, explore, and correlate so-called access paths. An access path, which will be described more formally below, basically specifies in which way a person operates on the data managed in a DBMS. Different components of an access path are annotated with data and user profiles and allow for an easy comparison of access correlations at the different layers of access. The access path model provides a comprehensive framework and methodology for a security re-engineering approach to databases, with a focus on misuse and anomaly detection driven by data and user profiles.

Model Components:

The access path model consists of several components that help describing correlations between users and accesses at different layers. Figure 3 gives an overview of the basic components, with applications and a single database as back-end being the core components.



In this figure, there are three applications and several persons who have access to the applications based on some application accounts. We assume that a person does not directly connect to the database (e.g., at the database server) but through an application once the she has been authenticated at the application layer. We also make this assumption for DBAs, who operate on the database only using administrative tools (applications). The connection to the database occurs through database user accounts, which then, based on the database privileges assigned to the accounts, are allowed to perform operations on database objects. For the sake of simplicity, we only consider operations on database relations, which are shown on the right. One could also include other types of database objects, such as views or stored procedures.

The figure shows several cases that are typical in real-world settings. We first discuss these cases in more detail, and then elaborate on how an instance of the access path model is obtained before we cover some more formal properties of the model.

Application Layer Figure 3 shows three applications A1, A2, and A3 to which some persons have access. At application A1, each user has a separate application account, and each application account is associated with a database user account. At application A2, again each user has a separate application account, but the application uses only one database account. For application A3, two persons share an application account, and the application uses different database accounts. An important observation from the latter two cases is that accounts are shared, which obviously causes problems in correlating data accesses to a particular person.

Database Layer with each database user account, one or more database roles are associated. For application A1, each application user account corresponds to a database user account, and each database user account has two roles. For application A2, there is only one (default) role for the database account through which all accesses to the database occur.

Gathering information about sub-paths from persons to database user accounts and roles and representing these access structures and access correlations to administrators can be supported by some kind of security reengineering workbench that, among other tasks such as the selective profiling of data and users, allows visualizing different access paths. This simple representation of access information can lead to some interesting insights, all of which relate to the discovery of potential vulnerabilities in the current security setup of the applications and the database. These include

- unused application accounts for which no person exists who makes use of that account,
- unused database accounts for which no application account exists that uses the database account, and
- unused database roles that have not been assigned to any user or have no privileges associated with them

Annotating and Exploring Access Paths:

Assume a partially instantiated access path model for some applications and database where no access correlations between database accounts/roles and database objects have been made yet. In order to focus on evaluating the security of mission-critical and sensitive database objects and relations in particular, only respective relations are considered first in exploring access correlations. Is is assumed that for a given database schema such relations can be identified based on existing security policies and regulations. In the following, let DBobj denote only such critical relations. Now, for each relation R ∈ DBobj , a snapshot profile DataP rof(R, ti) is determined at time ti . This process follows exactly the tasks presented in Section 3.2. Respective profiles are inspected, evaluated and subsequently access profiles AccessP rof(R, ti , tj) are established for a time window [ti , tj].

The profiles obtained in this way are now used to annotate components of the access path model instance comprised of vertices (representing accounts, roles, and objects) and edges (representing access correlations). Each relation R ∈ DBobj is annotated with its snapshot profile DataP rof(R, ti) and access profile AccessP rof(R, ti , tj). It is assumed that a snapshot profile is captured at the same time ti the collection of access information for R during the time window [ti , tj] is initiated. From an access profile AccessP rof(R, ti , tj) and its user profiles UserP rof(R, udb, ti , tj) then access sub-paths from database accounts/roles to the relation R are established. Each such sub-path ((udb −>rdb) −→ R) is annotated with access information specific to the database user udb and role rdb, respectively. The process is repeated for the other relations in DBobj until all sub-paths from database accounts/roles to the relations DBobj have been annotated with respective access information.

Security Reconfiguration:

We now detail and summarize some basic tasks to reconfigure the security setting and mechanisms of a database system using the information obtained from data and user profiling and access path correlations presented in the previous sections. We assume that appropriate (coordinated) authentication and auditing mechanisms are in place at the application and database layer.

The objective of the reconfiguration of security mechanisms is to constrain the behavior of database users and roles such they have exactly those privileges necessary and sufficient to perform their tasks. In other words, the mechanisms should realize the principle of least privilege. The problem with most security mechanisms in database systems in achieving this objective, however, is that they basically rely solely on the SQL grant statement, which in general does not allow specifying fine-grained access policies. This means, for example, that if a user is only allowed to insert tuples that have some well-defined properties, this admissible behavior cannot be specified using the grant statement. In general, this leaves plenty of room for malicious or accidental misuse of privileges by insiders.

Integrity Constraints: As discussed in Section 3.2, semantic integrity constraints provide an effective mechanism to constrain attribute values and combinations thereof to admissible ones. Most static integrity constraints can be implemented easily, e.g., through database triggers or check clauses in relation schemas. These mechanisms can be used to prevent inadmissible attribute values a user tries to (maliciously) insert or modify. Recall that such mechanisms also have the side-effect of helping maintaining the quality of the data, an important prerequisite for evaluating the security of a database.

Unused Accounts and Roles: The access path model provides a framework to detect unused database accounts and roles. In particular unused accounts are vulnerable to misuse and thus should be locked or simply deleted. Similarly, unused roles should be deleted, because they likely implement obsolete policies. The deletion of roles can also have implications on the re-design of role-hierarchies.

Unused Privileges: As indicated at the end of Section 4, unused privileges can be detected once access sub-paths have been annotated with respective access profiles. As the content of an access profile clearly depends on the time window for which information has been collected, care has to be taken to ensure that the suspected unused privilege is not used only on rare occasions.

Discovery and Re-design of Database Roles: Database roles help in administering privileges associated with complex tasks [24, 45], i.e., activities by different types of application users. Assume that for a set of missioncritical relations data and user access profiles have been determined. Respective access paths in the access path model can now be analyzed to determine whether there are similarities among accesses to the relations by these different database users. If no roles (or only some default role) have been associated with these users in the context of respective accesses, database roles (and role hierarchies) can be introduced to better capture similar privileges used by these accounts and to ease the management of privileges.

Derivation of Database Views: One of the most effective mechanisms to achieve the principle of least privilege is to have database views, and to give database users privileges on these views rather than the underlying relations. The derivation of such views is not an easy task and requires a careful analysis of access profiles. Assume a database user udb who has been granted select, insert, and update privileges on a relation R. The access profile for udb might show that the user only selects certain types of tuples and the tuples she modifies have similar characteristics. The description of these tuples, represented in the user profile for udb, then can be used to derive a view V that contains only tuples the user typically operates on. The privileges udb has on R are then revoked, and the user is assigned respective privileges on the view V. More complex views can be derived as well for SQL select statements that refer to more than one relation.

Stored Procedures: A major limitation of discretionary access control models in relational databases is that SQL grant statements for assigning privileges to users and roles are often too coarse-grained. Some of these limitations can be circumvented by view privileges instead of privileges on base relations, as outlined above. However, oftentimes granting access to some relations (and views) depends on context information that cannot easily be specified in a view definition. Such context information might include concurrent database user sessions, time information, and the origin of database requests (e.g., application hosts). There has been some substantial work on more expressive access control models that take such context information into account and therefore provide more dynamic access control mechanisms.

Database Watermarking for Copyright Protection:

As increasing amounts of data are produced, packaged and delivered in digital form, in a fast, networked environment, one of its main features threatens to become its worst enemy: zero-cost verbatim copies. The ability to produce duplicates of digital Works at almost no cost can now be misused for illicit profit. This mandates mechanisms for effective rights assessment and protection.

One such mechanism is based on Information Hiding. By concealing a resilient rights holder identity "signature" (watermark) within the digital Work(s) to be protected, Information Hiding for Rights Assessment (Watermarking) enables ulterior court-time proofs associating particular Works with their respective rights holders. One main challenge is the fact that altering the Work in the process of hiding information could possibly destroy its value. At the same time one has to be concerned with a malicious adversary, with major incentives to remove or alter the watermark beyond detection – thus disabling the ability for court-time proofs – without destroying the value of the Work – to preserve its potential for illicit profit.

Introduction

Mechanisms for privacy assurances (e.g., queries over encrypted data) are essential to a viable and secure management solution for outsourced data. On a somewhat orthogonal dimension but equally important, we find the requirement to be able to assert and protect rights over such data. Different avenues are available, each with its advantages and drawbacks. Enforcement by legal means is usually ineffective, unless augmented by a digital counterpart such as Information Hiding. Digital Watermarking as a method of Rights Assessment deploys Information Hiding to conceal an indelible "rights witness" ("rights signature", watermark) within the digital Work to be protected (see Figure 1). The soundness of such a method relies on the assumption that altering the Work in the process of hiding the mark does not destroy the value of the Work, while it is difficult for a malicious adversary ("Mallory") to remove or alter the mark beyond detection without doing so. The ability to resist attacks from such an adversary, mostly aimed at removing the watermark, is one of the major concerns in the design of a sound solution.

Model:

Before we proceed however, let us first understand how the ability to prove rights in court relates to the final desiderata, namely to protect those rights. After all, doesn't simply publishing a summary or digest of the Work to be protected – e.g., in a newspaper, just before releasing the Work – do the job? It would seem it enables one to prove later in court that (at least a copy of) the Work was in one's possession at the time of release. In the following we address these and other related issues. 2.1 Rights Protection through Assess.

Rights Protection through Assessment

The ability to prove/assess rights convincingly in court constitutes a deterrent to malicious Mallory. It thus becomes a tool for rights protection if counterincentives and legal consequences are set high enough. But because Information Hiding does not provide a means of actual access control, the question of rights protection still remains. How are rights protected here? It is intuitive that such a method works only if the rightful rights-holder (Alice) actually knows about Mallory's misbehavior and is able to prove to the court that: (i) Mallory possesses a certain Work X and (ii) X contains a "convincing" (e.g., very rare with respect to the space of all considered similar Works) and "relevant" watermark (e.g., the string "(c) by Alice"). What watermarking itself does not offer is a direct deterrent. If Alice does not have knowledge of Mallory's illicit possession of the Work and/or if it is impossible to actually prove this possession in court beyond reasonable doubt, then watermarking cannot be deployed directly to prevent Mallory. If, however, Information Hiding is aided by additional access control levels, it can become very effective.

