Cooper, J.A., Goldreich, O. “Computer Security and Cryptography”
The Electrical Engineering Handbook
Ed. Richard C. Dorf
Boca Raton: CRC Press LLC, 2000

97 Computer Security and Cryptography

J. Arlin Cooper
Sandia National Laboratories

Oded Goldreich
Weizmann Institute of Science

97.1 Computer and Communications Security

J. Arlin Cooper

Computer security is protection of computing assets and computer network communication assets against abuse, unauthorized use, unavailability through intentional or unintentional actions, and protection against undesired information disclosure, alteration, or misinformation. In today's environment, the subject encompasses computers ranging from supercomputers to microprocessor-based controllers and microcomputers, software, peripheral equipment (including terminals, printers), communication media (e.g., cables, antennas, satellites), people who use computers or control computer operations, and networks (some of global extent) that interconnect computers, terminals, and other peripherals.

Widespread publicity about computer crimes (losses estimated at between $300 million and $500 billion per year), **hacker** (cracker) penetrations, and **viruses** has given computer security a high profile in the public eye [Hafner and Markoff, 1991]. The same sorts of technologies that have made computers and computer network communications essential tools for information and control in almost all businesses and organizations have provided new opportunities for adversaries and for accidents or natural occurrences to interfere with crucial functions. Some of the important aspects are industrial/national espionage, loss of functional integrity (e.g., in air traffic control, monetary transfer, and national defense systems), and violation of society's desires (e.g., compromise of privacy). The emergence of the World Wide Web access to the **Internet** has been accompanied by recent focus on financial transaction vulnerabilities, crypto system weaknesses, and privacy issues.

Fortunately, technological developments also make a variety of controls (proactive and follow-up) available for computer security. These include personal transaction devices (e.g., **smart cards**, and **tokens**), **biometric verifiers**, **port protection devices**, encryption, authentication, and digital signature techniques using symmetrical (single-key) or asymmetrical (**public-key**) approaches, automated auditing, formal evaluation of security features and security products, and decision support through comprehensive system analysis techniques. Although the available technology is sophisticated and effective, no computer security protective measures are perfect, so the goal of prevention (security assurance) is almost always accompanied by detection (early discovery of security penetration) and penalty (denial of goal, e.g., information destruction; or response, e.g., prosecution and punishment) approaches.

The information in this section is intended to survey the major contemporary computer security threats, vulnerabilities, and controls. A general overview of the security environment is shown in Fig. 97.1. The oval in the figure contains an indication of some of the crucial concentrations of resources that exist in many facilities, including digital representations of money; representations of information about operations, designs, software, and

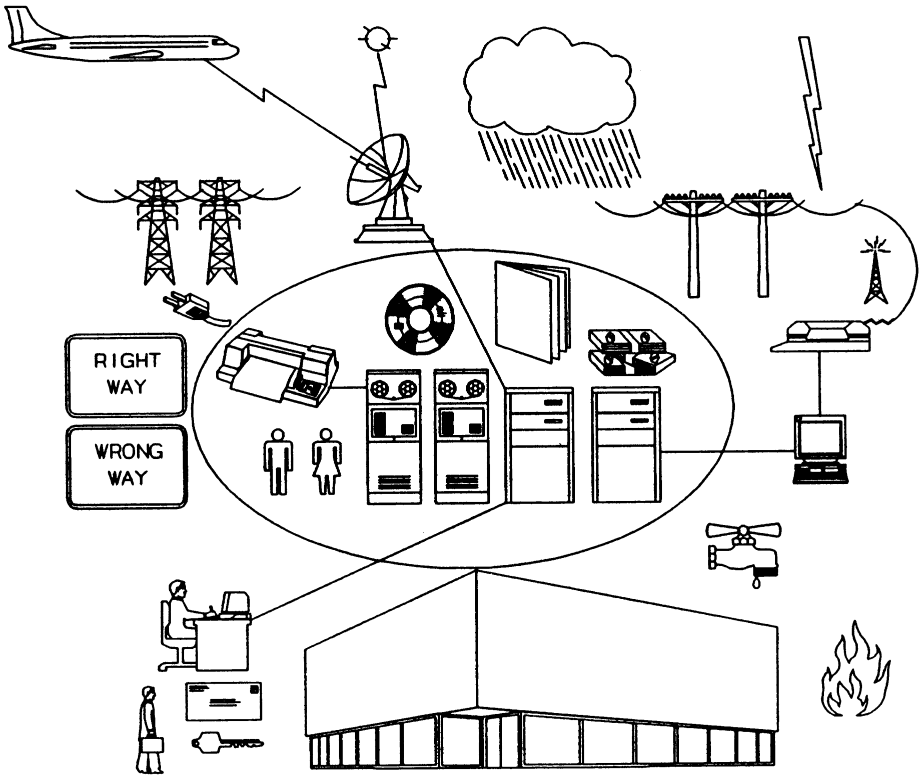

FIGURE 97.1 An overview of the computer and communications security environment.

people; hardware for carrying out (or peripheral to) computing and communications; people involved in operating the facility; utility connections (e.g., power); and interconnection paths to outside terminals and users, including hard-wired connections, modems for computer (and FAX) communication over telephone lines, and electromagnetic links (e.g., to satellite links, to ground antenna links, and to aircraft, spacecraft, and missiles). Each of these points of termination is also likely to incorporate computer (or controller) processing.

Other factors implied include the threats of fire, water damage, loss of climate control, electrical disturbances (e.g., due to lightning or power loss), line taps or **TEMPEST emanations** interception, probes through known or unknown dial-up connections, unauthorized physical entry, unauthorized actions by authorized personnel, and delivery through ordinary channels (e.g., mail) of information (possibly misinformation) and software (possibly containing embedded threat programs). Also indicated is guidance for personnel about acceptable and unacceptable actions through policy and regulations. The subject breadth can be surveyed by categorizing into physical security, cryptology techniques, software security, hardware security, network security, and personnel security (including legal and ethical issues). Because of the wide variety of threats, vulnerabilities, and assets, selections of controls and performance assessment typically are guided by security-specific decision-support analyses, including risk analysis and probabilistic risk assessment (PRA).

Physical Security

Physical access security ranges from facility access control (usually through personal identification or authentication) to access (or antitheft) control for individual items (e.g., diskettes and personal computers). Techniques used generally center around intrusion prevention (or invoking a significant time delay for an adversary) and intrusion detection, which allows a response through security guard, legal or administrative action, or automatic devaluation of the penetration goal (e.g., through information destruction) [Cooper, 1989].

Physical environmental security protects against natural threats, such as power anomalies or failures, water damage, fire, earthquake, and lightning damage, among others. An example suited to computer requirements

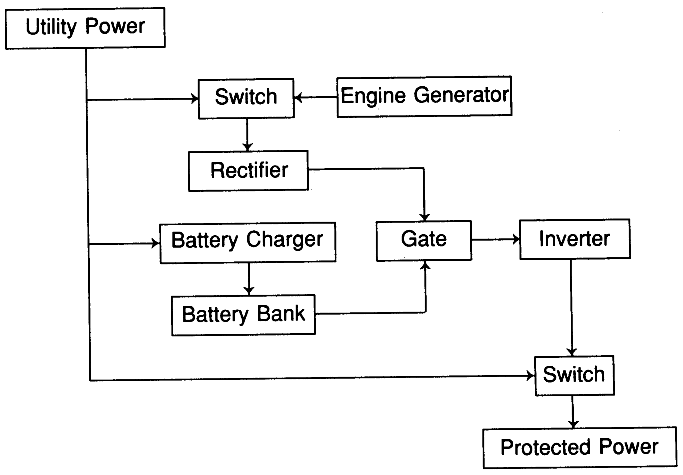

FIGURE 97.2 Uninterruptible power system.

is Halon fire suppression (although Halon use is now being replaced because of environmental concern). Note that some of the natural threats can also be adversary-caused. Since there is potential (in spite of protection) for a loss, contingency planning is essential. This includes provisions for software backup (usually off-site), hardware backup (e.g., using reciprocal agreements, hot sites, or cold sites [Cooper, 1989]), and disaster recovery, guided by a structured team that has prepared through tests (most typically simulated).

An example of power protection technology is the widely used uninterruptible power system (UPS). An online UPS implementation is shown in Fig. 97.2. Utility power is shown passed through a switch to a rectifier and gated to an inverter. The inverter is connected to the critical load to be protected. In parallel, continuous charge for a battery bank is provided. Upon loss of utility power, the battery bank continues to run the inverter, thereby furnishing power until graceful shutdown or switching to an auxiliary engine generator can be accomplished. The switch at the lower right protects the UPS by disconnecting it from the load in case of a potentially catastrophic (e.g., short) condition.

Cryptology

Cryptology includes techniques for securely hiding information (encrypting) from all but intended recipients, for authenticating messages, and for digital signatures, all through the use of ciphers (cryptosystems) [Simmons, 1992]. It also includes techniques for deducing at least a subset of encrypted information (cryptanalysis) without the privileged knowledge possessed by the intended recipients. Cryptanalysis knowledge is an important asset in the development of cryptosystems. An example of a contemporary measure of cryptanalysis resistance is *computational complexity,* which can be applied to measure the inherent difficulty of numeric cryptanalysis processing for some cryptosystems. Figure 97.3 shows the main components of cryptology. The information to be protected is called plaintext (cleartext), and protected information is called ciphertext. Adversaries can passively obtain ciphertext, or they might actively interrupt the communication link and attempt to spoof the information recipient.

Some of the objectives of encryption are secrecy, authentication (assurance to recipient of sender identity), and digital signatures (authentication plus assurance to the sender and to any third parties that the recipient could not have created the signature). As in physical security, assurance of integrity means preventing interference in the information-conveying process or, failing that, detecting interference. Here, interference may have the aims of eavesdropping, modifying, introducing misinformation, disavowing messages, and falsely claiming receipt of messages.

Almost all cryptosystems involve transformations (frequently made public and almost always assumed to be known by adversaries) of information based on one or more *keys* (see Fig. 97.3), at least one of which must be kept secret to protect against adversaries. A single-key (symmetric) cryptosystem has only one secret key,

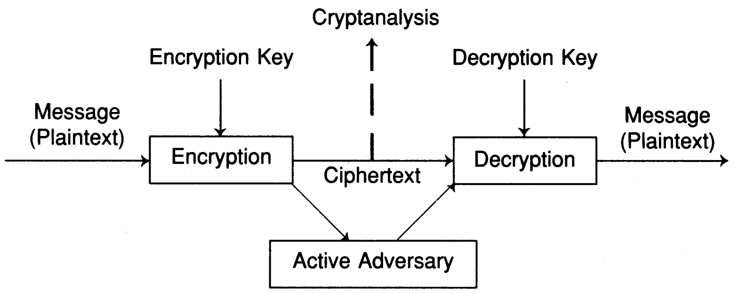

FIGURE 97.3 Basic cryptosystem functions.