Newspaper Digests:

To achieve the above however, Alice could publish a summary or digest (e.g., cryptographic hash) of X in a newspaper, thus being able to claim later on at least a time-stamp on the possession of X. This could apparently result in a quite effective, albeit costly, alternative to Watermarking the Work X. There are many simple reasons why it would not work, including (i) scalability issues associated with the need for a trusted third party (newspaper), (ii) the cost of publishing a digest for each released Work, (iii) scenarios when the fact that the Work is watermarked should be kept secret (stealthiest) etc.
Maybe the most important reason however, is that Mallory can now claim that his ownership of the Work precedes X's publication date, and that Alice simply modified it (i.e., a stolen copy) and published a digest thereof herself. It would then be up to the court to decide if Mallory is to be believed or not, hardly an encouraging scenario for Alice. This could work if there existed a mechanism for the mandatory publication of digests for each and every valuable Work, again quite likely impractical due to both costs and lack of scalability to a virtually infinite set of data producers and Works.
Deploying such aids as rights assessment tools makes sense only in the case of the Work being of value only un-modified. In other words if it does not tolerate any changes, without losing its value, and Mallory is caught in possession of an identical copy, Alice can successfully prove in court that she possessed the original at the time of its publication (but she cannot prove more). Considering that, in the case of watermarking, the assumption is that, no matter how small, there are modifications allowed to the Works to be protected, in some sense the two approaches complement each other. If no modifications are allowed, then a third-party "newspaper" service might work for providing a time-stamp type of ownership proof that can be used in court.

Steganography and Watermarking:

There exists a fundamental difference between Watermarking and generic Information Hiding (steganography) from an application perspective and associated challenges. Information Hiding in general (and covert communication in particular), aims usually at enabling Alice and Bob to exchange messages in a manner as resilient and stealthy as possible, through a hostile medium where Malory could lurk. On the other hand, Digital Watermarking is deployed by Alice as a court proof of rights over a Work, usually in the case when Mallory benefits from using/selling that very same Work or maliciously modified versions of it.

In Digital Watermarking, the actual value to be protected lies in the Works themselves whereas pure steganography usually makes use of them as simple value "transporters". In Watermarking, Rights Assessment is achieved by demonstrating (with the aid of a "secret" known only to Alice – "watermarking key") that a particular Work exhibits a rare property ("hidden message" or "watermark"). For purposes of convincing the court, this property needs to be so rare that if one considers any other random Work "similar enough" to the one in question, this property is "very improbable" to apply (i.e., bound false-positives rate). It also has to be relevant, in that it somehow ties to Alice (e.g., by featuring the bit string "(c) by Alice").

There is a threshold determining the ability to convince the court, related to the "very improbable" assessment. This defines a main difference from steganography: from the court's perspective, specifics of the property (e.g., watermark message) are not important as long as they link to Alice (e.g., by saying "(c) by Alice") and, she can prove "convincingly" it is she who induced it to the (non-watermarked) original.

Consumer Driven Watermarking:

An important point about watermarking should be noted. By its very nature, a watermark modifies the item being watermarked: it inserts an indelible mark in the Work such that (i) the insertion of the mark does not destroy the value of the Work, i.e., it is still useful for the intended purpose; and (ii) it is difficult for an adversary to remove or alter the mark beyond detection without destroying this value. If the Work to be watermarked cannot be modified without losing its value then a watermark cannot be inserted. The critical issue is not to avoid alterations, but to limit them to acceptable levels with respect to the intended use of the Work.

Thus, an important first step in inserting a watermark, i.e., by altering it, is to identify changes that are acceptable. Naturally, the nature and level of such change is dependent upon the application for which the data is to be used. Clearly, the notion of value or utility of the data becomes thus central to the watermarking process. For example, in the case of software, the value may be in ensuring equivalent computation, whereas for natural language text it may be in conveying the same meaning – i.e., synonym substitution is acceptable. Similarly, for a collection of numbers, the utility of the data may lie in the actual values, in the relative values of the numbers, or in the distribution (e.g., normal with a certain mean). At the same time, the concept of value of watermarked Works is necessarily relative and largely influenced by each semantic context it appears in. For example, while a statistical analyst would be satisfied with a set of feature summarizations (e.g., average, higherlevel moments) of a numeric data set, a data mining application may need a majority of the data items, for example to validate a classification hypothesis.

It is often hard to define the available “bandwidth” for inserting the watermark directly. Instead, allowable distortion bounds for the input data can be defined in terms of consumer metrics. If the watermarked data satisfies the metrics, then the alterations induced by the insertion of the watermark are considered to be acceptable. One such simple yet relevant example for numeric data, is the case of maximum allowable mean squared error (MSE), in which the usability metrics are defined in terms of mean squared error tolerances as (si − vi) 2 < ti , ∀i = 1, ..., n and (si − vi) 2 < tmax, where S = {s1, ..., sn} ⊂ R, is the data to be watermarked, V = {v1, ..., vn} is the result, T = {t1, ..., tn} ⊂ R and tmax ∈ R define the guaranteed error bounds at data distribution time. In other words T defines the allowable distortions for individual elements in terms of MSE and tmax its overall permissible value.

Often however, specifying only allowable change limits on individual values, and possibly an overall limit, fails to capture important semantic features associated with the data – especially if the data is structured. Consider for example, age data. While a small change to the age values may be acceptable, it may be critical that individuals that are younger than 21 remain so even after watermarking if the data will be used to determine behavior patterns for under-age drinking. Similarly, if the same data were to be used for identifying legal voters, the cut-off would be 18 years. Further still, for some other application it may be important that the relative ages, in terms of which one is younger, not change. Other examples of constraints include: (i) uniqueness – each value must be unique; (ii) scale – the ratio between any two number before and after the change must remain the same; and (iii) classification – the objects must remain in the same class (defined by a range of values) before and after the watermarking. As is clear from the above examples, simple bounds on the change of numerical values are often not enough.

Structured collections, present further constraints that must be adhered to by the watermarking algorithm. Consider a data warehouse organized using a standard Star schema with a fact table and several dimension tables. It is important that the key relationships be preserved by the watermarking algorithm. This is similar to the “Cascade on update” option for foreign keys in SQL and ensures that tuples that join before watermarking also join after watermarking. This requires that the new value for any attribute should be unique after the watermarking process. In other words, we want to preserve the relationship between the various tables. More generally, the relationship could be expressed in terms of an arbitrary join condition, not just a natural join. In addition to relationships between tuples, relational data may have constraints within tuples. For example, if a relation contains the start and end times of a web interaction, it is important that each tuple satisfies the condition that the end time be later than the start time.

Naturally, one can always identify some use that is affected by even a minor change to any portion of the data. It is therefore important that (i) the main intended purpose and semantics that should be preserved be identified during watermarking and that (ii) the watermarking process not interfere with the final data consumer requirements. We call this paradigm consumer driven watermarking. Some of the solutions discussed here are consumer driven enabled through feedback mechanisms (see Figure 2) that allow the watermarking process to “rollback” modifications that would violate quality constraints in the result on a step by step basis. This ensures the preservation of desired quality metrics with respect to the original un-watermarked input Work.

Discrete Data vs. Multimedia

Information Hiding and Watermarking in frameworks such as signal processing and multimedia (e.g., images, video and audio). Here we explore Information Hiding as a rights assessment tool for discrete data types. Because, while the terms might be identical, the associated models, challenges and techniques are different, almost orthogonal. Whereas in the signal processing case there usually exists a large noise bandwidth, due to the fact that the final data consumer is likely human – with associated limitations of the sensory system – in the case of discrete data types this cannot be assumed and data quality assessment needs to be closely tied with the actual watermarking process.

Another important differentiating focus is the emphasis on the actual ability to convince in court as a success metric, unlike most approaches in the signal processing realm, centered on bandwidth. While bandwidth is a relevant related metric, it does not consider important additional issues such as malicious transforms and removal attacks. For rights assertion, the concerns lie not as much with packing a large amount of information (i.e., watermark bits) in the Works to be protected, as with being able to both survive removal attacks and convince in court.

Maybe the most important difference between the two domains is that, while in a majority of watermarking solutions in the multimedia framework, the main domain transforms are signal processing primitives (e.g., Works are mainly considered as being compositions of signals rather than strings of bits), in our case data types are mostly discrete and are not naturally handled as continuous signals. Because, while discrete versions of frequency transforms can be deployed as primitives in information encoding for digital images [8], the basis for doing so is the fact that, although digitized, images are at the core defined by a composition of light reflection signals and are consumed as such by the final human consumer. By contrast, arbitrary discrete data is naturally discrete 1 and often to be ingested by a highly sensitive semantic processing component, e.g., a computer rather than a perceptual system tolerant of distortions.

Relational Data

In such a model, relations between information items are explicitly specified: data is organized as “a number of differently sized tables” composed of “related” rows/columns. A table is a collection of rows or record and each row in a table contains the same fields. Certain fields may be designated as data keys (not to be confused with “cryptographic keys”) when a functional dependency or key constraint, holds for the table. Often, indexing is deployed to speed up searches on values of such primary key fields. Data is structured logically into valued attributes. From this perspective, a table is a collection of such attributes (the columns of the table) and models a relation among them. The data rows in the tables are also called tuples. Data in this model is manipulated using a relational algebra. Main operations in this algebra are set operations (e.g., union, intersection, and Cartesian product), selection (of some tuples in tables) and projection (of some columns/attributes).

Rights protection for such data is important in scenarios where it is sensitive, valuable and about to be outsourced. A good example is a data mining application, where data is sold in pieces to parties specialized in mining it, e.g., sales patterns database, oil drilling data, financial data. Other scenarios involve for example online B2B interactions, e.g., airline reservation and scheduling portals, in which data is made available for direct, interactive use (see Figure 3). Given the nature of most of the data, it is hard to associate rights of the originator over it. Watermarking can be used to solve this issue.

The Adversary

Watermarking is a game between the watermarker and malicious Mallory. In this game, the watermarker and Mallory play against each other within subtle trade-off rules aimed at keeping the quality of the result within acceptable bounds. It is as if there exists an impartial referee (the data itself) moderating each and every “move”. As discussed above, it is important to make this “referee” an explicit part of the marking process (consumer-driven paradigm). It is also important to understand Mallory and the adversarial setting.

Once outsourced, i.e., out of the control of the watermarker, data might be subjected to a set of attacks or transformations; these may be malicious – e.g., with the explicit intent of removing the watermark – or simply the result of normal use of the data. An effective watermarking technique must be able to survive such use. In a relational data framework important attacks and transformations are:

A1. Sampling. The attacker (Mallory) can randomly select and use a subset of the watermarked data set that might still provide value for its intended purpose (“subset selection”). More specifically, here we are concerned with both (A1.a) horizontal and (A1.b) vertical data partitioning – in which a valuable subset of the attributes are selected by Mallory.

A2. Data Addition. Mallory adds a set of tuples to the watermarked set. This addition is not to significantly alter the useful properties of interest to Mallory.

A3. Alteration. Altering a subset of the items in the watermarked data set such that there is still value associated with the result. In the case of numeric data types, a special case needs to be outlined here, namely (A3.a) a linear transformation performed uniformly to all of the items. This is of particular interest as it can preserve significant valuable data-mining related properties of the data.

A4. Ulterior Claims of Rights. Mallory encodes an additional watermark in the already watermarked data set and claims rights based upon this second watermark.

A5. Inevitability Attack. Mallory attempts to establish a plausible (watermark, key) pair that matches the data set and then claims rights based on this found watermark.

Given the attacks above, several properties of a successful solution surface. For immunity against A1, the watermark has to be likely encoded in overall data properties that survive sampling, e.g., confidence intervals, statistical bias. With respect to (A1.b) special care has to be taken such that the mark survives this partitioning. The encoding method has to feature a certain attribute-level property that could be recovered in such a vertical partition of the data. We believe that while vertical data partitioning attacks are possible and also very likely in certain scenarios, often value is to be found in the association between a set of relation attributes. These attributes are highly likely to survive such an attack, as the final goal of the attacker is to produce a still-valuable result. If the assumption is made that the attack alterations do not destroy the value of the data, then A3 could be handled by encoding the primitive mark in resilient global data properties. As a special case, A3.a can be resisted by a preliminary normalization step in which a common divider to all the items is first identified and applied.

While powerful, for arbitrary watermarks, the invertibility attack A5 can be defeated by requiring the encoded string to be relevant (e.g. "(c) by Mallory") and the encoding to be "convincing" (see Section 2.1). Then the probability of success of invertibility searches becomes upper bound. In order to defeat A4, the watermarking method has to provide the ability to determine encoding precedence, e.g., if it can be proved in court that one watermark encoding was "overwritten" by a later one. Additionally, in the case of such a (court time) dispute, the parties could be requested to present a portion of the original, un-watermarked data. Only the rightful rights holder would be able to produce such a proof, as Mallory could only have access to already watermarked data.

It is worth also noting that, intuitively, if, in the process of watermarking, the data is altered to its usability limits, any further alteration by a watermarker is likely bound to yield an unusable result. Achieving this might be often desirable 2 and has been explored by Sion et. al. in a proof of concept implementation [34] as well as by Li et. al. in [20] (this is discussed in more detail elsewhere in this book). The challenges of achieving such a desiderata however, lies in the impossibility to define absolute data quality metrics that consider all value dimensions of data.

Numeric Types:

In this section we explore watermarking solutions in the context of relational data in which one or more of the attributes are of a numeric type. Among existing solutions we distinguish between single-bit (the watermark is composed of a single bit) and multi-bit (the watermark is a string of bits) types. Orthogonally, the encoding methods can be categorized into two; we chose to call them direct-domain and distribution encodings. In a direct-domain encoding, each individual bit alteration in the process of watermarking is directly correlated to (a part of) the encoded watermark. In distribution encodings, the encoding channel lies often in higher order moments of the data (e.g., running means, hierarchy of value averages). Each individual bit alteration impacts these moments for the purpose of watermark encoding, but in itself is not directly correlated to any one portion of the encoded watermark.

Single Bit Direct Domain Encoding:

Kiernan, Agrawal et.al. Propose a direct domain encoding of a single bit watermark in a numeric relational database.

Overview. Its main algorithm proceeds as follows. A subset of tuples are selected based on a secret criteria; for each tuple, a secret attribute and corresponding least significant (ξ) bit position are chosen. This bit position is then altered according to yet other secret criteria that is directly correlated to the watermark bit to be encoded. The main assumption is, that changes can be made to any attribute in a tuple at any least significant ξ bit positions. At watermark detection time, the process will re-discover the watermarked tuples and, for each detected accurate encoding, become more "confident" of true-positive detection.

There are a set of important assumptions underlying this method. Maybe the most important one is that "the relational table being watermarked is such that if all or a large number of the ξ least significant bits of any attribute are dropped or perturbed, then the value of the data is significantly reduced. However, it is possible to change a small number of the bits and not decrease the value of the data significantly"

The authors make an argument for this being a reasonable assumption as such techniques have been used by publishers of books of mathematical tables for a long time – e.g., by introducing small errors in published logarithm tables and astronomical ephemerides to identify pirated copies [15]. Examples of real-world data sets that satisfy such an assumption are given, including tables of parametric specifications (mechanical, electrical, electronic, chemical, etc.), surveys (geological, climatic, etc.), and life sciences data (e.g., gene expression).

Solution Details. For consistency, the original notation is used: a database relation R with the following schema is R(P, A0,...,Aν−1), is assumed, with P the primary key attribute. All ν attributes A0,...,Aν−1 are candidates for marking: the values are assumed such that small changes in the ξ least significant bits are imperceptible. γ denotes a control parameter that determines the average number ω of tuples marked ($\omega = \eta\ \gamma$), where η is the number of tuples in the database. r. X is used to denote the value of attribute X in tuple r, α denotes a "significance level" and τ a "threshold" for the test of "detecting a watermark". K is a key known only to the database owner, and there exists G, a pseudo-random sequence number generator [23] (next (G) denotes the next generated sequence number) Watermark insertion is illustrated in Figure 4. The main steps of the algorithm are as follows. Initially (step 2) the random sequence generator is initialized such that its output is distinct for any given distinct tuple value. This mechanism is deployed in order to achieve a certain tuple ordering independence of the encoding. The output of G is then used to determine: (if

```
1) foreach tuple r ∈ R do
2)   seed G with r.P concatenated with K
3)   if (next(G) mod γ = 0) then // mark this tuple
4)      attribute index i = next(G) mod v // mark attribute A_i
5)      bit index j = next(G) mod η // mark j^th bit
6)      r.A_i = mark(next(G),r.A_i ,j)
7) mark(random number i, value v, bit index j) return value
8)   if (i is even) then
9)      set the j^th least significant bit of v to 0
10)  else
11)     set the j^th least significant bit of v to 1
12)  return v
```

Fig. 4. Watermark insertion for the single-bit encoding of [1, 16]

the current tuple is to be watermarked (step 3), (ii) which attribute value to mark (step 4), (iii) which bit within that attribute's value to alter (step 5), and (iv) what new bit-value to assign to that bit-position in the result (step 6, invocation of mark()). This encoding guarantees that, in order to entirely remove a watermark, Mallory is put in the position of guessing correctly the marked tuples, attributes and altered bit positions. Once R is published, the data owner, Alice, would like to determine whether the (similar) relation S published by Mallory has been pirated from R. The sets of tuples and of attributes in S are assumed to be strict subsets of those in R. Additionally, Mallory is assumed not to drop the primary key attribute or change the value of primary keys. Then watermark detection is a direct inverse of insertion. It proceeds as follows:

```
1) totalcount = matchcount = 0
2) foreach tuple s ∈ S do
3)   seed G with s.P concatenated with K
4)   if (next(G) mod γ = 0) then // tuple was marked
5)      attribute index i = next(G) mod v // A_i was marked
6)      bit index j = next(G) mod η // j^th bit was marked
7)      totalcount = totalcount + 1
8)      matchcount = matchcount + match (next(G,s.A_i ,j)
9) τ = threshold(totalcount,α)
10)       if ((matchcount < τ ) or (matchcount > totalcount - τ )) then
11)       suspect piracy
12)       match(random number i, value v, bit index j) return integer
13)  if (i is even) then
14)     return 1 if the j^th least significant bit of v is 0 else return 0
15)  else
16)     return 1 if the j^th least significant bit of v is 1 else return 0
```

Fig. 5. Watermark detection for the single-bit encoding of [1, 16]

Multi-Bit Watermarks:

While there likely exist applications whose requirements are satisfied by singlebit watermarks, often it is desirable to provide for “relevance”, i.e., linking the encoding to the rights holder identity. This is especially important if the watermark aims to defeat against invertibility attacks (A5).

In a single-bit encoding this cannot be easily achieved. Additionally, while the main proposition of watermarking is not covert communication but rather rights assessment, there could be scenarios where the actual message payload is of importance. One apparent direct extension from single-bit watermarks to a multi-bit version would be to simply deploy a different encoding, with a separate watermark key, for each bit of the watermark to be embedded. This however, might not be possible, as it will raise significant issues of inter-encoding interference: the encoding of later bits will likely distort previous ones. This will also make it harder to handle ulterior claim of rights attacks (A4).

Multi-Bit Direct Domain Encoding:

The scheme functions as follows. The database is parsed and, at each bit-encoding step, one of the watermark bits is randomly chosen for embedding; the solution in [1,16] is then deployed to encode the selected bit in the data at the “current” point. The “strength of the robustness” of the scheme is claimed to be increased with respect to [1, 16] due to the fact that the watermark now possesses an additional dimension, namely length. This should guarantee a better upper bound for the probability that a valid watermark is detected from unmarked data, as well as for the probability that a fictitious secret key is discovered from pirated data (i.e., invertibility attacks A5). This upper bound is said to be independent of the size of database relations thus yielding robustness against attacks that change the size of database relations.

Multi-Bit Distribution Encoding:

Encoding watermarking information in resilient numeric distribution properties of data presents a set of advantages over direct domain encoding, the most important one being its increased resilience to various types of numeric attacks. In [27, 29, 30, 32, 33] and [34], Sionet. al. introduce a multi-bit distribution encoding watermarking scheme for numeric types. The scheme was designed with both an adversary and a data consumer in mind. More specifically the main desiderata were: (i) watermarking should be consumer driven – i.e., desired semantic constraints on the data should be preserved – this is enforced by a feedback-driven rollback mechanism, and (ii) the encoding should survive important numeric attacks, such as linear transformation of the data (A3.a), sampling (A1) and random alterations (A3).

Overview. The solution starts by receiving as user input a reference to the relational data to be protected, a watermark to be encoded as a copyright proof, a secret key used to protect the encoding and a set of data quality constraints to be preserved in the result. It then proceeds to watermark the data while continuously assessing data quality, potentially backtracking and rolling back undesirable alterations that do not preserve data quality. Watermark encoding is composed of two main parts: in the first stage, the input data set is securely partitioned into (secret) subsets of items; the second stage then encodes one bit of the watermark into each subset. If more subsets (than watermark bits) are available, error correction is deployed to result in an increasingly resilient encoding. Each single bit is encoded/ represented by introducing a slight skew bias in the tails of the numeric distribution of the corresponding subset. The encoding is proved to be resilient to important classes of attacks, including subset selection, linear data changes and random item(s) alterations

Solution Details. The algorithm proceeds as follows (see Figure 6): (a) Userdefined queries and associated guaranteed query usability metrics and bounds are specified with respect to the given database (see below). (b) User input determines a set of attributes in the database considered for watermarking, possibly all. (c) From the values in each such attribute select a (maximal) number of (e) unique, non-intersecting, secret subsets. (d) For each considered subset, (d.1) embed a watermark bit into it using the single-bit encoding convention described below and then (d.2) check if data constraints are still satisfied. If data constraints are violated, (d.3) retry different encoding parameter variations or, if still no success, (d.4) try to mark the subset as invalid (see single-bit encoding convention below), or if still no success (d.5) ignore the current set3 . Repeat step (d) until no more subsets are available.