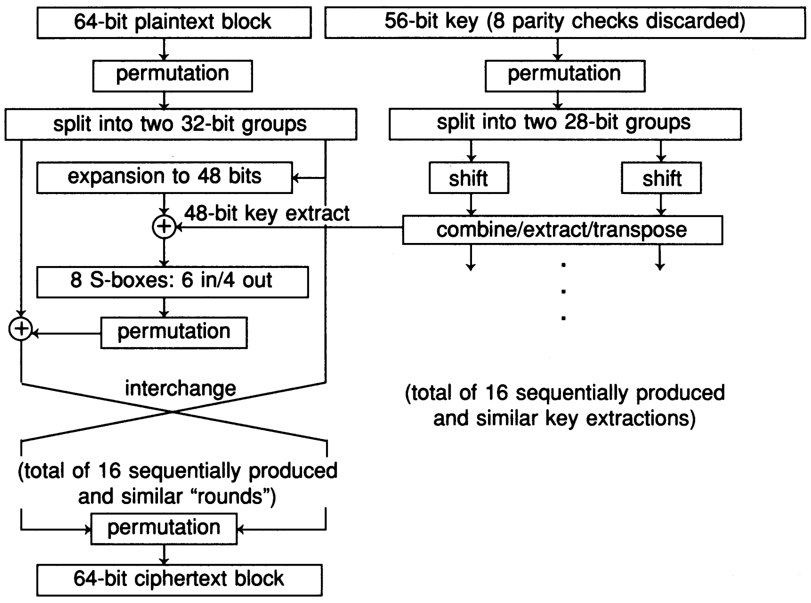
64-bit plaintext block
56-bit key (8 parity checks discarded)
permutation
permutation
split into two 32-bit groups
split into two 28-bit groups
expansion to 48 bits
shift
shift
48-bit key extract
combine/extract/transpose
8 S-boxes: 6 in/4 out
permutation
interchange
(total of 16 sequentially produced and similar key extractions)
(total of 16 sequentially produced and similar "rounds")
permutation
64-bit ciphertext block

FIGURE 97.4 Basic function of the DES algorithm.

which is used to encrypt information by the sender and to decrypt information by the recipient. A prior secure process is necessary so that both sender and recipient know (and no adversary knows) the key.

The most well-known and most widely used single-key cryptosystem in history is the Data Encryption Standard (DES), published by the U.S. National Bureau of Standards [1977] (now the National Institute of Standards and Technology, NIST), with National Security Agency (NSA) consultation. DES utilizes a 56-bit key (some *weak* and *semi-weak* keys are excluded) to encipher information in blocks of 64 bits. It involves substitution and permutation, linear and nonlinear transformations, and 16 successive "rounds" of key-dependent processing (general indication of logic shown in Fig. 97.4). The DES cryptosystem is identical for encryption and decryption, except that the order of application of the 16 key extractions is reversed. Like most cryptosystems of this type, DES is usually used with some form of *chaining* (mixing ciphertext or information that produces ciphertext from one block with plaintext or information that produces ciphertext in the subsequent block at the transmitter, and then inverting the process at the receiver). Three chaining techniques specified for DES (and usable in most other cryptosystems) are indicated in Fig. 97.5, along with the basic electronic codebook block form. The k bits shown are typically eight bits, and these are shifted into the first k positions of a shift-register/buffer after each encryption. Coordinated time stamps or initial values (IVs) are used to prevent identical transformation for each system start.

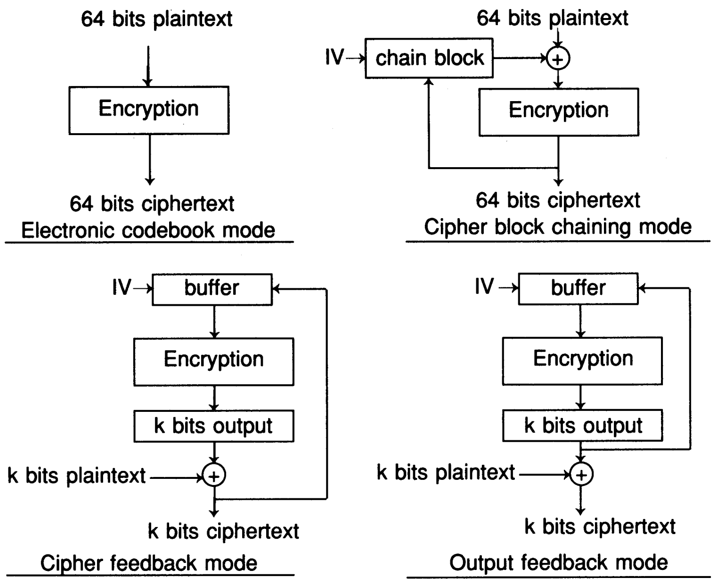

FIGURE 97.5 Modes of use for block cryptosystems.

Although the DES key length was acceptable to most users when the standard was released in 1977, increases in computing power have made exhaustive search less expensive, so the relative security of DES has decreased. NSA now supports some of its own secret algorithms as DES replacements ("COMSEC Commercial Endorsement Program, Type II" devices), although NIST support for DES continues and no algorithmic weaknesses in DES have been publicly revealed.

Public-key cryptosystems [Diffie and Hellman, 1976] use two different keys (asymmetric systems). For example, information can be encrypted with one key and decrypted with a different (but related through a secure process) key. If the aim is secrecy, the decryption key must be secret so only the recipient can decrypt. In this case, however, the encryption key can be publicly known and known to be associated with a particular potential recipient. Although the sender can be assured of information secrecy in this process, the recipient cannot be assured of sender authenticity. If the secret key of a pair of keys is used by a sender to encrypt, any recipient who knows the sender's public key can be assured of sender authenticity, but there is no assurance of secrecy. If the public-key cryptosystem has commutative transformations (as does the RSA cryptosystem), encryption with the sender's secret key and with the recipient's public key for encipherment, and decryption by the recipient with his or her secret key and with the sender's public key provides both secrecy and authenticity.

RSA (named after Rivest, Shamir, and Adleman) is the most well known and most widely used public-key cryptosystem. Unlike DES, the key length of RSA encryption is user-selectable. However, the length chosen must be securely long (long enough that knowledge of the public key is not helpful in determining the secret key). Key selection begins with the choice of two prime numbers, each can be approximately 150 decimal digits long, giving about a 300-digit number on which the RSA encryption is based [Eq. (97.1)]. The security of the system depends on the difficulty of factoring large numbers that have no relatively small factors. Equation (97.2) shows how a secret modulus is determined, and Eq. (97.3) shows how the modulus is used to relate the secret key and the public key. Equation (97.4) gives the RSA encryption process, and Eq. (97.5) gives the RSA decryption process. An adversary who could factor n could use Eq. (97.2) to determine the modulus, ϕ, and then the secret key, d, from Eq. (97.3), given the public key, e.

$$n = pq \tag{97.1}$$

$$\phi = (p - 1)(q - 1) \tag{97.2}$$

$$ed = 1 \pmod{\phi} \tag{97.3}$$

$$C = M^e \pmod{n} \tag{97.4}$$

$$M = C^d \pmod{n} \tag{97.5}$$

For equivalent security, the computational burden of RSA and similar public-key cryptosystems is significantly greater than DES and similar single-key cryptosystems. As a result, where large amounts of information must be communicated, public-key systems are frequently used for secure communication of a key intended for a single-key system, which is then in turn used for mainstream encryption.

RSA has well known cryptographic digital signature capabilities (transformed by the sender using the sender's secret key; transformed by the receiver using the sender's public key), which gives assurance that the information was initiated by the signer and that the sender cannot deny creating the information. A signature technique, Digital Signature Standard (DSS) [NIST, 1991], has been proposed by NIST. The basic differences between DSS and RSA are that DSS is intended only for digital signatures, DSS patents are intended to be government owned, the proposed DSS key lengths will be constrained, and the security of DSS is based on the difficulty of finding logarithms of large numbers.

Examples of relatively new encryption techniques coming into popular use are PGP (Pretty Good Privacy), IDEA (International Data Encryption Algorithm), and PEM (Privacy Enhanced Mail). The U.S. Government has proposed SKIPJACK, a secret and controlled system, as an intended replacement for DES. The proposal, which includes "trusted third-party" key escrow, has met with significant controversy.

Software Security

A number of techniques that are commonly implemented in software can contribute to protection against adversaries. These include password authentication; memory, file, and database access restrictions; restrictions on processing actions; development and maintenance controls; and auditing.

Passwords, which are intended to authenticate a computer user in a cost-effective way, are sometimes user-selected (a technique resulting in a relatively small potential population), sometimes user-selected from a computer-generated collection, sometimes randomly generated, and sometimes randomly generated from a phonetic construction (for pronounceability and memorization ease) [Cooper, 1989]. Examples of phonetic passwords are TAMOTUT, OTOOBEC, SKUKOMO, ALTAMAY, and ZOOLTEE. These five were each chosen from a different phonetic construction (five of the approximately 25 commonly used).

Security control can be physical, temporal, logical, or procedural. Two important logical or procedural control principles are part of fundamental multilevel security (multiple levels of sensitivity and multiple user clearance levels on the same system), as described by part of the Bell–La Padula model. The simple security principle restricts users of a particular clearance level from reading information that is of a more sensitive (more highly classified) level. The star property prohibits information flow from the level at which its sensitivity has been determined to any lower level (write-down). Analogous integrity protection is provided by the Biba integrity model [Gasser, 1988].

Protection rules can be mandatory (used mainly by the government or military) or discretionary (compartmented according to need-to-know regimes of trust typically determined by file owners). The combination of security levels and protection rules at the same level can be associated with a lattice model. In addition to matching the security controls, the lattice model facilitates mathematical verification of security implementations.

A common logical protection rule specification gives the rights of subjects (action initiators) to act on objects (action targets) at any particular time. One way to view these rules (although seldom implemented in this manner) is to consider an access matrix (Table 97.1) containing rows for subject indicators and columns for object indicators. The matrix entries are the *rights* of subjects to objects. Actual implementation may differ, e.g., by using directories, or *capability lists,* or capability tokens (row designations for rights of subjects) or *access control lists* (column designation for rights to objects).

These types of rules can be augmented by software (and/or hardware) memory protection through techniques including fences, base/bounds registers, tagged registers, and paging [Gasser, 1988].

Database management system (DBMS) security and integrity protections include access controls but generally require finer granularity and greater protection (especially for relational databases) against subtle forms of

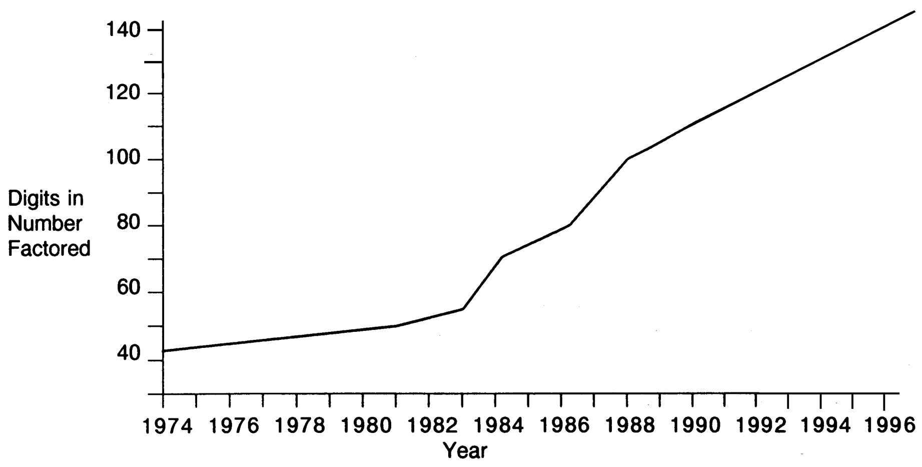

FIGURE 97.6 Factoring history. Because of the importance of factoring to RSA security, factoring methodology and accomplishments are of considerable interest. Techniques for factoring "hard" numbers were available for only up to about 50 digits in about a day's computing time until 1983, when a match between mathematical development (the quadratic sieve) and computer vector processing capabilities contributed to factoring up to 58-digit numbers in equivalent time. The next year, a 69-digit number was factored in about 32 hours on a Cray 1S. A few months later, a 71-digit number was factored in less than 10 hours on a Cray XMP. By the end of the decade, collections of small computers had been coupled in a worldwide effort to demonstrate that numbers of more than 100 (116 in 1991) digits could be cost-effectively factored. This explosive trend, although not expected to continue because of current mathematical limitations (at present many orders of magnitude more computation time is needed than would threaten 300-digit numbers), demonstrates the importance of factoring prognosis in forecasting the long-term security of RSA.

TABLE 97.1 An Access Matrix

Subjects/Objects	*O*1	*O*2	*O*3	*O*4	*O*5
S_1	Own, write, read	Own, read, execute	Own, read, delete	Read, write, execute	Read
S_2		Read	Execute		Read
S_3	Write		Read		Read

information deduction such as inference and aggregation. Integrity protection mechanisms include field checks, change logs, two-phase updates, error protection codes, range comparisons, and query controllers [Pfleeger, 1989]. Secrecy depends on access control (e.g., file passwords), query controllers, and encryption.

Processing restrictions can, in addition to those implied by memory, file, and database controls, limit the ability of users to, for example, try multiple passwords or multiple user IDs; make financial transactions; change security parameters; move, rename, or output information; and deliver covert channel information (signaling systematically using authorized actions to codify unauthorized data delivery).

Software development and maintenance controls include standards under which programs (including security features) are designed to meet requirements, coded in structured or modular form, reviewed during development, tested, and maintained. Configuration or change control is also important. Computer auditing is intended to provide computer records about user actions for routine review (a productive application for expert systems) and for detailed investigation of any incidents or suspicious circumstances. It is essential that audit records be tamper-proof.

Software security features (including auditing) can be provided as part of the computer operating system or they can be added to an operating system as an add-on product. A U.S. government multilevel *trusted computing base* development program through NSA's National Computer Security Center (NCSC) resulted in a well known security methodology and assessment scheme for these types of software (and hardware) products [DOD, 1985]. A significant number of operating systems and software security packages have been evaluated and given

TABLE 97.2 NCSC Security Evaluation Ratings

Class Name	Summary of Salient Features
Class A1	Formal top-level specification and verification of security features, trusted software distribution, covert channel formal analysis
Class B3	Tamper-proof kernelized security reference monitor (tamper-proof, analyzable, testable), structured implementation
Class B2	Formal security model design, covert channel identification and tracing, mandatory controls for all resources (including communication lines)
Class B1	Explicit security model, mandatory (Bell–La Padula) access control, labels for internal files and exported files, code analysis and testing
Class C2	Single-level protection for important objects, log-in control, auditing features, memory residue erasure
Class C1	Controlled discretionary isolation of users from data, authentication, testing
Class D	No significant security features identified

ratings by NCSC, in addition to hardware–software combinations, encryption devices, and network security systems. The basic evaluation determines the degree of confidence that the system will be resistant to external penetration and internal unauthorized actions. The most secure systems known are classified A1 and utilize a reference monitor (checking every request for access to every resource), a security kernel (concentration of all security-related functions into a module that facilitates protection and validation), and protection against covert channels. Formal analysis is used to assure that the implementation correctly corresponds to the intended security policy. There is an operational efficiency penalty associated with secure multilevel operating systems.

Other classes (in order of progressively fewer security features, which results in decreasing security) are B3, B2, B1, C2, C1, and D (see Table 97.2, where security features generally accumulate, reading up from the table bottom).

In addition to computer activity directly controlled by personnel, a family of software threats can execute without direct human control. These techniques include the **Trojan horse**, the **virus,** the **worm**, the **logic bomb**, and the **time bomb**. The virus and worm (because they copy themselves and spread) are both capable of global-spanning attacks over relatively short time frames. Protection against these threats includes limiting user threats through background screening, using expert system software scanners that search for adversarial program characteristics, comparators, and authenticators or digital signatures that facilitate detection of software tampering.

Other software-intensive threats include trapdoors, superzapping, browsing, asynchronous attacks, and the salami attack [Cooper, 1989]. These all usually involve unauthorized actions by authorized people and are most effectively counteracted by insider personnel controls (see Section 97.7, "Personnel Security").

Hardware Security

In addition to personal authentication through something known (e.g., passwords or PINs), users can be authenticated through something possessed or by something inherent about the user (or by combinations of the three). Hardware devices that contribute to computer security using the approach of something possessed include **tokens** and **smart cards. Biometric verifiers** authenticate by measuring human characteristics. Other hardware security devices include encryptor/decryptor units and **port protection devices** (to make dial-up attacks by hackers more difficult). A generic diagram depicting some of these applied to control of users is shown in Fig. 97.7. The controls can be used individually or in various combinations.

Tokens are devices that can be hand-carried by authorized computer users and are intended to increase password security by assuring that passwords are used only once, thereby reducing the vulnerability to password compromise. The devices contain an internal algorithm, which either works in synchronization with an identical algorithm in the host computer or transforms an input derived from a computer prompt into a password that matches the computer-transformed result. In order to protect against loss, most also require a user password for token access.

Smart cards are credit-card-sized devices intended to facilitate secure transactions, such as credit card purchases, purchases or cash withdrawals that result in bank account debits, or information interchanges. The most common application uses a card reader/network that exchanges data with the smart card over a serial data bus. User information and security information are stored in encrypted form in the card, and physical

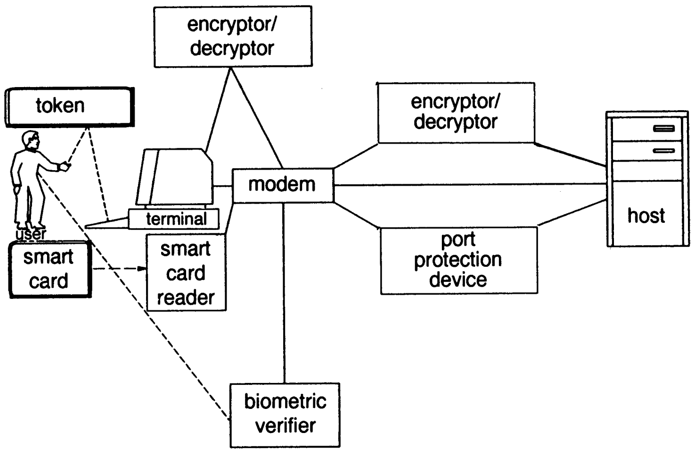

FIGURE 97.7 Depiction of hardware controls.

access to the internal card circuitry is protected by tamper-proof (self-destructive) sealing. Use of the card is controlled by password access.

Because of the vulnerability of passwords to compromise by disclosure or various forms of information tapping, and because of the vulnerability of loss of carried items (e.g., ROM keys, magnetic stripe cards), biometric devices have been developed to measure human characteristics in ways that are resistant to counterfeiting. These devices include signature verifiers (for examining the velocity, acceleration, and pressure characteristics imparted during signing as a function of time), fingerprint and palmprint readers (for examining print pattern characteristics, for example, with the flesh applied to a glass platen), voice verifiers (which evaluate speech characteristics, usually in response to system prompts), hand geometry (including some three-dimensional aspects), eye retina vessel pattern examination (through infrared reflection), and typing rhythm assessment (for user keyboard inputs).

Systematic cracker attacks on dial-up computer ports frequently include searches for modem tones followed by attempts to guess passwords. In response, port protection devices (PPDs) enhance dial-up security. The basic feature of many PPDs is that no modem tone is provided until an additional security barrier (or barriers) is overcome. Most PPDs require a code before computer port connection. Some also identify the user by the code entered, disconnect the call, and dial the number at which the user is expected to be (typically using a separate line to avoid dial-in intercept of the outgoing call).

Personal computer (PC) security is of contemporary interest because these relatively new tools have contributed to a set of security vulnerabilities that differs substantially from conventional computer security concerns. For example, PC users may be more naive about security in general, PC hardware and software and administrative controls are generally more primitive, the PC physical environment is generally less controlled, and PCs are generally more easily misused (e.g., company PCs used for personal benefit).