Several methods for subset selection (c) are discussed. In one version, it proceeds as follows. The input data tuples are sorted (lexicographically) on a secret keyed cryptographic hash H of the primary key attribute (K). Based on this value, compose a criteria (e.g., H(K, key)) mod e = 0) for selecting a set of "special" tuples such that they are uniformly distributed and average a total number of e = length(attribute)/ subset size. These special tuples are going to be used as subset "markers". Each subset is defined as the elements between two adjacent markers, having on average subset size elements. The detection phase will then rely on these construction criteria to re-discover the subset markers. This process is illustrated in Figure 6.

```
wm(attribute, wm key, mark data[],
    plugin handler, db primary key, subset size, $v_{false}$ , $v_{true}$ , c)
    sort attribute ← sort on normalized hash(wm key,db primary key,wm key)
    for (i=0; i < $\frac{\text{length(attribute)}}{\textbf{subset size}}$ ;i++)
        subset bin ← next subset size elements from sort attribute compute rollback data
        encode(mark data[i % mark data.length], subset bin, $v_{false}$ , $v_{true}$ , c) propagate changes into attribute
        if (not goodness plugin handler.isSatisfied(new data,changes)) then rollback rollback data
            continue
        else
            commit
            map[i] = true
            subset boundaries[i] = subset bin[0] return map, subset boundaries
```

Fig. 6. Watermark Embedding (version using subset markers and detection maps shown).

Encoding the individual mark bits in different subsets increases the ability to defeat different types of transformations including sampling (A1) and/or random data addition (A2), by "dispersing" their effect throughout the data, as a result of the secret ordering. Thus, if an attack removes 5% of the items, this will result in each subset Si being roughly 5% smaller. If Si is small enough and/or if the primitive watermarking method used to encode parts of the watermark (i.e., 1 bit) in Si is made resilient to these kind of minor transformations then the probability of survival of most of the embedded watermarks is accordingly higher. Additionally, in order to provide resilience to massive "cut" attacks, the subsets are made to be of sizes equal to a given percent of the overall data set, i.e., not of fixed absolute sizes.

Once constructed, each separate subset Si will be marked separately with a single bit, in the order it appears in the actual watermark string. The result will be a e-bit (i.e., i = 1,...,e) overall watermark bandwidth in which each bit is "hidden" in each of the marked Si . If the watermark is of size less than e, error correction can be deployed to make use of the additional bandwidth to increase the encoding resilience.

The single-bit distribution encoding proceeds as follows. Let b be a watermark bit that is to be encoded into Si and G represent a set of user specified change tolerance, or usability metrics. The set G will be used to implement the consumer-driven awareness in the watermark encoding.

Let vfalse, vtrue, c ∈ (0, 1), vfalse < vtrue be real numbers (e.g., c = 90%, vtrue = 10%, vfalse = 7%). c is called confidence factor and the interval (vfalse, vtrue) confidence violators hysteresis. These are values to be remembered also for watermark detection time. They can be considered as part of the encoding key. Let avg(Si) and δ(Si) be the average and standard deviation, respectively, of Si . Given Si and the real number c ∈ (0, 1) as above, vc(Si) is defined as the number of items of Si that are greater than avg(Si) +c×δ(Si). vc(Si) is called the number of positive "violators" of the c confidence over Si.

The single-bit mark encoding convention is then formulated: given Si , c, vfalse and vtrue as above, mark(Si) ∈ {true, f alse, invalid} is defined to be true if vc(Si) > (vtrue × |Si |), f alse if vc(Si) < vfalse × |Si | and invalid if vc(Si) ∈ (vfalse × |Si |, vtrue × |Si |). In other words, the watermark is modeled by the percentage of positive confidence violators present in Si for a given confidence factor c and confidence violators hysteresis (vfalse, vtrue). Encoding the single bit (see Figure 8), b, into Si is therefore achieved by minor alterations to some of the data values in Si such that the number of positive violators (vc(Si)) is either (a) less than vfalse × |Si | if b = 0, or (b) more than vtrue × |Si | if b = 1. The alterations are then checked against the change tolerances, G, specified by the user.

```
encode(bit, set, $v_{false}$, $v_{true}$, c)
    compute avg(set), $\delta$(set)
    compute $v_c$ (set)
    if $v_c$ (set) satisfies desired bit value return **true** if (bit)
        compute $v_* \leftarrow v_{true} - v_c$ (set)
        alter $v_*$ items close to the stddev boundary so that they become > $v_{true}$ else
        (!bit) case is similar
    compute $v_c$ (set)
    if $v_c$ (set) satisfies desired bit value return **true**
    else rollback alterations (distribution shifted too much?)
    return **false**
```

Fig. 8. Single Bit Encoding Algorithm (illustrative overview)

At detection time the secret subsets are reconstructed and the individual bits are recovered according to the single-bit mark encoding convention. This yields the original e-bit string. If e is larger than the size of the watermark, error correction was deployed to increase the encoding resilience. The watermark string can be then recovered by applying error correction decoding to this string, e.g., majority voting for each watermark bit. This process is illustrated in Figure 9.

```
det(attr, wm_key, db_primary_key, subset_sz, $v_{false}$, $v_{true}$, c, map[], subset_bnds[]) srt_attr ← sort on normalized crypto
  hash(wm_key,db_primary_key,wm_key) read_pipe ← null
  do { tuple ← next_tuple(srt_attr) }
  until (exists idx such that (subset_bnds[idx] == tuple))
  curr_subset ← idx
  while (not(srt_attr.empty())) do
    do {
      tuple ← next_tuple(srt_attr)
      read_pipe = read_pipe.append(tuple)
    } until (exists idx such that (subset_bnds[idx] == tuple))
    subset_bin ← (at most subset_sz elements of read_pipe, excluding last read)
    read_pipe.remove_all_remaining_elements_but_last_read()
    if (map[curr_subset]) then
      mark_data[curr_subset] ← decode (subset_bin, $v_{false}$, $v_{true}$, confidence)
      if (mark_data[curr_subset] != DECODING_ERROR)
          then map[curr_subset] ← true
    curr_subset ← idx
  return mark_data, map
```

Fig. 9. Watermark Detection (version using subset markers shown).

Categorical Types:

So far we have explored the issue of watermarking numeric relational content. Another important relational data type to be considered is categorical data. Categorical data is data drawn from a discrete distribution, often with a finite domain. By definition, it is either non-ordered (nominal) such as gender or city, or ordered (ordinal) such as high, medium, or low temperatures. There are a multitude of applications that would benefit from a method of rights protection for such data. In this section we propose and analyze watermarking relational data with categorical types.

Additional challenges in this domain derive from the fact that one cannot rely on arbitrary small (e.g., numeric) alterations to the data in the embedding process. Any alteration has the potential to be significant, e.g., changing DEPARTURE CITY from “Chicago” to “Bucharest” is likely to affect the data quality of the result more than a simple change in a numeric domain. There are no “epsilon” changes in this domain. This completely discrete characteristic of the data requires discovery of fundamentally new bandwidth channels and associated encoding algorithms.

Trustworthy Records Retention:

Trustworthy retention of electronic records has become a necessity to ensure compliance with laws and regulations in business and the public sector. Among other features, these directives foster accountability by requiring organizations to secure the entire life cycle of their records, so that records are created, kept accessible for an appropriate period of time, and deleted, without tampering or interference from organizational insiders or outsiders. In this chapter, we discuss existing techniques for trustworthy records retention and explore the open problems in the area.

Modern enterprises create, process, and store large quantities of records. The internal operations of an enterprise rely heavily on these records when making business decisions. Further, public confidence in an enterprise depends on its ability to maintain the confidentiality, integrity, and authenticity of its records throughout their life cycle. In response to a number of incidents of corporate fraud involving inappropriate modification and/or disclosure of financial and personal records, governments have issued laws and regulations that mandate organizations to provide trustworthy storage of their records for a guaranteed retention period, and to completely dispose of the records after their retention period has passed.

Unfortunately, most traditional security techniques are of little help in ensuring trustworthy retention of records, because traditional techniques focus on outsiders as the source of threats to the system. With organizational fraud, the threat comes from inside the organization, often from highly-placed employees who can coerce system administrators into aiding their coverup attempts. Trustworthy records retention requires new types of storage servers and database management systems, along with new techniques for indexing, record placement, data migration, and deletion.

Problem Definition:

Information subject to compliance regulations includes both structured records, such as relational database entries; and semi-structured or unstructured records, such as email, spreadsheets, reports, memos, and instant messages. We use the terms records and documents interchangeably when referring to semi-structured or unstructured collections of information.

Definition 1. *The goal of trustworthy record retention is to provide longterm retention and eventual disposal of organizational records in such a manner that no user can delete, hide, or tamper with any record during its retention period, nor recreate a record's content once it has been deleted.*

While each of the regulations mentioned above is designed for a particular application area and has its own unique features, a number of assurance criteria are common to many of the directives:

Guaranteed retention: Organizations must store records in a manner that prevents deletion of the records or tampering with their contents, even by insiders, for a regulation-mandated lifespan.

Long-term retention: The mandated retention periods are measured in years or even decades. For example, national intelligence information, educational records, and certain health records must be retained for over 20 years. Many mandated retention periods exceed the expected lifetime of today's storage devices.

Efficient access to data: Authorized requests for access to records must be serviced in a timely manner.

Data confidentiality: Only authorized parties can access confidential records.

Data integrity: Records can only enter the system through authorized means. Further, there must be procedures in place for correcting errors in the data, once detected.

Guaranteed deletion: Some laws require enterprises to properly dispose of records after a certain point in time (e.g., [24, 3]). In other situations, deletion may not be required by regulation, but still may be highly desirable from the organization's point of view, as the records may represent a liability. Once records are deleted, ideally it should be impossible to reconstruct any information about their contents, either directly or through metadata-based inference. We use the term trustworthy deletion to describe this combination of features.

Litigation holds: Electronic information may be used in litigation [18]. If a litigation hold is placed on a record, it must remain accessible until the hold is lifted, even if it reaches the end of its mandated lifespan.

Insider adversaries: Much recent high-profile corporate malfeasance has been at the behest of chief executive officers and chief financial officers who have the power to order the destruction or alteration of incriminating records. Thus many compliance regulations target powerful insiders as the primary adversaries. In effect, these adversaries have super user powers coupled with full access to the storage system hardware.

Auditing: The organization is subject to periodic audits of its records retention practices.

High penalties for non-compliance: Non-compliance with the regulations can bring stiff financial and criminal penalties [32]. For example, a chief financial officer can receive a prison sentence for publishing an incorrect financial report, even if the false information was included without his or her knowledge.

Usage Scenario and Threat Model:

A records retention system faces all the attacks that any computer system is vulnerable to (e.g., physical destruction, denial-of-service attacks), plus additional dangers that are unique to the compliance arena. In this section, we describe those latter dangers. For information on other kinds of threats and their countermeasures, we refer the reader to the other chapters in this volume and to any textbook on computer security.