An additional hardware security topic is associated with TEMPEST (a program to assess the potential for data processing equipment to inadvertently generate "compromising emanations" that convey information to a surreptitious remote sensor). Although originally of concern because of requirements to protect government and military classified data, industrial espionage is now also a concern. Various forms of protection can be used, such as electromagnetic shielding, physical separation of processing equipment from potential adversary locations, fiber-optic communication, and encrypted data transmission. Some commercial equipment has been certified by NSA to have low emanations.

Network Security

Many business, informational, and scientific interchanges take place nationally and internationally over networks under computer control. Management of network security is exacerbated by physical dispersal and security philosophy disparity. For example, network adversaries may be harder to identify and locate than local

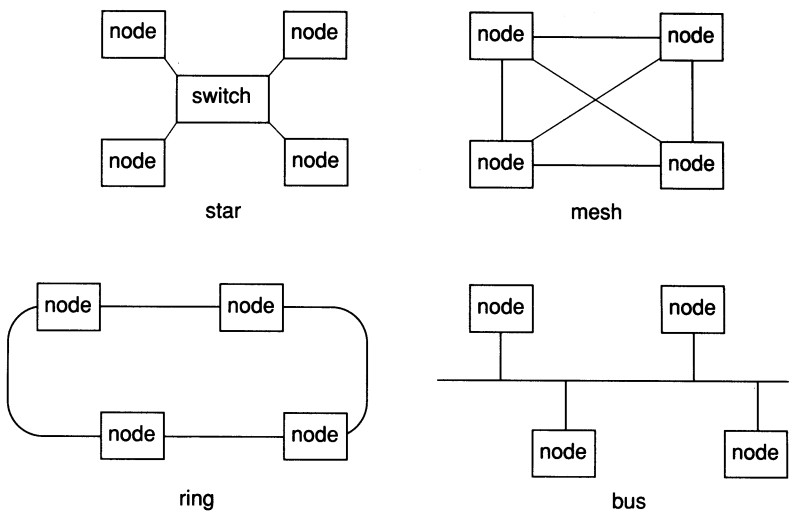

FIGURE 97.8 Basic network topologies.

computer adversaries. (For an interesting account of overcoming this problem, see [Stoll, 1989].) As another example, a user at a location that would not allow some form of activity (e.g., copying information from one level of security to a lower level) might find a network connection to a facility for which the activity was accepted. The intended local restriction might thereby be circumvented by conducting the restricted activity at the more permissive location. Opportunities for passive interception (tapping or emanations), or for active spoofing (involving misinformation, replay, etc.), or for disruption (including jamming) are also generally greater owing to network utilization.

There are many network topologies, but they can be decomposed into four basic canonical types (Fig. 97.8). The star topology has been traditionally used in centrally controlled networks (e.g., the control portion of the telephone system), and security is typically within the central control. Use of star topology in local-area networks (LANs) is increasing. Mesh topology is not readily amenable to central control but is well tailored to protect wide-area network integrity. Mesh topology accommodates variable-path routing schemes, such as packet transmission. The bus topology is commonly used in complex physically constrained systems, such as computing and communication processor interconnection, and missiles and aircraft. The ring and bus topologies are frequently used in LANs. The shared communication media for LANs can jeopardize secrecy unless communications are encrypted.

Network security considerations include secrecy, integrity, authenticity, and covert channels. Potential controls include cryptosystems (for secrecy, integrity, and authentication); error-protection codes, check sums (and other “signatures”), and routing strategies (for integrity); protocols (for key distribution and authentication); access control (for authentication); and administrative procedures (for integrity and covert channel mitigation). Where encryption is used, network key distribution can be difficult if the physical dimensions of the network are large. Note that several techniques described under hardware security (smart cards, tokens, biometrics, PPDs) are useful for network authentication.

Various network security approaches have been used; they can be basically classified into centralized security and distributed security (although the use of combinations of the two is common). It is difficult to maintain effective centralized administrative control in a network larger than a LAN, because of the logistics of maintaining current security and authentication data. Network efficiency, reliability, and integrity are also limited by the performance of the central security controller. The major weaknesses of distributed control are associated with inconsistent security enforcement and the security-relevant communication burden.

Networks frequently comprise networks of networks (the Internet is a worldwide interconnection of networks) and *firewalls* (filter between outside and internal networks), using *bridges* or *routers* for protocol pass-through and filtering and gateways for protocol translation and buffering. Bridges, routers, gateways, and firewalls may also have the role of distributed network security controllers.

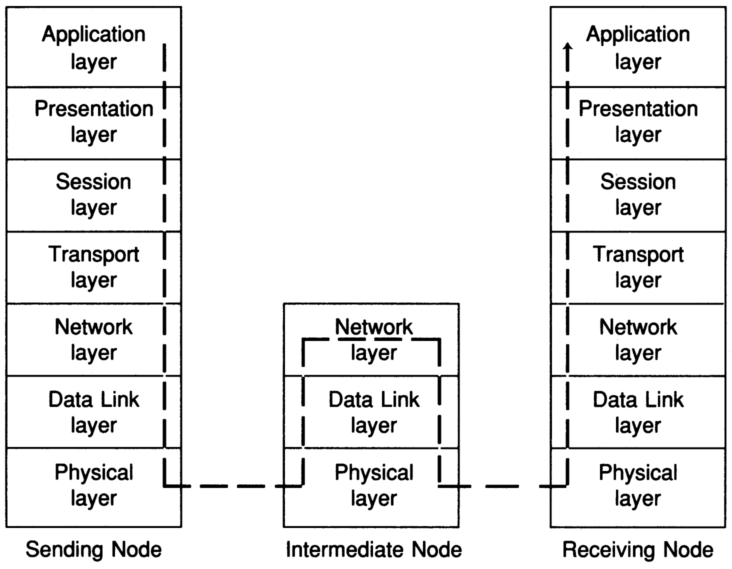

FIGURE 97.9 The ISO network protocol model.

Various security protocols (orderly coordinated sequences of steps) are used to authenticate network users to each other. The basic purpose of security protocols is to assure all of the parties involved that the parties with whom they are communicating are behaving as expected. Protocols can be arbitrated, adjudicated, or self-enforcing [Pfleeger, 1989]. Analogously, cryptographic *sealing* and cryptographic *time stamps* (or other sequence identifiers) can prevent message (or message segment) reuse. These approaches can authenticate the message contents as well as the communicating party. Secure key distribution protocol (e.g., by a network key server or key distribution center) is an important application for protocols.

Network communication involves several levels of nonsecurity protocol for the purpose of allowing users to communicate with integrity. A number of network standards have been developed (e.g., TCP/IP, ISDN, GOSIP, SNMP, SMTP, and http for the World Wide Web). An important example is the International Standards Organization Open Systems Interconnection model (OSI). The basic OSI structure is shown in Fig. 97.9 [Stallings, 1995].

The physical layer mediates access to the transmission medium. Network systems such as token ring, token bus, and carrier sense multiple access with collision detection (CSMA/CD) work at this level. The data link layer can be used to enhance transmission quality through framing, error correction, and check sums. Link (point-to-point) encryption is typically implemented in hardware at this level. The network layer handles network routing functions. This is the highest layer necessary for an intermediate node, as shown in the figure. Correct routing (and protection for pass-through information from users at the intermediate node) is important to security. The transport layer provides end-to-end (initial source to final destination) interprocess communication facilities. End-to-end encryption can be implemented in software at this level. The session layer manages the overall network connection during a user activity. This connectivity is also important to security. The presentation layer converts between user syntax and network syntax, and the application layer provides user services such as electronic mail and data file transfers. Encryption can be implemented at either of the latter two layers.

Link encryption requires exposure of information at intermediate nodes. While this is essential (at least for routing data) if further routing is required, it may be required to protect information from exposure at intermediate nodes. In addition to routing controls, this may require end-to-end encryption and separation of routing information from protected information. End-to-end encryption is generally performed at the higher levels (e.g., the transport layer). It is not unusual to use both link and end-to-end encryption.

Like isolated computers, networks can have multilevel security. However, implementation and verification of security features are more difficult. For example, covert channel protection is currently quite problematic in networks.

Personnel Security

Personnel security involves protecting personnel as assets and protecting against personnel because of the threat potential. The basic aspects of the latter topic are motivations that cause various types of threats, the approaches most likely to be used by adversaries, techniques for assessing the threat potential, and techniques for protecting against adversaries.

One motivation for computer/communication attacks is financial gain. This motivation ranges from career criminals who could view the volume of monetary transactions as tempting, and the potential detachment from personnel confrontations as affording low risk of apprehension, to basically honest people who have trouble resisting what they consider to be less serious temptations. Industrial espionage is one of many examples of financial motivation that may result in attempts to evade computer security control.

Another important motivation is information gain or information modification, which could represent no direct financial gain. Some people are curious about information to which they have no right. Some want to modify information (e.g., grades, criminal records, medical records, personnel records) because it reflects an image they want to change.

The motivation of causing personal or public outrage in order to advance a cause is a common motivation. Terrorism is an example, and many acts of terrorism against computers have occurred [Cooper, 1989], especially in Europe. Sometimes the cause is related to revenge, which may be manifested through vandalism.

Espionage activities can be motivated by financial gain, national or company loyalty, blackmail, or even love. Hackers are frequently motivated by the challenge of overcoming security barriers. Usually, self-image and image with peers (e.g., through electronic bulletin board proclamations of breakthroughs) is a strong factor.

Personnel adversaries most commonly choose what they perceive to be the easiest and/or the safest avenue to achieve the desired objective. This is analogous to looking for the weakest link to break a chain. Because these avenues may be either inherent or unknown, security barrier uniformity is frequently sought through the application of basic principles (e.g., separation of duties). Some adversaries are motivated enough and skilled enough to use ingenious approaches, and these provide a warning about what unexpected approaches might succeed. One of the most interesting and informative examples was the break by William Friedman of the Vernam cipher used by the Germans in the Second World War [Cooper, 1989]. Another was the use of an unintended electronic mail program feature that allowed an adversary to plant a **Trojan horse** in a privileged area of a computer, which resulted in system privileges when the computer ran routine periodic housekeeping. This was the genesis of the title of the book *The Cuckoo's Egg* [Stoll, 1989]. The same adversary was one of several known to have broken one-way transformed password encryption by downloading the transform and the transformed outputs to his own computer and then exhaustively encrypting a dictionary of words and potential passwords, noting where matches to transformed outputs were obtained.

Approaches used to assess the types of unexpected attacks that might be used are mainly to catalog past approaches that have been used and to foresee new approaches through adversarial simulation. An example of this simulation is the "tiger team" approach, where a collection of personnel of various backgrounds synergistically brainstorm approaches, at least some of which may be tested or actually carried out. Tiger teams have a long history of finding unexpected approaches that can be successful, and thereby identifying protection needs.

Protection against personnel attacks generally falls into two categories: protection against insiders (those having some authorization to use resources) and against outsiders (those who gain access to resources in unauthorized ways). Some protective measures are tailored to one or the other of these two groups; some are applicable in either case.

Typical protections against unauthorized insider activities include preemployment screening and background investigation, polygraph examinations (within legal limits), administrative controls (e.g., security plans and access logs), routine examination and monitoring of activities through audit records, ethics and motivational training, and the threat of legal or job-related punishment for improper activities. The ethics of computer use varies from organization to organization because of society's inexperience in weighing the moral aspects of the topic.

Protection against outsiders includes physical and logical access control using the various forms of hardware and software authentication discussed previously and the threat of legal prosecution for transgressions. This threat depends in large measure on the available laws covering computer security violations. Computer security laws and law enforcement have traditionally been weak relative to laws covering other types of activities (largely because of new legal aspects associated with computing), but a large number of legal approaches are now possible because of laws enacted during the past two decades.

Computer laws or laws that apply to computing include the Copyright Act of 1976 (amended in 1980 to allow software copyrights and help protect against **software piracy**), the Patent Act of 1980 (adding firmware and software coverage), "shrinkwrap licenses" (some legal protection, some deterrent), the Computer Crime Statute of 1984 (applicable to U.S. government computers), the Privacy Act of 1974 (for U.S. government applications and similar to privacy laws in a number of other countries), the National Security Decision Directive (NSDD) 145 (for NSA-enhanced protection of "sensitive unclassified" information, largely intended to prevent technology drain to unfriendly countries), the Computer Security Act of 1987 (restoring NIST as the primary agency responsible for sensitive unclassified security), the Right to Financial Privacy Act of 1978, the Freedom of Information Act of 1966, the Electronic Funds Transfer Act, the Fair Credit Reporting Act, the Crime Control Act, the Electronic Communications Privacy Act, the Computer Fraud and Abuse Act, and the Foreign Corrupt Practices Act.

There is now considerable interest in international legal computer communication agreements, especially among countries that interchange significant amounts of computer data. International forums have brought many countries together with the intents of regulatory commonality and transborder data communication control.

Defining Terms

Biometric verifiers: Devices that help authenticate by measuring human characteristics.

Hacker: Person who explores computer and communication systems, usually for intellectual challenge, commonly applied to those who try to circumvent security barriers (crackers).

Internet: An international connection of networks which can be navigated by the World Wide Web protocols, and over which e-mail, information transfers, and credit card orders can transverse.

Logic bomb: Destructive action triggered by some logical outcome.

Port protection device: Device in line with modem that intercepts computer communication attempts and requires further authentication.

Public-key cryptosystem: System that uses a pair of keys, one public and one private, to simplify the key distribution problem.

Smart cards: Credit-card-sized devices containing a microcomputer, used for security-intensive functions such as debit transactions.

Software piracy: Unauthorized copying of software for multiple uses, thereby depriving software vendor of sales.

TEMPEST emanations: Electromagnetic, conductive, etc. leakage of information that can be recovered remotely.

Time bomb: Destructive action triggered by computer calendar/clock reading.

Token: Device that generates or assists in generation of one-time security code/passwords.

Trojan horse: Implanted surreptitious code within an authorized program.

Virus: A self-replicating program that is inserted in an application program or other executable routine.

Worm: Independent self-replicating code that, once initiated, migrates across networks, consuming memory resources.

Related Topics

70.1 Coding • 92.2 Local Area Networks

References

K. Hafner and J. Markoff, *Cyberpunk*, New York: Simon and Schuster, 1991.
J. A. Cooper, *Computer and Communications Security*, New York: McGraw-Hill, 1989.
G. J. Simmons, *Contemporary Cryptology*, Piscataway, N.J.: IEEE Press, 1992.

National Bureau of Standards, “Data Encryption Standard,” *FIPSPUB 46,* Washington, D.C., January 1977.
W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Trans. Inf. Theory,* vol. IT-22, November 1976.
National Institute of Standards and Technology, “Digital Signature Standard (Draft),” *FIPSPUB,* Washington, D.C., August 1991.
M. Gasser, *Building a Secure Computer System*, New York: Van Nostrand Reinhold, 1988.
C. P. Pfleeger, *Security in Computing*, Englewood Cliffs, N.J.: Prentice-Hall, 1989.
“Department of Defense Trusted Computer System Evaluation Criteria,” DOD 5200.28-STD, December 1985.
C. Stoll, *The Cuckoo’s Egg*, New York: Doubleday, 1989.
W. Stallings, *Networks and Internet Security,* Englewood Cliffs, N. J.: Prentice-Hall, 1995.

Further Information

B. Bloombecker, *Spectacular Computer Crimes*, Homewood, Ill.: Dow-Jones-Irwin, 1990.
P. J. Denning, *Computers Under Attack*, New York: ACM Press, 1990.
D. E. Robling Denning, *Cryptography and Data Security*, Reading, Mass.: Addison-Wesley, 1982.
P. Fites, P. Johnston, and M. Kratz, *Computer Virus Crisis*, New York: Van Nostrand Reinhold, 1989.
J. L. Mayo, *Computer Viruses*, Blue Ridge Summit, Pa.: Windcrest, 1989.
National Research Council, *Computers at Risk*, Washington, D.C.: National Academy Printers, 1991.
E.F. Troy, “Security for Dialup Lines,” U.S. Department of Commerce, Washington, D.C., 1986.
“Trusted Network Interpretation,” NCSC-TG-005, Ft. George G. Meade, Md.: National Computer Security Center, 1987.
C. Kaufman, R. Perlman, and M. Speciner, *Network Security,* Englewood Cliffs, N. J.: Prentice-Hall, 1995.

97.2 Fundamentals of Cryptography

Oded Goldreich

Cryptography is concerned with the construction of systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality. Given a desired functionality, a cryptographer should design a system that not only satisfies the desired functionality under “normal operation,” but also maintains this functionality in the face of adversarial attempts to abuse it. These attempts are devised after the cryptographer has completed his/her work, by an adversary who tries to take actions other than the ones the designer had envisioned. Thus, it makes little sense to make assumptions regarding the specific strategy that the adversary may use. The only assumptions that can be justified refer to the computational abilities of the adversary.

The design of cryptographic systems must be based on firm foundations; ad hoc approaches and heuristics are a very dangerous way to go. A heuristic might make sense when the designer has a very good idea about the environment in which a system is to operate. In contrast, a cryptographic system must operate in a maliciously selected environment that typically transcends the designer’s view. The design of good cryptographic systems consists of two main steps:

1. The Definitional Step: The identification, conceptualization, and rigorous definition of a cryptographic task that capture the intuitive security concern at hand.
2. The Constructive Step: The design of cryptographic schemes satisfying the definition distilled in Step 1, possibly while relying on widely believed and well-understood intractability assumptions.

Note that most of modern cryptography relies on intractability assumptions. Relying on assumptions is unavoidable (in the sense discussed below in “Computational Difficulty”). Still, there is a huge difference between relying on a simple, explicitly stated assumption and just assuming that an ad hoc construction satisfies some vaguely specified (or even unspecified) goals.

Although encryption, signatures, and secure protocols are the primary tasks of cryptography, this section starts with basic paradigms and tools such as computational difficulty, pseudorandomness, and zero-knowledge. Once these are presented, encryption, signatures, and secure protocols are discussed.

Central Paradigms

Modern cryptography, as surveyed here, is concerned with the construction of *efficient* schemes for which it is infeasible to violate the security feature. Thus, we need a notion of efficient computations as well as a notion of infeasible ones. The computations of the legitimate users of the scheme ought to be efficient; whereas violating the security features (via an adversary) ought to be infeasible.

Efficient computations are commonly modeled by computations that are polynomial-time in the security parameter. The polynomial bounding the running-time of the legitimate user's strategy is fixed and typically explicit and small. Here (i.e., when referring to the complexity of the legitimate user), we are in the same situation as in any algorithmic setting. Things are different when referring to our assumptions regarding the computational resources of the adversary. A common approach is to postulate that the latter are polynomial-time too, where the polynomial is not *a priori* specified. In other words, the adversary is restricted to the class of efficient computations and anything beyond this is considered *infeasible*. Although many definitions explicitly refer to this convention, this convention is *inessential* to any of the results known in the area. In all cases, a more general statement can be made by referring to adversaries of running-time bounded by any super-polynomial function (or class of functions).

Last, consider the notion of a *negligible probability*. The idea behind this notion is to have a robust notion of rareness: a rare event should occur rarely even if we repeat the experiment a feasible number of times. That is, if we consider any polynomial-time computation to be feasible, then any function $f\colon \mathbb{N} \mapsto \mathbb{N}$ so that $(1 - f(n)^{p(n)} > 0.99$, for any polynomial p, is considered negligible (i.e., f is negligible if for any polynomial p the function $f(\cdot)$ is bounded above by $1/p(\cdot)$). However, if we consider the function $T(n)$ to provide our notion of infeasible computation, then functions bounded above by $1/T(n)$ are considered negligible (in n).

Computational Difficulty

Modern cryptography is concerned with the construction of schemes that are easy to operate (properly) but hard to foil. Thus, a complexity gap (i.e., between the complexity of proper usage and the complexity of defeating the prescribed functionality) lies in the heart of modern cryptography. However, gaps as required for modern cryptography are not known to exist — they are only widely believed to exist. Indeed, almost all of modern cryptography rises or falls with the question of whether **one-way functions** exist. One-way functions are functions that are easy to evaluate but hard (on the average) to invert (cf., [Diffie and Hellman, 1976]). That is, a function $f\colon \{0, 1\}^* \mapsto \{0, 1\}^*$ is called *one-way* if there is an efficient algorithm that, on input x, outputs $f(x)$; whereas any feasible algorithm that tries to find a preimage of $f(x)$ under f may succeed only with negligible probability (where the probability is taken uniformly over the choices of x and the algorithm's coin tosses). For example, multiplication of two (equal length) primes is widely believed to be a one-way function.