The main focus in trustworthy records retention is on preventing malicious insiders from tampering with or destroying records. Further, the traditional notion of an insider attack is refined to assume a very powerful insider who is capable of gaining physical, root-level access to the storage media. While outside adversaries may also pose threats, measures that are effective against super user insiders will also stymie external attackers.

A second key factor in the threat model for trustworthy records retention is that the visible alteration or destruction of records is tantamount to an admission of guilt, in the context of litigation. Thus a successful adversary must perform their misdeeds undetectably.

The target usage scenario for trustworthy records retention is as follows. First, a legitimate user Alice creates and stores a record R that is subject to compliance regulations. Later, a user Mallory starts to regret R's existence and will do everything he can to prevent a subsequent user Bob from accessing R or inferring its existence. For example, Bob may be a regulatory authority looking for evidence of malfeasance, while Mallory may be the super user CEO or Alice herself. The primary goal of trustworthy records retention is to ensure that Bob can still find and read R until the end of R's mandated lifespan, no matter what Mallory does. For some applications, undetectable post hoc insertion of records is also considered a threat and must be addressed.

Once R reaches the end of its mandated lifespan and is deleted, then Mallory may wish to determine whether R ever existed or infer information about the contents of R, based on any traces of information about R that may remain in the system. A second goal of trustworthy records retention is to ensure that Mallory cannot make these inferences.

To illustrate some of the implications of the threat model, consider the following hypothetical scenarios:

Trustworthy retention: Mallory can employ his super user powers to attempt to modify or delete R, or to hide R by modifying indexes so that they no longer lead to R. Mallory can also swap out the disks in the storage server, replacing them with disks that do not contain any trace of R. We must make sure that Bob can detect Mallory's attacks and, where feasible, we must prevent them.
Trustworthy access and migration: Suppose that Alice's organization needs to migrate its compliance records to a new compliance storage server. Mallory is effectively in charge of the migration, and he wants to omit incriminating record R during the transfer. For trustworthy record retention, Bob must be able to detect whether any such modifications or omissions occurred during migration.
Trustworthy deletion: When its mandatory retention period is over and any litigation holds on R have been lifted, Alice's organization removes R. Mallory subsequently gains access to the storage server and looks for magnetic traces of R. He also looks in the current copies of indexes and other metadata and supporting data structures, to try to glean information about R. For many applications, trustworthy record retention needs to prevent Mallory from gaining any information about R.

Storage Architectures:

Conventional file/storage system access control mechanisms and data outsourcing techniques are intended to ensure that records and their metadata are only modified by legitimate applications. Under the outsourcing threat model, insiders are trusted but the storage server is not. The correctness of outsourced query answers can be guaranteed by the data owner by attaching appropriate signatures to the data that can be verified by the querier. These signature-based approaches only detect whether a record has been tampered with; they do not prevent tampering. The techniques for outsourcing and traditional access control are powerless against an adversary with superuser powers who can obtain any secret key and control the behavior of applications. Data owner Alice could alter the contents of her record R and re-sign it after it has already been committed to the storage server, or superuser Mallory could obtain access to Alice's private key and alter and re-sign R himself. In many applications, a key requirement for trustworthy retention of records is to prevent deletion and modification of the records. To thwart these attacks, we need a new kind of storage architecture.

- Based on the computer security principle of minimizing the trusted computing base, the component for enforcing the storage security properties should be as small as possible, both to reduce the probability that something could go wrong or be compromised, and to increase our ability to verify the correctness of the component. This means that we cannot rely on having a trusted database management system running on the storage server, or even a trusted indexing package.
- The cost of any effective attack against the component must be high, and its results must be conspicuous. For example, perhaps a simple auditing routine is guaranteed to be able to detect the aftereffects of the attack; or else many or all records insertions will fail after the attack. A number of design principles follow from this requirement; for example, the component should have a simple and well-defined interface, to robustly restrict traffic into the component to legitimate requests only. Further, the component must mediate all requests; in other words, the overwrite protection cannot be circumvented by, for example, directly accessing the rewritable disk.
- The resulting system must provide end-to-end security guarantees, not just guarantees for individual components.
- The price per byte of storage must be modest, as data volumes are very high. The conflict between security, cost-effectiveness, and efficiency makes the design of compliance storage extremely challenging.

Tape-based products: Due to the favorable cost-per-MB ratio of tapebased storage in the past, tape was a natural choice for massive data storage in commercial enterprise deployments where regulatory compliance is of concern. Thus storage vendors offered tape-based compliance storage first. The Quantum DLTSage predictive, preventative and diagnostic tools for tape storage environments [34] are a representative instance. The WORM assurances of the tape systems are provided under the assumption that only Quantum tape readers are deployed: “DLTSage WORM provides features to assure compliance, placing an electronic key on each cartridge to ensure WORM integrity. This unique identifier cannot be altered, providing a tamper-proof archive cartridge that meets stringent compliance requirements to ensure integrity protection and full accessibility with reliable duplication” [34]. Such systems, however, make impractical assumptions. Given the nature of magnetic tape, an attacker can easily dismantle the plastic tape enclosure and access the underlying data on a different customized reader, thus compromising its integrity. Relying on the physical integrity of a “plastic yellow label,” as in one product, to safeguard essential enterprise information is likely to be unacceptable in high-stakes commercial scenarios.

Optical-disk products: Optical disk media (CDs) have been around experimentally since 1969 and commercially available since 1983. Given the prohibitive costs of high-powered lasers in small form factors, in the early days, most CD devices were only capable of reading disk information. As the technology matured, write-once (and later read-write) media appeared. Optical WORM-disk solutions rely on irreversible physical write effects to ensure the inability to alter existing content. However, with ever increasing amounts of information being produced and requiring constant low-latency accessibility in commercial scenarios, it is challenging to deploy a scalable optical-only WORM solution. Moreover, optical WORM disks are plagued with other practical issues such as the inability to fine-tune WORM and secure deletion granularity (problems partially shared also by tape-based solutions). Moreover, due to bulk production requirements, optical disks are vulnerable to simple data replication attacks, with the end result that they do not provide any strong security features. Optical WORM disks also perform relatively poorly in price-performance measurements, because current technology is somewhat undersized for the volumes of data associated with compliance regulations. Sony’s Professional Disk for Data optical disk system, for example, holds only 23 GB per disk side [43]. Nevertheless, because it is faster than tape and cheaper than hard disks, optical WORM storage technology is often deployed as a secondary, high-latency storage medium to be used as second-tier storage in the framework of a hard disk-based solution. In such an environment, care needs to be taken in establishing points of trust and data integrity when data leaves the secured hard disk store for the optical media. As we will discuss below, such integrity assurances can be maintained with the help of additional secure hardware hosted inside the main store.

Hard disk products: Magnetic disk recording currently offers better overall cost and performance than optical or tape recording. Moreover, while immutability is often specified as a requirement for records, what is required in practice is that they be “term-immutable”, i.e., immutable for a specified retention period. Thus almost all recently-introduced WORM storage devices are built atop conventional rewritable magnetic disks, with write-once semantics enforced through software (“soft-WORM”).

Resistance to Physical Attack:

Our insider adversary Mallory has physical access to the storage media. To limit the damage that he can do, one potential approach is to house the storage in a tamperproof or tamper-evident box. However, such a box would trap heat, making it necessary to run the storage server at lower speeds and reducing its cost-effectiveness. Thus this solution is unlikely to be popular with customers or vendors. Further, disks do fail and require replacement, which is hard to reconcile with the notion of tamper-evidence.

Unfortunately, this method does not provide strong WORM guarantees. Using off-the-shelf resources, an insider can open the storage medium enclosures to gain physical access to the underlying data and to any flash-based checksum storage. She can then surreptitiously replace a device by copying an illicitly modified version of the stored data onto a identical replacement unit. Maintaining integrity-authenticating checksums at device or software level does not prevent this attack, due to the lack of tamper-resistant storage for keying material. The adversary can access integrity checksum keys and construct a new matching checksum for the modified data on the replacement device, thus remaining undetected. This attack will still be effective if we add tamper-resistant storage for keying material [11], because a superuser is likely to have access to keys while they are in active use: achieving reasonable data throughputs will require integrity keys to be available in main memory for the main (untrusted) run-time data processing components.

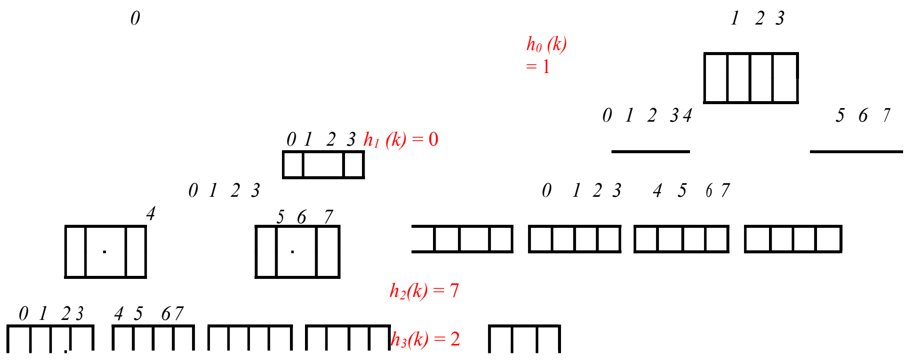
A potentially more effective approach is to leverage components that are both tamper-resistant and active, such as general-purpose secure coprocessors (SCPUs). By adding a trusted SCPU inside the storage server, we can guarantee the trustworthiness of records from that server, even if the records subsequently pass through untrusted and possibly hostile environments inside or outside the server. The SCPU can run certified code; its close proximity to the data storage, coupled with its tamper-resistance guarantees, offers the possibility of higher security assurances at minimal extra cost.

Trustworthy Indexing:

Indexing ensures that a target record can be quickly extracted from terabytes of data. In our discussion, we will assume that each record is assigned an integer identifier as it arrives at the storage server, and that identifiers are given out in increasing order. Any indexing approach for trustworthy records retention must have the following properties:

- The index itself must be trustworthy. As explained below, in practice this means that the search path to an index entry must be immutable for the lifetime of the record that it indexes.
- To keep the trusted computing base small, the indexing code should reside outside the storage server.
- To ensure that a record R is entered and retained in the appropriate indexes before Mallory regrets its existence, the insertion and indexing of R must be performed atomically.
- All traces of R must be removed from the index when R is deleted.

Trustworthy Records Retention 369



(a) GHT Before Insertion (b) GHT After Insertion

The first step in ensuring trustworthy indexing is to store the index on WORM. The problem of creating an index on write-once media has been studied before. For example, in write-once B-trees [6] (Figure 1(a)), a node is split into two new nodes when it overflows. Two pointers are added to the end of its parent node, superseding the earlier pointer to the old node. If the parent node overflows, it is split as well and only the most recent pointers are copied. When the root node splits, a new root is created. Unfortunately, the use of WORM alone is insufficient to make this or any other index trustworthy. For example, in a write-once B-tree, an adversary can effectively modify any record he wishes by creating a new version of the appropriate nodes, as shown in Figure 1(b). Although every alteration of the nodes is preserved in WORM storage, it is too expensive to examine each version of the tree on each lookup to guard against tampering.