Computational Indistinguishability

A central notion in modern cryptography is that of *effective similarity* (cf., [Goldwasser and Micali, 1984] and [Yao, 1982]). The underlying idea is that we do not care if objects are equal or not; all we care is whether a difference between the objects can be observed by a feasible computation. In case the answer is negative, we may say that the two objects are equivalent as far as any practical application is concerned. Indeed, it will be our common practice to interchange such (computationally indistinguishable) objects. Let $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ be probability ensembles such that each X_n and Y_n ranges over strings of length n. We say that X and Y are *computationally indistinguishable* if for every feasible algorithm A, the difference $d_A(n) \overset{\text{def}}{=} |\Pr(A(X_n) = 1) - \Pr(A(Y_n) = 1)|$ is a negligible function in n.

The Simulation Paradigm

A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can be obtained within the same computational effort by a benign behavior (cf., [Goldwasser and Micali, 1984] and [Goldwasser, Micali, and Rackoff, 1989]). The definition of the benign behavior captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. See further discussion in "Encryptions" and "Cryptographic Protocols."

A notable property of the above simulation paradigm, as well as of the entire approach surveyed here, is that this approach is overly liberal with respect to its view of the abilities of the adversary as well as to what might constitute a gain for the adversary. Thus, the approach may be considered overly cautious. However, it seems impossible to come up with definitions of security that distinguish "breaking the scheme in a harmful sense" from "breaking it in a non-harmful sense." What is harmful is application dependent, whereas a good definition of security ought to be application independent (as otherwise, using the scheme in any new application will require a full reevaluation of its security). Furthermore, even with respect to a specific application, it is typically very difficult to classify the set of "harmful breakings."

Pseudorandomness

In practice, "pseudorandom" sequences are used instead of truly random sequences in many applications. The underlying belief is that if an (efficient) application performs well when using a truly random sequence, then it will perform essentially as well when using a pseudorandom sequence. However, this belief is not supported by ad hoc notions of pseudorandomness such as passing the statistical tests in [Knuth, 1969] or having large linear complexity. In contrast, the above belief is an easy corollary of defining pseudorandom distributions as ones that are computationally indistinguishable from uniform distributions.

Pseudorandom Generators

A *pseudorandom generator* with *stretch function* $\ell:\mathbb{N}\mapsto\mathbb{N}$ is an efficient (deterministic) algorithm that on input, a random n-bit *seed* outputs a $\ell(n)$-bit sequence that is computationally indistinguishable from a uniformly chosen $\ell(n)$-bit sequence (cf., [Blum and Micali, 1984] and [Yao, 1982]). Thus, pseudorandom sequences can replace truly random sequences not only in "ordinary" computations, but also in cryptographic ones. That is, *any* cryptographic application that is secure when the legitimate parties use truly random sequences, is also secure when the legitimate parties use pseudorandom sequences.

Various cryptographic applications of pseudorandom generators will be presented in the sequel, but first let us show a construction of pseudorandom generators based on the simpler notion of a one-way function. We start with the notion of a *hard-core* predicate of a (one-way) function: The predicate b is a *hard-core* of the function f if b is easy to evaluate but $b(x)$ is hard to predict from $f(x)$. That is, it is infeasible, given $f(x)$ when x is uniformly chosen, to predict $b(x)$ substantially better than with probability 1/2. It is known that for any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x, r) = (f(x), r)$ [Goldreich and Levin, 1989].

Now, let f be a 1-1 function that is length-preserving and efficiently computable, and b be a hard-core predicate of f. Then, $G(s) = b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))$ is a pseudorandom generator (with stretch function ℓ), where $f^{i+1}(x)\overset{\text{def}}{=}f(f^i(x))$ and $f^0(x)\overset{\text{def}}{=}x$ (cf., [Blum and Micali, 1984] and [Yao, 1982]). As a concrete example, consider the permutation $x\mapsto x^2 \bmod N$, where N is the product of two primes each congruent to 3 (mod 4), and x is a quadratic residue modulo N. Then, we have $G_N(s) = \mathrm{lsb}(s)\cdot\mathrm{lsb}(s^2 \bmod N)\ldots\mathrm{lsb}(s^{2\ell(|s|)-1} \bmod N)$, where $\mathrm{lsb}(x)$ is the least significant bit of x (which is a hard-core of the modular squaring function [Alexi, Chor, Goldreich, and Schnorr, 1988]).

Pseudorandom Functions

Pseudorandom generators allow us to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions are even more powerful: they allow efficient direct access to a huge pseudorandom sequence (which is not feasible to scan bit-by-bit). More precisely, a *pseudorandom function* is an efficient (deterministic) algorithm that, given an n-bit *seed*, s, and an n-bit *argument*, x, returns an n-bit *string*, denoted $f_s(x)$, so that it is infeasible to distinguish the responses of f_s for a uniformly chosen $s \in \{0, 1\}^n$, from the responses of a truly random function F: $\{0, 1\}^n\mapsto\{0, 1\}^n$.

Pseudorandom functions are a very useful cryptographic tool: one can first design a cryptographic scheme assuming that the legitimate users have black-box access to a random function, and next implement the random function using a pseudorandom function. Pseudorandom functions can be constructed using any pseudorandom generator [Goldreich, Goldwasser, and Micali, 1986].

Zero-Knowledge

Loosely speaking, **zero-knowledge proofs** are proofs that yield nothing beyond the validity of the assertion. That is, a verifier obtaining such a proof only gains conviction in the validity of the assertion. Using the simulation paradigm, this requirement is stated by postulating that anything that is feasibly computable from a zero-knowledge proof is also feasibly computable from the valid assertion alone [Goldwasser, Micali, and Rackoff, 1989].

The above informal paragraph refers to proofs as to interactive and randomized processes. That is, here a proof is a (multi-round) protocol for two parties, called verifier and prover, in which the prover wishes to convince the verifier of the validity of a given assertion. Such an *interactive proof* should allow the prover to convince the verifier of the validity of any true assertion, whereas *no* prover strategy can fool the verifier to accept false assertions. Both the above *completeness* and *soundness* conditions should hold with high probability (i.e., a negligible error probability is allowed). The prescribed verifier strategy is required to be efficient. No such requirement is made with respect to the prover strategy; yet we will be interested in "relatively efficient" prover strategies (see below). Zero-knowledge is a property of some prover-strategies. More generally, we consider interactive machines that yield no knowledge while interacting with an arbitrary feasible adversary on a common input taken from a predetermined set (in our case, the set of valid assertions). A strategy A is *zero-knowledge* on inputs from S if, for every feasible strategy B^*, there exists a feasible computation C^* so that the following two probability ensembles are computationally indistinguishable:

1. $\{(A, B^*)(x)\}_{x \in S} \stackrel{\text{def}}{=}$ the output of B^* when interacting with A on common input $x \in S$
2. $\{C^*(x)\}_{x \in S} \stackrel{\text{def}}{=}$ the output of C^* on input $x \in S$

The above definition does not account for auxiliary information that an adversary may have prior to entering the interaction. Accounting for such auxiliary information is essential for using zero-knowledge proofs as subprotocols inside larger protocols. Another concern is that we prefer that the complexity of C^* be bounded as a function of the complexity of B^*. Both concerns are taken care of by a more strict notion called *black-box zero-knowledge.*

Assuming the existence of commitment schemes, there exist (black-box) zero-knowledge proofs for membership in any **NP-set** (i.e., sets having efficiently verifiable proofs of membership) [Goldreich, Micali, and Wigderson, 1991]. Furthermore, the prescribed prover strategy is efficient, provided it is given an NP-witness to the assertion to be proven. This makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols. In a typical cryptographic setting, a user, referred to as A, has a secret and is supposed to take some steps, depending on the secret. The question is how can other users verify that A indeed took the correct steps (as determined by A's secret and the publicly known information). Indeed, if A discloses the secret, then anybody can verify that A took the correct steps. However, A, does not want to reveal the secret. Using zero-knowledge proofs, we can satisfy both conflicting requirements. That is, A can prove in zero-knowledge that he took the correct steps. Note that A's claim to having taken the correct steps is an NP-assertion and that A has an NP-witness to its validity. Thus, by the above result, it is possible for A to efficiently prove the correctness of his actions without yielding anything about the secret.

Encryption

The problem of providing *secret communication over insecure media* is the traditional and most basic problem of cryptography. The setting of this problem consists of two parties communicating through a channel that is possibly tapped by an adversary. The parties wish to exchange information with each other, but keep the "wire-tapper" as ignorant as possible regarding the contents of this information. Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called *encryption*, is applied by the sender (i.e., the party sending a message), while the other algorithm, called *decryption*, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message, and sends the result, called the *ciphertext*, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it, and retrieves the original message (called the *plaintext*).

In order for the above scheme to provide secret communication, the communicating parties (at least the receiver) must know something that is not known to the wire-tapper. (Otherwise, the wire-tapper can decrypt the ciphertext exactly as done by the receiver.) This extra knowledge can take the form of the decryption algorithm itself, or some parameters and/or auxiliary inputs used by the description algorithm. We call this extra knowledge the *decryption key*. Note that, without loss of generality, we may assume that the decryption algorithm is known to the wire-tapper, and that the decryption algorithm operates on two inputs — a ciphertext and a decryption key. We stress that the existence of a secret key, not known to the wire-tapper, is merely a necessary condition for secret communication.

Evaluating the security of an encryption scheme is a very tricky business. A preliminary task is to understand what "security" is (i.e., to properly define what is meant by this intuitive term). Two approaches to defining security are known. The first (*classic*) approach is *information theoretic*. It is concerned with the information about the plaintext that is present in the ciphertext [Shannon, 1949]. Loosely speaking, if the ciphertext contains information about the plaintext then the encryption scheme is considered insecure. It has been shown that such high (i.e., perfect) levels of security can be achieved only if the key in use is at least as long as the total amount of information sent via the encryption scheme. The fact that the key must be longer than the information exchanged using it, is indeed a drastic limitation on the applicability of such encryption schemes.

The second (modern) approach, followed in the current text, is based on computational complexity. This approach is based on the observation that it does not matter whether the ciphertext contains information about the plaintext, but rather whether this information can be efficiently extracted. In other words, instead of asking whether it is *possible* for the wire-tapper to extract specific information, we ask whether it is *feasible* for the wire-tapper to extract this information. It turns out that the new (i.e., computational complexity) approach can offer security even if the key is much shorter than the total length of the messages sent via the encryption scheme.

The computational complexity approach allows the introduction of concepts and primitives that cannot exist under the information theoretic approach. A typical example is the concept of **public-key encryption schemes** [Diffie and Hellman, 1976]. Note that in the above discussion we concentrated on the decryption algorithm and its key. It can be shown that the encryption algorithm must get, in addition to the message, an auxiliary input that depends on the decryption key. This auxiliary input is called the *encryption key*. Traditional encryption schemes, and in particular all the encryption schemes used in the millennia until the 1980s, operate with an encryption key equal to the decryption key. Hence, the wire-tapper in this scheme must be ignorant of the encryption key and, consequently, the *key distribution* problem arises (i.e., how can two parties wishing to communicate over an insecure channel agree on a secret encryption/decryption key?). (The traditional solution is to exchange the key through an alternative channel that is secure, although more expensive to use, for example by a convoy.) The computational complexity approach allows the introduction of encryption schemes in which the encryption key can be given to the wire-tapper without compromising the security of the scheme. Clearly, the decryption key in such schemes is different and, furthermore, infeasible to compute from the encryption key. Such encryption schemes, called *public key*, have the advantage of trivially resolving the key distribution problem since the encryption key can be publicized.

In contrast, traditional encryption schemes in which the encryption key equals the description key are called *private-key* schemes, as in these schemes the encryption key must be kept secret (rather than be public as in public-key encryption schemes). We note that a full specification of either scheme requires the specification of the way keys are generated; that is, a key-generation (randomized) algorithm that, given a security parameter, produces a (random) pair of corresponding encryption/decryption keys (which are identical in case of private-key schemes).

Thus, both private-key and public-key encryption schemes consist of three efficient algorithms: *key generation*, *encryption*, and *decryption*. The difference between the two types is reflected in the definition of security — the security of a public-key encryption scheme should hold also when the adversary is given the encryption key, whereas this is not required for a **private-key encryption scheme.** Below we focus on the public-key case.

Definitions

For simplicity, consider only the encryption of a single message. As implied by the above, a public-key encryption scheme is said to be secure if it is infeasible to gain any information about the plaintext by looking at the ciphertext (and the encryption key). That is, whatever information about the plaintext one may compute from the ciphertext and some *a priori* information, can be essentially computed as efficiently from the *a priori* information along [Goldwasser and Micali, 1984]. This definition (called semantic security) turns out to be equivalent to saying that, for any two messages, it is infeasible to distinguish the encryption of the first message from the encryption of the second message, also when given the encryption key.

It is easy to see that a secure public-key encryption scheme must employ a probabilistic (i.e., randomized) encryption algorithm. Otherwise, given the encryption key as (additional) input, it is easy to distinguish the encryption of the all-zero message from the encryption of the all-ones message. The same holds for *private-key* encryption schemes when considering the security of encrypting several messages (rather than a single message as done above). (Here, for example, using a deterministic encryption algorithm allows the adversary to distinguish two encryptions of the same message from the encryptions of a pair of different messages.) This explains the linkage between the above robust security definitions and the *randomization paradigm* (discussed below).

We stress that the definitions presented above go way beyond saying that it is infeasible to recover the plaintext from the ciphertext. The latter statement is indeed a minimal requirement from a secure encryption scheme, but is far from being a sufficient requirement; typically, encryption schemes are used in applications where even obtaining partial information on the plaintext may endanger the security of the application. When designing an application-independent encryption scheme, we do not know which partial information endangers the application and which does not. Furthermore, even if one wants to design an encryption scheme tailored to a specific application, it is rate (to say the least) that one has a precise characterization of all possible partial information that endangers this application. Thus, we need to require that it is infeasible to obtain any information about the plaintext from the ciphertext. Furthermore, in most applications, the plaintext may not be uniformly distributed and some *a priori* information regarding it is available to the adversary. We require that the secrecy of all partial information is also preserved in such a case. That is, even in presence of *a priori* information on the plaintext, it is infeasible to obtain any (new) information about the plaintext from the ciphertext (beyond what is feasible to obtain from the *a priori* information on the plaintext). The definition of semantic security postulates all of this. The equivalent definition of indistinguishability of encryptions is useful in demonstrating the security of candidate construction, as well as for arguing about their effect as part of larger protocols.

Constructions

It is common practice to use *pseudorandom generators* as a basis for private-key stream ciphers. We stress that this is a very dangerous practice when the pseudorandom generator is easy to predict (such as the linear congruential generator or some modifications of it that output a constant fraction of the bits of each resulting number). However, this common practice becomes sound provided one uses pseudorandom generators (as defined above). An alternative, more flexible construction follows.

Private-Key Encryption Scheme Based on Pseudorandom Functions

The key generation algorithm consists of selecting a seed, denoted s, for such a function, denoted f_s. To encrypt a message $x \in \{0, 1\}^n$ (using key s), the encryption algorithm uniformly selects a string $r \in \{0, 1\}^n$ and produces the ciphertext $(r, x \oplus f_s(r))$, where $\oplus$ denotes the exclusive-or of bit strings. To decrypt the ciphertext (r, y) (using key s), the decryption algorithm just computes $y \oplus f_s(r)$. The proof of security of this encryption scheme consists of two steps (suggested as a general methodology in "Pseudorandomness" section).

1. Prove that an idealized version of the scheme, in which one uses a uniformly selected function F: $\{0, 1\}^n \mapsto \{0, 1\}^n$, rather than the pseudorandom function f_s, is secure.
2. Conclude that the reach scheme (as presented above) is secure (since otherwise one could distinguish a pseudorandom function from a truly random case).

Note that we could have gotten rid of the randomization if we had allowed the encryption algorithm to be history dependent (e.g., use a counter in the role of r). Furthermore, if the encryption scheme is used for FIFO communication between the parties and both can maintain the counter-value, then there is no need for the sender to send the counter-value.

The Randomization Paradigm

We demonstrate this paradigm by presenting a construction of a public-key encryption scheme. We are going to use the RSA scheme [Rivest, Shamir, and Adleman, 1978] as a trapdoor permutation (rather than using it directly as an encryption scheme). The RSA scheme has an instance-generating algorithm that randomly selects two primes, p and q, computes their product $N = p \cdot q$, and selects at random a pair (e, d) so that $e \cdot d \equiv 1 \pmod{\phi(N)}$, where $\phi(N) \overset{\text{def}}{=} (p-1)\cdot(q-1)$. (The "plain RSA" operations are raising to power e or d modulo N.) Following Goldwasser and Micali [1984], the randomized public-key encryption scheme is as follows. The key-generation algorithm is identical to the instance-generator algorithm of RSA, and the encryption key (resp., decryption key) is set to (N, e) (resp., (N, d)), just as in the "plain RSA." To encrypt a single bit σ (using the encryption key (N, e)), the encryption algorithm uniformly selects an element, r, in the set of residues mod N, and produces the ciphertext $(r^e \bmod N, \sigma \oplus \text{lsb}(r))$, where lsb$(r)$ denotes the least significant bit of r. To decrypt the ciphertext (y, τ) (using the decryption key (N, d)), the decryption algorithm just computes $\tau \oplus \text{lsb}(y^d \bmod N)$. The above scheme is quite wasteful in bandwidth; however, the paradigm underlying its construction is valuable in practice. For example, assuming the intractability of factoring large integers, one can derive a secure public-key encryption scheme with efficiency comparable to that of RSA (cf., [Blum and Goldwasser, 1984]). Recall that RSA itself is not semantically secure (as it employs a deterministic encryption algorithm).

Beyond Eavesdropping Security

The above definitions refer only to a "passive" attack in which the adversary merely eavesdrops the line over which ciphertexts are being sent. Stronger types of attacks, culminating in the so-called Chosen Ciphertext Attach, may be possible in various applications. In their stronger form, such attacks are also related to the notion of *non-malleability* of the encryption scheme, where one requires that it should be infeasible for an adversary given a ciphertext to produce a valid ciphertext for a related plaintext [Dolev, Dwork, and Naor, 1991]. Encryption schemes secure against such attacks can be constructed under various assumptions (e.g., intractability of factoring).

Signatures

The need to discuss *digital signatures* has arisen with the introduction of computer communication in the business environment (in which parties need to commit themselves to proposals and/or declarations they make) (cf., [Diffie and Hellman, 1976] and [Rabin, 1977]). Loosely speaking, a *scheme for unforgeable signatures* requires that:

- Each user can *efficiently produce his own signature* on documents of his choice.
- Every user can *efficiently verify* whether a given string is a signature of another (specific) user on a specific document.
- But, *it is infeasible to produce signatures of other users* to documents they did not sign.

Note that the formulation of unforgeable digital signatures also provides a clear statement of the essential ingredients of handwritten signatures. The ingredients are each person's ability to sign for himself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least hard) to forge signatures in a manner that can pass the verification procedure. It is not clear to what extent handwritten signatures meet these requirements. In contrast, our discussion of digital signatures provides precise statements concerning the extent to which digital signatures meet the above requirements. Furthermore, unforgeable digital **signature schemes** can be constructed based on some reasonable computational assumptions (i.e., existence of one-way functions).

Definitions

A signature scheme consists of three algorithms corresponding to the key generation, signing, and verification tasks. As in the case of encryption, the signing key is the (secret) information that distinguishes the legitimate signer from all other users. Analogous to the case of public-key encryption, other users only have the corresponding verification key allowing them to verify signatures (but not to produce them). Following Goldwasser, Micali, and Rivest [1988], we consider very powerful attacks on the signature scheme as well as a very liberal notion of breaking it. Specifically, the attacker is allowed to obtain signatures to any message of his choice. One may argue that in many applications, such a general attack is not possible (as messages to be signed must have a specific format). Yet, our view is that it is impossible to define a general (i.e., application-independent) notion of admissible messages, and thus a general/robust definition of an attack seems to have to be formulated, as suggested here. (Note that, at worst, our approach is overly cautious.) Likewise, the adversary is said to be successful if he can produce a valid signature to any message for which he has not asked for a signature during his attack. Again, this refers to the ability to form signatures to possibly "nonsensical" messages as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" messages (so that only forging signatures to them will be consider a breaking of the scheme).

A *chosen message attack* is a process that, on input, a verification key can obtain signatures (relative to the corresponding signing key) to messages of its choice. Such an attack is said to *succeed* (in existential forgery) if it outputs a valid signature to a message for which it has *not* requested a signature during the attack. A signature scheme is *secure* (or unforgeable) if every *feasible* chosen message attack succeeds with, at most, negligible probability.