Traditional hash-based structures are also not secure against tampering. For example, in dynamic hashing and its extensions (e.g., [5, 8, 2]), when the number of records in a hash table exceeds a high water mark, a new hash table with a larger size is allocated and all the records are rehashed and moved into the new table. In this way, dynamic data growth can be supported with good performance. The ability to relocate records, however, provides an opportunity for an adversary to alter the records during the copying step. A trie that stores records in its leaf nodes and moves records to new leaf nodes as the trie grows will also be untrustworthy. All these approaches are vulnerable because the search path to a particular record is not term-immutable. To date, researchers have proposed trustworthy versions of hashing and inverted indexes, both of which guarantee term-immutable search paths. For example, a generalized hash tree (GHT) supports exact-match lookups of records based on attribute values [50]. One can use such an index, for example, to find all email sent from a particular address.

A GHT is a balanced tree-based data structure that does not require periodic rebalancing. In a GHT, predefined hashes of the record key determine all its possible lookup or insertion locations. The locations where a record can be inserted or looked up are therefore immutable. To insert or look up a record in a GHT, the record key is hashed to obtain a position within the root node (see Figure 2). If the corresponding node position at the root node is empty, the record is inserted there. If there is a collision, the key is rehashed (using a different hash function) and an attempt is made to insert the key in the appropriate subtree of the root node. This process is repeated until an empty node position is found. If a record cannot be inserted in any existing node of the tree, a new leaf node is added.

Full-text search (keyword search) is the most convenient way to query unstructured records such as email bodies and reports. Search engines typically use inverted indexes for this purpose [9]. As shown in Figure 3(a), an inverted index comprises a dictionary of terms (i.e., words that appear in documents) plus a posting list for each term, containing the identifiers of all records containing that term (with additional metadata such as term frequency within the record, term type, and term position within the record). Queries are answered by scanning the posting lists of the terms in the query. The records referenced in the posting lists are assigned scores for the query, based on similarity measures (e.g., cosine, Okapi, pivoted, Dirichlet [47, 36]). The scores are used to rank the records, producing an ordered list of results. Multi-keyword conjunctive queries can be answered by intersecting the posting lists of the query terms. To make the intersection fast, an additional index such as a B+ tree is usually kept for each posting list, and a zigzag join is used to perform the intersection

Trustworthy Migration:

Storage systems technology evolves rapidly. It is impractical to store records on a single server for decades, as the server will become obsolete and too expensive to maintain. Organizations themselves may also evolve, via mergers, spin-offs, and reorganizations that require records to be copied or moved. When records must be moved, the migration process needs to be trustworthy; that is, it should be possible to verify that the migration was completed appropriately, even if a superuser adversary performed the migration.

To date, researchers have developed two schemes for trustworthy migration of records between compliance storage servers; both of these schemes rely on secure coprocessors (SCPUs). In the scheme proposed in [41], the SCPU of the original storage server (SCPU1) should be provided assurances that the migration target environment (SCPU2) is trustworthy and endorsed by the relevant regulatory authority (RA).

To achieve this, the migration process is initiated by (i) the system operator retrieving a migration certificate (MC) from the RA. The MC is in effect a signature on a message containing the time stamped identities of SCPU1 and SCPU2. Upon migration, (ii) the MC is presented to SCPU1 (and possibly SCPU2), who authenticates the signature of the RA. If this succeeds, SCPU1 is ready to (iii) mutually authenticate and perform a key exchange with SCPU2, using their internally stored key pairs and certificates. SCPU2 will need backwards-compatible authentication capabilities, as the default authentication mechanisms of SCPU2 may be unknown to SCPU1. This backwards compatibility is relatively easy to achieve as long as the participating certificate authorities (i.e., SCPU manufacturer or delegates thereof) still exist and have not been compromised yet.

A cross-certification chain can be set up between the old and the new certification authority root certificates. Once (iii) succeeds, SCPU1 will be ready and willing to transfer a description of the state of the compliance records and index contents on a secure channel provided by an agreed-upon symmetric key (e.g., using a Diffie-Hellman variant). After the state information has been migrated, the actual records and index contents can be transferred by the main CPUs , without SCPU involvement.

The scheme proposed in [22] supports the migration of files through multiple servers while maintaining integrity guarantees. In this scheme, file and directories can be rearranged or omitted during the migration, based on corporate policies. The approach relies on the existence of a trusted third party, such as a storage system vendor, who records the public keys associated with the sequence of storage servers purchased by an organization.

The migration process is divided into three phases:

- In phase 1, the party in charge of migration prepares a plan for the migrations. The log of this plan includes the policies governing the migration and, in compact form, a representation of the list of files and directories to be migrated, the planned file and directory omissions, and the planned directory restructurings
- In phase 2, the current storage server generates certificates that attest to the current state of the directory tree and file contents, and adds them to the log. The scheme assumes that the server will generate the certificates correctly, either because it is part of the trusted computing base or because it contains trusted hardware that is capable of perusing directories and creating certificates. In either case, these certificates can be generated reasonably quickly.
- Finally, in phase 3, the party in charge of migration moves the files to be migrated, and also copies the signed log to the new server.

After migration, anyone can look up the public keys used by an organization's series of storage servers, and then use validation routines to check whether the migration took place appropriately. For example, a trusted thirdparty auditor can certify the migration immediately after its completion, at approximately the same rate of speed as it took to generate the certificates in phase 2. At any point after the migration, a user can also quickly check whether a particular file was migrated appropriately.

Migration policies can be very complex. For example consider the policy Delete all files containing the word Martha. This deletion should preserve confidentiality: a person reading files and logs on the destination server should not learn anything other than the fact that the deleted file contained Martha. One can handle this problem in an elegant manner if the storage server contains a small amount of trusted hardware that can run downloaded query code and sign the results, to testify that only a certain set of files contained the word Martha [22]. Then this certificate can be included in the log file and migrated to the new server along with the appropriate subset of files. Anyone can verify that exactly the set of files listed in the query certificate was omitted during the migration.

Trustworthy Deletion:

Since the primary purpose of WORM devices is to prevent data deletion, it is not surprising that cost-effective and trustworthy deletion of records is difficult. WORM devices use physical security measures, such as repeatedly overwriting the data blocks with certain patterns, to erase records from the media. However, simple erasure is not enough for trustworthy deletion, as an erased record can be recreated by reverse-engineering an index. Overall, no index entry deletion scheme developed so far meets all the requirements for trustworthy deletion.

For the deletion of document d to be strongly secure, the presence or absence of any word w in any reconstruction of d should not convey any formation about its presence in the original document. More formally, let S be a reconstruction of the set of words in d, generated by the adversary by scanning directories, indexes, and migration logs. We say that d's deletion is strongly secure iff

$$\forall w.P(w \in d|w \in S) = P(w \in d)\ \forall w.P(w \in d|w /\in S) = P(w \in d),$$

where $P(w \in d)$ denotes the probability of the word w belonging to document d, while $P(w \in d|w \in S)$ denotes the probability of w being in D given that w is in S.

Damage Quarantine and Recovery in Data Processing Systems:

In this article, we address transparent Damage Quarantine and Recovery (DQR), a very important problem faced today by a large number of mission, life, and/or business-critical applications and information systems that must manage risk, business continuity, and assurance in the presence of severe cyber attacks. Today, these critical applications still have a "good" chance to suffer from a big "hit" from attacks. Due to data sharing, interdependencies, and interoperability, the hit could greatly "amplify" its damage by causing catastrophic cascading effects, which may "force" an application to halt for hours or even days before the application is recovered. In this paper, we first do a thorough discussion on the limitations of traditional fault tolerance and failure recovery techniques in solving the DQR problem. Then we present a systematic review on how the DQR problem is being solved. Finally, we point out some remaining research issues in fully solving the DQR problem.

There are at least two main reasons on why mission/life/business-critical applications have an urgent need for transparent damage quarantine and recovery. Firstly, despite that significant progress has been made in protecting applications and systems, mission/life/business-critical applications still have a "good" chance to suffer from a big "hit" from attacks. Due to data sharing, interdependencies, and interoperability between business processes and applications, the hit could greatly "amplify" its damage by causing catastrophic cascading effects, which may "force" an application to shut down itself for hours or even days before the application is recovered from the hit. (Note that high speed Internet, e-commerce, and global economy have greatly increased the speed and scale of damage spreading.) The cascading damage and loss of business continuity (i.e., DoS) may yield too much risk. Because not all intrusions can be prevented, DQR is an indispensable part of the corresponding security solution, and a quality DQR scheme may generate significant impact on risk management, business continuity, and assurance.

Overview of the DQR Problem:

Although the DQR problem may be addressed at several abstraction levels (e.g., disk level, OS level, DBMS level, transaction level, application level), solving the DQR problem at the transaction level is particularly appealing due to the following reasons. The transaction abstraction has revolutionized the way reliability, including recoverability, is engineered for applications. Through a simple API interface provided by an easy-to-use transaction (processing) package which is today an integral part of mainstream application development environments such as J2EE and .NET, programmers can make applications transactional in a rather automatic, effort-free fashion. And the benefits of making applications transactional are significant: "failure atomicity simplifies the maintenance of invariants on data" [2]; a guaranteed level of data consistency can be achieved without worrying about say race conditions; durability makes it much easier for programmers to get the luxury of recoverability.

Scope of Transaction Level DQR

In the rest of this paper, we will focus on transaction level DQR problems, models, and solutions. In particular, the transaction-level scope of an application and its environment are shown in Figure 1. In an information system, the transaction processing components of an application do not form an "isolated" system. Instead, these components will interact with their environment, which includes the Physical World, the various non-transactional actions, and the various types of data sources. Through these interactions, inputs are taken, physical world effects can be caused, and non-transactional attacking actions can "poison" the application's transaction scope. Although we are aware that the cyberspace damage and cascading effects can certainly cause damage in the physical world, this paper will focus on the cyberspace DQR solutions which will help minimize the damage caused in the physical world.

The Threat Model and Intrusion Detection Assumption

Working at the transaction level does not mean that malicious transactions are the only threat we can handle. Instead, as shown in Figure 1, we allow threats to come from either inside or outside of the transaction-level scope of applications. Nevertheless, to exploit the application's transaction mechanism to achieve a malicious goal, both inside and outside threats need to either directly corrupt certain data objects or get certain malicious transactions launched. Outside non-transactional attack actions (e.g., Witty worm) may bypass the transaction interface and corrupt some data objects via low-level (e.g., file or disk) operations. In addition, non-transactional buffer overflow attacks may break in certain running program of the application; then the attacker can manipulate the program to launch certain malicious transactions.

The intrusion detection assumption: We assume that a set of external intrusion detection sensors will do their job and tell us which operations (or transactions) were malicious or which data objects were originally corrupted by the attack. These sensors may be a network-level (e.g., [12]), host-level (e.g., [13]), database-level (e.g., [14]) or transaction-level (e.g., [15, 16]) intrusion detection sensor. These sensors may enforce misuse detection (e.g., [17]), anomaly detection (e.g., [18, 19]), or specification-based (e.g., [20, 21]) detection mechanisms. We assume these sensors are usually associated with false positives, false negatives, and detection latency. Finally, sensors that detect data corruption (e.g., [22, 23, 24]) may also be used.

The DQR Problem/Solution Space:

In our view, the DQR problem is a 6-dimensional problem:

(1) The damage propagation dimension explains why cascading effects can be caused and why quarantine is needed. Although some specific types of damage (e.g., when an individual credit card account is corrupted) could be self-contained, a variety types of damage are actually very infectious due to data sharing, interdependencies, and interoperability between business processes and applications. For example, in a travel assistant Web Service, if a set of air tickets are reserved due to malicious transactions, some other travelers may have to change their travel plans in terms of which airlines to go, which nights to stay in hotel, etc.. Furthermore, the changed travel plans can cause cascading effects to yet another group of travelers; and the propagation may go on and on.

(2) The recovery dimension covers three semantics for recovery: the coldstart semantics mean that the system is "halted" while damage is being assessed and repaired. (Damage assessment is to identify the set of corrupted data objects. Damage repairing is to restore the value of each corrupted data object to the latest before-infection version.) To address the DoS threat, recovery mechanisms with warmstart or hotstart semantics are needed. Warmstart semantics allow continuous, but degraded, running of the application while damage is being recovered. Hotstart semantics make recovery transparent to the users.

(3) The quarantine dimension covers a spectrum of quarantine strategies: (a) coldstart recovery without quarantine, (b) warmstart recovery with conservative, reactive quarantine, (c) warmstart recovery with proactive or predictive quarantine, (b) hotstart recovery with optimistic quarantine, to name a few.

(4) The application dimension covers the various transaction models deployed by conventional and modern applications. The uniqueness of each model may introduce new challenges for solving the DQR problem.

(5) The correctness dimension tells whether a DQR scheme is correct in terms of consistency, recoverability, and quarantinability.

(6) The quality dimension allows people to measure and compare the quality levels achieved by a set of correct yet different DQR schemes.

What Transaction Level DQR Solutions Cannot Do:

First, although transaction-level DQR solutions will help minimize the damage caused by cyberspace attacks in the physical world, they cannot repair physical damage, which is a different field of study. Second, transaction-level DQR solutions are not designed to patch software which is another critical intrusion recovery problem. Nevertheless, transaction-level DQR solutions and software patching are complementary to each other. Transaction-level DQR solutions can help quarantine and repair the damage done by unpatched software broken-in by the adversary.

Traditional Failure Recovery Techniques and Their Limitations:

DQR theories and mechanisms draw on work from several areas of systems research such as survivable computing, fault-tolerant computing, and transaction processing. Among all the relevant areas, the closest one should be Failure Recovery, which is part of Fault Tolerance [25]. In the literature, failure recovery has not only been extensively studied in data processing systems [3, 26, 4], but also been thoroughly studied in other types of computing systems. In [27] and [28], operating systems failure recovery is investigated.

Transactional Undo/Redo:

The crux of transactional undo/redo techniques is correcting the application states that are corrupted due to failures. For data-processing systems or dataoriented applications in which doing read and write operations on various data objects (managed by a set of databases) represents the main activities, failure recovery is rooted in the transaction concept which has been around for a long time. This concept encapsulates the ACID (Atomicity, Consistency, Isolation, and Durability) properties. Data-oriented applications are not limited to the database area. The basic recovery procedure is almost the same for all applications: when a failure happens, a set of undo operations will be performed to rollback the application's state to the most recent checkpoint, which is maintained through logging, then a set of redo operations will be performed to restore the state to exactly the failing point. Nevertheless, the concrete recovery algorithms depend heavily upon how changes are logged. WAL (Write Ahead Logging) is today the standard approach widely accepted by the database industry. Some of the commercial systems and prototypes based on WAL are ARIES.

Limitations in Solving the DQR Problem: Although existing transaction recovery methods are matured in handling failures, they are not designed to deal with malicious attacks. In particular, first, the durability property ensures that traditional recovery mechanisms never undo committed transactions. However, the fact that a transaction commits does not guarantee that its effects are desirable. Specifically, a committed transaction may reflect inappropriate and/or malicious activity.

Second, although attack recovery is related to the notion of cascading abort [3], cascading aborts only capture the read-from relation between active transactions, and in standard recovery approaches cascading aborts are avoided by requiring transactions to read only committed data.

Third, there are two common approaches to handling the problem of undoing committed transactions: rollback and compensation. (3a) The rollback approach is simply to roll back all activity – desirable as well as undesirable – to a checkpoint believed to be free of damage. The rollback approach is effective, but expensive, in that all of the desirable work between the time of the checkpoint and the time of recovery is lost. Although there are algorithms for efficiently establishing snapshots on-the-fly [38, 39, 40], maintaining frequent checkpoints may not work since no checkpoint taken between the time of attack and the time of recovery can be used. (3b). The compensation approach seeks to undo either committed transactions or committed steps in long-duration or nested transactions [54] without necessarily restoring the data state to appear as if the malicious transactions or steps had never been executed. There are two kinds of compensation: action-oriented and effectoriented. Action-oriented compensation for a transaction or step Ti compensates only the actions of Ti . Effect-oriented compensation for a transaction or step Ti compensates not only the actions of Ti , but also the actions that are affected by Ti . Although various types of compensation are possible, all of them require semantic knowledge of the application, and none of them is adopted by mainstream commercial systems.

Fourth, classic redo operations cannot repair damage because they do not re execute affected transactions.

Replication-based Recovery:

The crux of the replication based recovery is using redundancy to mask/tolerate failures. Replication-based recovery does not undo erroneous operations. In data-oriented applications, the replication idea is embodied through the widely adopted practice of data replication [60, 3] and standby databases [53]. In such replicated systems, each request (or transaction) will be processed by all the replicas in which each data object is replicated. When a failure happens to the primary database, the responses (or outputs) generated by a standby (or replicated) database can be returned to the client as if the failure had never happened. (In distributed computing, the replication idea is embodied through such techniques as RAPS (reliable array-structured partitioned service), the state-machine approach [61], and virtual synchrony.
Limitations in Solving the DQR Problem: Both data replication and standy databases will not only replicate good work, but also replicate infection!

Storage Media Backup-Restore:

The idea of storage media backup-restore is proven very practical and valuable. It is fully embraced by the IT industry: Computer Associates large enterprise backup solutions [63], Symantec LiveState recovery products [64], the Sonasoft Solution [65], just to name a few. This idea is complementary to the recovery idea and the replication idea, but in many cases it cannot achieve fine-grained data consistency, while the two other ideas can.
Limitations in Solving the DQR Problem: Among the data objects included in a backup, storage media backup-restore techniques cannot distinguish clean data objects from dirty, corrupted ones.

Solving the DQR Problem:

In this section, we present a systematic review on how the DQR problem is being solved in the literature. Although self repairable file systems are proposed, most DQR mechanisms proposed in the literature are transactionlevel solutions. So here we concentrate on transaction-level.

The Model: In our model, a transaction is a set of read and write operations that either commits or aborts. For clarity, we assume there are no blind writes, although the theory can certainly be extended to handle blind writes. At the transaction level, an application (e.g., the application types shown in Figure 2) is a transaction execution history. Since recovery of uncommitted transactions is addressed by standard mechanisms [3], we can safely ignore aborted transactions and only consider the committed projection C(H) of every history H.

In principle, the correctness of a DQR scheme (or solution) can be "checked" either by the operations performed by the scheme or by the resulted effects. Here, we use the resulted history of a DQR scheme to study its correctness. In our model, the DQR histories resulted from a DQR scheme may contain the following information:

- A DQR history may contain two types of malicious transactions, four types of legitimate transactions, and one type of cleaning transactions: Type 1 malicious transactions are issued by attackers or malicious code; more broadly, transactions executed by mistake can be viewed as a Type 2 malicious transaction A legitimate transaction may be either a regular transaction or a reexecuted transaction; and both regular and reexecuted transactions may be affected or damaged if they read any corrupted data object. Finally, cleaning transactions only contain backward or forward overwrite operations, depending upon how the recovery is performed.
- A classic history consists of only operations, while a DQR history is an interleaved sequence of operations and data store states. The data store contains all the data objects that a transaction may access. The state of the data store at time t is determined by the latest committed values of every data object in the store.
- A data store state (e.g., a database state) contains three types of corrupted data objects and two types of clean data objects. Type 1 corrupted data objects are originally generated by the writes of malicious transactions. Type 2 are originally generated by affected transactions. Type 3 are originally generated by non-transactional attacking actions outside of the application's transaction scope. Note that a corrupted data object may be read or updated several times before it is repaired (a.k.a. cleaned). Type 1 clean data objects are never corrupted. Type 2 clean data objects are once corrupted, but they are repaired.

Damage Propagation:

Based on the threat model, we know where malicious transactions come from. To see how affected transactions are generated and how the damage spreads, we should do dependency (or causality) analysis.

Definition 4.1 (dependency graph) As stated in [68], transaction Tj is dependent upon Ti in a history if there exists a data object o such that Tjreads o after Ti has updated o; Ti does not abort before Tj reads o; and every transaction (if any) that updates o between the time Ti updates o and Tj reads o is aborted before Tj reads x. In a history, T1 affects T2 if the ordered pair (T1, T2) is in the transitive closure of the dependent upon relation. Finally, we define the dependency graph for a (any) set of transactions S in a history as DG(S)=(V,E) in which V is the union of S and the set of transactions that are affected by S. There is an edge, Ti → Tj , in E if Ti ∈ V , Tj ∈ (V − S), and Tj is dependent upon Ti . ✷ Example Consider the following history over (B1, B2, G1, G2, G3, G4):

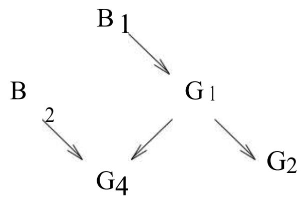


Fig. 3. Dependency Graph for History H1

Example Consider the following history over (B1, B2, G1, G2, G3, G4):

H1 : rB1 [x]wB1 [x]cB1 rG1 [x]wG1 [x]rG3 [z]wG3 [z]cG3 rG1 [y]wG1 [y]cG1 rG2 [y]wG2 [y]rB2 [z]wB2 [z]cB2 rG2 [v]wG2 [v]cG2 rG4 [z]wG4 [z]rG4 [y]wG4 [y]cG4

In H1, B1 and B2 are malicious transactions while the other three are legitimate transactions; rT [x] (wT [x]) is a read (write) operation by transaction T on data object x; cT is the commit operation of T. Let B = {B1, B2}, DG (B) is shown in Figure 3.