We stress that *plain* RSA (similar to plain versions of Rabin's [1979] scheme and DSS) is not secure under the above definition. However, it may be secure if the message is "randomized" before RSA (or the other schemes) is applied. Thus, the randomization paradigm seems pivotal here too.

Message Authentication Schemes

Message authentication is a task related to the setting considered for encryption schemes; that is, communication over an insecure channel. This time, we consider an active adversary that is monitoring the channel and may alter the messages sent on it. The parties communicating through this insecure channel wish to authenticate the messages they send so that their counterpart can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a *scheme for message authentication* requires that:

- Each of the communicating parties can *efficiently produce an authentication tag* to any message of his choice.
- Each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message.
- But, *it is infeasible for an external adversary* (i.e., a party other than the communicating parties) to *produce authentication tags* to messages not sent by the communicating parties.

Note that in contrast to the specification of signature schemes, we do not require universal verification. That is, only the receiver is required to be able to verify the authentication tags; and the fact that the receiver can also produce such tags is of no concern. Thus, schemes for message authentication can be viewed as a private-key version of signature schemes. The difference between the two is that, in the setting of message authentication, the ability to verify tags can be linked to the ability to authenticate messages; whereas in the setting of signature schemes, these abilities are separated (i.e., everybody can verify signatures but only the holder of the signing key can produce valid signatures). Hence, digital signatures provide a solution to the message authentication problem, but message authentication schemes do not necessarily constitute digital signature schemes.

Constructions

Message authentication schemes can be constructed using pseudorandom functions. However, as noted by Bellare, Canetti, and Krawczyk [1996], *extensive* usage of pseudorandom functions seems an overkill for achieving message authentication, and more efficient schemes can be obtained based on other cryptographic primitives.

Three central paradigms in the construction of *signature schemes* are the "refreshing" of the "effective" signing-key, the usage of an "authentication tree," and the "hashing paradigm." The first paradigm is aimed at limiting the potential dangers of a chosen message attack by signing the given message using a newly generated (random) instance of the signature scheme, and authenticating this random instance relative to the fixed public key. A natural way of carrying on the authentication of the many newly generated keys is by using an authentication tree [Merkle, 1980]. Finally, the hashing paradigm refers to the common practice of signing real documents via a two-stage process: first, the document is hashed into a (relatively) short bit string, and then the basic signature scheme is applied to the resulting string. This practice, as well as other usages of the paradigm, is sound provided the hashing function belongs to a family of *Universal One-Way Hash Functions* (cf., [Naor and Yung, 1989]).

Cryptographic Protocols

A general framework for casting *n*-party cryptographic problems consists of specifying a random process that maps *n* inputs to *n* outputs. The inputs to the process are to be thought of as local inputs of *n* parties, and the *n* outputs are their corresponding local outputs. The random process describes the desired functionality. That is, if the *n* parties were to trust each other (or trust some outside party), then they could each send their local input to the trusted party, who would compute the outcome of the process and send each party the corresponding output. The question addressed in this section is: to what extent can such a trusted party be "emulated" by the mutually distrustful parties themselves? We consider two general settings: two-party protocols and multiparty protocols in which the majority of the players are honest. We start with the latter case.

Multi-Party Protocols with Honest Majority

Consider any multi-party protocol. We first observe that each party can change its local input before even entering the execution of the protocol. However, this is also unavoidable when the parties utilize a trusted party. In general, the basic paradigm underlying the definitions of *secure multi-party computations* amounts to saying that situations that can occur in the real protocol, can be simulated in an ideal model (where the parties can employ a trusted party). Thus, the effective malfunctioning of parties in secure protocols is restricted to what is postulated in the corresponding ideal model.

Here, we consider an ideal model in which any minority group (of the parties) may collude as follows. First, this minority shares its original inputs and decides together on replaced inputs to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.) When the trusted party returns the output, each majority player outputs it locally, whereas the colluding minority can compute outputs based on all they know. A *secure multi-party computation with honest majority* is required to simulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the players in the actual protocol, can be essentially simulated by a (different) feasible adversary that controls the corresponding players in the ideal model. This means that in a secure protocol, the effect of each minority group is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority players) before the protocol starts, and replacing its own local outputs (depending only on its local inputs and outputs) after the protocol terminates. (We stress that in the real execution, the minority players do obtain additional pieces of information; yet in a secure protocol, they gain nothing from these additional pieces of information, as they can actually reproduce those by themselves.)

It turns out that efficient and secure multi-party computation of any functionality is possible, provided that a majority of the parties are honest and that trapdoor permutations do exist (e.g., factoring is intractable) [Goldreich, Micali, and Wigderson, 1987].

Two-Party Protocols

Secure multi-party protocols with honest majority can even tolerate a situation in which a minority of the parties abort the execution. This cannot be expected to happen when there is no honest majority (e.g., in a two-party computation). In light of this fact, we consider an ideal model in which each of the two parties can "shut down" the trusted (third) party at any point in time. In particular, this can happen after the trusted party has supplied the outcome of the computation to one party, but before it has supplied it to the second. A *secure two-party computation allowing abort* is required to simulate this ideal model. That is, each party's "effective malfunctioning" in a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. We stress that, as above, the choice of the initial input of each party may *not* depend on the input of the other party. Again, it turns out that efficient and secure two-party computation (allowing abort) of any functionality is possible, provided that trapdoor permutations do exist [Yao, 1986].

Defining Terms

Message authentication scheme: A scheme to generate keys so that signing messages and authenticating the validity of signatures can be executed efficiently when knowing the key; but without the key, it is infeasible to falsely authenticate a new message even when given authentication tags to messages of one's choice.

NP-sets: The set S is in NP if there exists a polynomial-time recognizable binary relation R_S and a polynomial ℓ so that $x \in S$ if and only if there exists y so that $|y| \leq \ell(|x|)$ and $(x, y) \in R_S$. Such a y is called an *NP-witness* to $x \in S$.

One-way functions: Functions that are easy to evaluate but difficult (on average) to invert.

Private-key encryption scheme: A scheme to generate keys so that encryption and decryption of messages can be executed efficiently when knowing the key, but it is infeasible to gain any information of the message given only the encrypted message (but not the key).

Pseudorandom functions: Efficient deterministic programs that, given a random seed, present an input-output behavior that is indistinguishable from the one of a truly random function.

Pseudorandom generators: Efficient deterministic programs that stretch short random seeds into longer sequences that are computationally indistinguishable from truly random sequences.

Public-key encryption scheme: A scheme to generate pairs of encryption-decryption keys to that encryption and decryption of messages can be executed efficiently when knowing the corresponding key, but it is infeasible to gain any information of the message given only the encrypted message and the encryption key (but not the decryption key).

Signature scheme: A scheme to generate pairs of signing-verifying keys so that signing messages and verifying the validity of signatures can be executed efficiently when knowing the corresponding key; but without the signing key, it is infeasible to forge a signature to a new message even when given signatures to messages of one's choice.

Zero-knowledge proofs: Two-party protocols by which one party can convince the other party of the validity to an assertion without yielding anything else. That is, when executing such a protocol, the verifier is essentially in the same situation as he would have been if he was granted by a trusted party that the assertion is valid.

References

W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA/Rabin functions: certain parts are as hard as the whole, *SIAM Journal on Computing*, 17, 194–209, 1988.

M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication, in *Crypto96*, Springer Lecture Notes in Computer Science, 1109, 1–15, 1996.

M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information, in *Crypto84*, Lecture Notes in Computer Science, 196, 289–302, 1984.

M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits, *SIAM Journal on Computing*, 13, 850–864, 1984. Preliminary version in *23rd IEEE Symposium on Foundations of Computer Science*, 1982.

W. Diffie and M.E. Hellman. New directions in cryptography, *IEEE Trans. on Info. Theory*, IT-22, 644–654, 1976.

D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography, in *23rd ACM Symposium on the Theory of Computing*, 542–552, 1991.

O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions, *Journal of the ACM*, 33, 792—807, 1986.

O. Goldreich and L.A. Levin. Hard-core predicates for any one-way function, in *21st ACM Symposium on the Theory of Computing*, 25–32, 1989.

O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, *Journal of the ACM*, 38, 691–729, 1991. Preliminary version in *27th IEEE Symposium on Foundations of Computer Science*, 1986.

O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — A completeness theorem for protocols with honest majority, in *19th ACM Symposium on the Theory of Computing*, 218–229, 1987.

S. Goldwasser and S. Micali. Probabilistic encryption, *Journal of Computer and System Science*, 28, 270–299, 1984. Preliminary version in *14th ACM Symposium on the Theory of Computing*, 1982.

S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems, *SIAM Journal on Computing*, 18, 186–208, 1989. Preliminary version in *17th ACM Symposium on the Theory of Computing*, 1985.

S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks, *SIAM Journal on Computing*, 281–308, 1988.

D.E. Knuth. *The Art of Computer Programming, Vol. 2 (Seminumerical Algorithms)*. Addison-Wesley Publishing Company, 1969 (first edition) and 1981 (second edition).

R.C. Merkle. Protocols for public key cryptosystems, in *Proc. of the 1980 Symposium on Security and Privacy*, 1980.

M. Naor and M. Yung. Universal one-way hash functions and their cryptographic application, *21st ACM Symposium on the Theory of Computing*, 33–43, 1989.

M.O. Rabin. Digitalized signatures, in *Foundations of Secure Computation* (R.A. DeMillo et al., eds.), Academic Press, 1977.

M.O. Rabin. Digitalized signatures and public key functions as intractable as factoring, MIT/LCS/TR-212, 1979.

R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems, *CACM*, 21, 120–126, 1978.

C.E. Shannon. Communication theory of secrecy systems, *Bell Sys. Tech. J.*, 28, 656–715, 1949.

A.C. Yao. Theory and application of trapdoor functions, in *23rd IEEE Symposium on Foundations of Computer Science*, 80–91, 1982.

A.C. Yao. How to generate and exchange secrets, in *27th IEEE Symposium on Foundations of Computer Science*, 162–167, 1986.

Further Information

O. Goldreich. *Foundation of Cryptography — Fragments of a Book*. February 1995. Available from http://theory.lcs.mit.edu/~oded/frag.html.

O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms and Combinatorics series (Vol. 17), Springer, 1998.

O. Goldreich. *Secure Multi-Party Computation*. June 1998. Available from http://theory.lcs.mit.edu.

A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1996.