The Spectrum of DQR Scheme:

The concept of DQR histories allows us to see the differences between the ones on the "spectrum" of DQR schemes. (a) On one end of the spectrum, a static DQR scheme will stop processing new transactions until every corrupted data object is repaired or cleaned. (A corrupted data object is repaired if its value is restored to the latest clean version before corruption.) So since the time of detection, which is also the time when the recovery starts, the corresponding DQR history will proceed with only cleaning transactions until the repair is completed. In addition, affected transactions should be reexecuted; otherwise, DoS is caused. (b) On the other end, an optimistic, dynamic DQR scheme may do dependency analysis (a.k.a. damage assessment), execute cleaning transactions, execute to-be-reexecuted transactions, and execute new transactions concurrently. (c) Semi-dynamic DQR schemes may certainly stay on the spectrum between the two ends. For example, in [69, 70], there is a dedicated scan phase during which dependency analysis is performed, but no new transactions can be executed.

Static DQR Solutions:

Static DQR solutions "halt" the database (service) before the repair is completed. Since no new transactions can be executed during static DQR, the damage will not spread unless there are incorrect repair operations. Hence, damage quarantine is not an issue in static DQR. As a result, static DQR has two aspects: damage assessment, which identifies every corrupted data object, and damage repair, which restores the value of each corrupted data object to its pre-corruption version.

In terms of how damage assessment and repair can be done, existing static DQR methods are either data-oriented [71] or transaction-oriented [68]. Transaction-oriented methods assess and repair the damage by identifying and backing out affected transactions. In particular, they work as follows.

- **Damage Assessment** Build the dependency graph defined in Definition 4.1 for the set of malicious transactions detected. Based on Lemma 4.1, the dependency graph consists of all and only the affected transactions that have "contributed" to damage propagation. Assuming that read operations are logged together with write operations, it is not difficult to build the dependency graph. It is shown in [68] that the log can be scanned forward only once (i.e., one-pass) from the entry where the first malicious transaction starts to locate every affected transaction.
- **Repair** When the damage assessment part is done, scan backward from the end of the log to semantically revoke (or undo) the effects of all the malicious transactions and the transactions included in the dependency graph. Note that here the undoes should be performed in the reverse commit order.

In contrast, data-oriented methods use the read and write operations of transactions to trace the damage spreading from one data object to another, and compose a specific piece of code to repair each damaged data object. In particular, data-oriented methods work as follows.

- Damage Assessment Construct a specific damage propagation graph in which each node is a (corrupted) data object while each directed edge from node x to y is a transaction T such that T reads x and writes y. The damage propagation graph can be built by one-pass scanning of the log.
- Repair Once the damage propagation graph is constructed, for each data object x contained in the graph, search through the log to find the latest pre-corruption version of x. Then repair x by overwriting the value of x with the searched version.

Static Repair via History Rewriting:

From the correctness point of view, both data-oriented methods and transactionoriented methods would result in a history that is conflict equivalent to the serial history composed of only the legitimate, unaffected transactions. (C(H1) is conflict equivalent to C(H2) if they contain the same set of operations and they order every pair of conflicting operations in the same way.) Nevertheless, the history rewriting framework proposed in [74] shows that if we relax the correctness requirement from conflict equivalence to view equivalence, we may even save the work of affected transactions. In particular, by exploiting two new semantic relationships between transactions, denoted can-follow and can-precede, respectively, the history rewriting framework can rewrite every "infected" history, which always starts with a malicious transaction, to a ready-to-repair history in which every legitimate, unaffected transaction precedes all the malicious transactions. Such a rewritten history typically looks like the following. Here, Gi is a legitimate, unaffected transaction and AGi is an affected transaction. In addition, Fi is called a fix. A fix for a transaction like B1 is a set of variables read by the transaction given values as in the original position of the transaction before the history is rewritten. Gi1...AGj1...Gin...AGjm B F1 1 AGFk1 k1 ...BFl1 ...AGFk.

Dynamic DQR Solutions:

In static DQR, new transactions are blocked during the repair process. This prevents static DQR mechanisms from being deployed by 24*7 database applications. As 24*7 database applications are becoming more and more common, dynamic DQR solutions that can do non-stop, zero down-time attack recovery are in demand.

Dynamic DQR Solutions with Reactive Quarantine:

To have zero down-time, neither damage assessment nor repair can block the execution of new transactions. This means that dependency analysis, execution of new transactions, execution of cleaning transactions, and reexecution of affected transactions need to be done in parallel. To meet this challenge, people may wonder if the traditional transaction management architecture needs to be rebuilt. Fortunately, Figure 4 shows that the traditional transaction management architecture [3] is adequate to accommodate on-the-fly repair. The Repair Manager is applied to the growing logs of on-the-fly histories to mark any bad as well as affected transactions. For every bad or affected transaction, the Repair Manager builds a cleaning transaction and submits it to the Scheduler. The cleaning transaction is only composed of write operations. The Scheduler schedules the operations submitted either by user transactions or by cleaning transactions to generate a correct on-the-fly history. Affected transactions that are semantically revoked (or undone) can be resubmitted to the Scheduler either by users or by the Repair Manager. Finally, the Recovery Manager executes the operations submitted by the Scheduler and logs them.

On-the-fly attack recovery faces several unique challenges. First, since new transactions may first read corrupted data objects then update clean data objects, the damage may continuously spread, and the attack recovery process may never terminate. Accordingly, we face two critical questions. (a) Will the attack recovery process terminate? (b) If the attack recovery process terminates, can we detect the termination? Second, we need to do repair forwardly since the assessment process may never stop. The assessment process may never stop since the damage may continuously spread. Third, cleaned data objects could be re-damaged during attack recovery.

Dynamic DQR Solutions with Proactive Quarantine:

From the viewpoint of on-the-fly non-stop recovery, fault/damage quarantine can be viewed as part of recovery. The goal of damage quarantine is to prevent the damage from spreading out during recovery. One problem of the solution shown in Figure 4 is that its damage quarantine may not be effective, since it contains the damage by disallowing transactions to read the set of data objects that are identified (by the Damage Assessor) as corrupted. This reactive one-phase damage containment approach has a serious drawback, that is, it cannot prevent the damage caused on the objects that are corrupted but not yet located from spreading. Assessing the damage caused by a malicious transaction B can take a substantial amount of time, especially when there are a lot of transactions executed during the detection latency of B. During the assessment latency, the damage caused during the detection latency can spread to many other objects before being contained.

It is clear that the containing phase overcontains the damage in most cases. Many objects updated within the containing time window can be undamaged. And we must uncontain them as soon as possible to reduce the corresponding availability loss. Accurate uncontainment can be done based on the reports from the Damage Assessor, which could be too slow due to the assessment latency. [75] shows that transaction types can be exploited to do much quicker uncontainment. In particular, assuming that (a) each transaction Ti belongs to a transaction type type(Ti) and (b) the profile for type(Ti) is known, the read set template and write set template can be extracted from type(Ti)'s profile. The templates specify the kind of objects that transactions of type(Ti) can read or write. As a result, the approximate read-from dependency among a history of transactions can be quickly captured by identifying the readfrom dependency among the types of these transactions. Moreover, the typebased approach can be made more accurate by materializing the templates of transactions using their inputs before analyzing the read-from dependency among the types.

Other damage quarantine methods:

(a) In [76], a color scheme for marking and containing damage is used to develop a mechanism by which databases under attack could still be safely used. This scheme assumes that each data record has an (accurate) initial damage mark or color (note that such marks may be generated by the damage assessment process), then specific color-based access controls are enforced to make sure that the damage will not spread from corrupted data objects to clean ones.

(b) Attack Isolation The idea is to isolate likely suspicious transactions before a definite determination of intrusion is reported. In particular, when a suspicious session B is discovered, isolating B and the associated transactions transparently into a separate environment that still appears to B to be the actual system allows B's activities to be kept under surveillance without risking further harm to the system. An isolation strategy that has been used in such instances is known as fishbowling. Fishbowling involves setting up a separate look-alike host or file system and transparently redirecting the suspicious entity's requests to it. This approach allows the incident to be further studied to determine the real source, nature, and goal of the activity, but it has some limitations, particularly when considered at the application level. First, the substitute host or file system is essentially sacrificed during the suspected attack to monitor B, consuming significant resources that may be scarce. Second, since B is cut off from the real system, if B proves innocent.

Quality Evaluation:

Correctness does not always imply high quality. Two correct DQR schemes may yield very different quality levels in the DQR services they provide. In failure recovery, the MTTF-MTTR model (Mean Time To Failure - Mean Time To Recovery model) provides a neat yet precise way to gain concrete understanding of the quality of a recovery service which is measured by MTTF/(MTTF+MTTR), and this quality model has played a crucial role in advancing the theories and technologies of failure recovery. Unfortunately, due to the reasons mentioned in Section 1, the MTTF-MTTR model is no longer sufficient for defining the quality of DQR services.

In principle, the quality of DQR services can be evaluated by a vector composed of three criteria regarding data integrity and two criteria regarding availability:

- **C1:** Dirtiness depends on the percentage of corrupted data objects in each data store state.
- **C2:** Data Freshness When a clean yet older version of a corrupted data object o is made accessible during recovery; freshness depends on whether a fresher version of o is used by new transactions. Note that one clean version can be much fresher than another clean version of the same data object.
- **C3:** Data Consistency Violation of serializability can compromise data consistency no matter the history is multi-versioned or not.
- **C4:** Rewarding Availability The more clean or cleaned data objects are made accessible to new transactions, the more rewarding availability (or business continuity) is achieved. The more rewarding availability, the less denial-of-service will be caused.
- **C5:** Hurting Availability The more corrupted data objects are made accessible to new transactions, the more hurting availability is yielded. Because hurting availability will hurt data integrity and spread the damage, hurting availability is worse than letting the corrupted objects be quarantined.

An important finding gained in reliability evaluation research (e.g., [34, 79]) is that state transition models may play a big role in quality evaluation. A state transition model specific for DQR systems can be the model shown in Figure 6, where in terms of any portion of the application (e.g., a set of data objects), the system has 6 basic states: they are self explanatory except that the 'M' state means that the portion is Marked as damaged. Ignoring the 'Q' state, we could measure Dirtiness by (MTTC+MTTM+MTTR)/(MTTC+ MTTD+MTTM+MTTR); and Rewarding Availability by (MTTC+MTTR)/ (MTTC+MTTD+MTTM+MTTR). In [80], this idea is well justified in the context of intrusion tolerant database systems through Continuous Time Markov Chain based state transition model analysis and prototype experiments based validation.