The magic words are Squeamish Ossifrage.	I5&AE<&UA9VEC'=0 <F1s"F%R92!3<75E96UI<V V@*3W-S:69R86=E+@K_

INPUT OUTPUT

18.6 Cryptography

Input description: A plaintext message T or encrypted text E, and a key k.

Problem description: Encode T (decode E) using k giving E (T).

Discussion: Cryptography has grown substantially in importance as computer networks make confidential documents more vulnerable to prying eyes. Cryptography increases security by making messages difficult to read even if they fall into the wrong hands. Although the discipline of cryptography is at least two thousand years old, its algorithmic and mathematical foundations have only recently solidified to the point where provably secure cryptosystems can be envisioned.

Cryptographic ideas and applications go beyond the commonly known concepts of "encryption" and "authentication." The field now includes such important mathematical constructs such as cryptographic hashes, digital signatures, and useful primitive protocols that provide associated security assurances.

There are three classes of cryptosystems everyone should be aware of:

- *Caesar shifts* – The oldest ciphers involve mapping each character of the alphabet to a different letter. The weakest such ciphers rotate the alphabet by some fixed number of characters (often 13), and thus have only 26 possible keys. Better is to use an arbitrary permutation of the letters, giving 26! possible keys. Even so, such systems can be easily attacked by counting the frequency of each symbol and exploiting the fact that "e" occurs more often than "z". While there are variants that will make this more difficult to break, none will be as secure as AES or RSA.

- *Block Shuffle Ciphers* – This class of algorithms repeatedly shuffle the bits of your text as governed by the key. The classic example of such a cipher is the *Data Encryption Standard* (DES). Although approved as a Federal Information Processing Standard in 1976, its 56-bit key length is now considered too short for applications requiring substantial levels of security. Indeed, a special purpose machine named "Deep Crack" demonstrated that it is possible to decrypt messages without a key in less than a day. As of May 19,

2005, *DES* has been officially withdrawn as a federal standard, replaced by the stronger *Advanced Encryption Standard* (AES).

However, a simple variant called *triple DES* permits an effective key length of 112 bits by using three rounds of DES with two 56-bit keys. In particular, first encrypt with key1, then *decrypt* with key2, before finally encrypting with key1. There is a mathematical reason for using three rounds instead of two; the encrypt-decrypt-encrypt pattern is used so that the scheme is equivalent to single DES when $key1 = key2$. This is enough to keep "Deep Crack" at bay. Indeed, *triple DES* has recently been approved by the National Institute of Standards and Technology (NIST) for sensitive government information through the year 2030.

- *Public Key Cryptography* – If you fear bad guys reading your messages, you should be afraid to tell anyone else the key needed to decrypt them. Public-key systems use different keys to encode and decode messages. Since the encoding key is of no help in decoding, it can be made public at no risk to security. This solution to the key distribution problem is literally its key to success.

  RSA is the classic example of a public key cryptosystem, named after its inventors Rivest, Shamir, and Adelman. The security of RSA is based on the relative computational complexities of factoring and primality testing (see Section 13.8 (page 420)). Encoding is (relatively) fast because it relies on primality testing to construct the key, while the hardness of decryption follows from that of factoring. Still, RSA is slow relative to other cryptosystems—roughly 100 to 1,000 times slower than DES.

The critical issue in selecting a cryptosystem is identifying your paranoia level—i.e. , deciding how much security you need. Who are you trying to stop from reading your stuff: your grandmother, local thieves, the Mafia, or the NSA? If you can use an accepted implementation of AES or RSA, you should feel pretty safe against anybody, at least for now. Increasing computer power often lays waste to cryptosystems surprisingly quickly; recall that DES lived less than 30 years as a strong system. Be sure to use the longest possible keys and keep abreast of algorithmic development if you are a planning long-term storage of criminal material.

That said, I will confess that I use DES to encrypt my final exam each semester. It proved more than sufficient the time an ambitious student broke into my office looking for it. The story would have been different had the NSA had been breaking in, but it is important to understand that *the most serious security holes are human, not algorithmic*. Ensuring that your password is long enough, hard to guess, and not written down is far more important than obsessing about the encryption algorithm.

Most symmetric key encryption mechanisms are harder to crack than public key ones for the same key size. This means one can get away with much shorter key lengths for symmetric key than for public key encryption. NIST and RSA Labs

both provide schedules of recommended key sizes for secure encryption, and as of this writing they recommend 80-bit symmetric keys as equivalent to 1024-bit asymetric keys. This difference helps explain why symmetric key algorithms are typically orders of magnitude faster than public key algorithms.

Simple ciphers like the Caesar shift are fun and easy to program. For this reason, it is healthy to use them for applications needing only a casual level of security (such as hiding the punchlines of jokes). Since they are easy to break, they should never be used for serious security applications.

Another thing you should *never* do is try to develop your own novel cryptosystem. The security of triple DES and RSA is accepted because these systems have survived many years of public scrutiny. In this time, many other cryptosystems have been proposed, proven vulnerable to attack, and then abandoned. This is not a field for amateurs. If you are charged with implementing a cryptosystem, carefully study a respected program such as PGP to see how they handle issues such as key selection and key distribution. Any cryptosystem is as strong as its weakest link.

Certain other problems related to cryptography arise often in practice:

- *How can I validate the integrity of data against random corruption?* – There is often a need to validate that transmitted data is identical to that which has been received. One solution is for the receiver to transmit the data back to the source and have the original sender confirm that the two texts are identical. This fails when the exact inverse of an error is made in the retransmission, but a more serious problem is that your available bandwidth is cut in half with such a scheme.

  A more efficient method uses a *checksum*, a simple mathematical function that hashes a long text down to a simple number or digit. We then transmit the checksum along with the text. The checksum can be recomputed on the receiving end and bells set off if the computed checksum is not identical to what was received. The simplest checksum scheme just adds up the byte or character values and takes the sum modulo of some constant, say $2^8 = 256$. Unfortunately, an error transposing two or more characters would go undetected under such a scheme, since addition is commutative.

  Cyclic-redundancy check (CRC) provides a more powerful method for computing checksums that is used in most communications systems and internally in computers to validate disk drive transfers. These codes compute the remainder in the ratio of two polynomials, the numerator of which is a function of the input text. The design of these polynomials involves considerable mathematical sophistication, but ensures that all reasonable errors are detected. The details of efficient computation are sufficiently complicated that we recommend that you start from an existing implementation, described below.

- *How can I validate the integrity of data against deliberate corruption?* – CRC is good at detecting random errors, but not malicious changes to a document. *Cryptographic hashing functions* such as MD5 and SHA-256 are (in principle) easy to compute for a document but hard to invert. This means that given a particular hash code value x, it is difficult to construct a document d such that $H(d) = x$. The property makes them valuable for digital signatures and other applications.

- *How can I prove that a file has not been changed?* – If I send you a contract in electronic form, what is to stop you from editing the file and then claiming that your version was what we had really agreed to? I need a way to prove that any modification to a document is fraudulent. *Digital signatures* are a cryptographic way for me to stamp my document as genuine.

  Given a file, I can compute a checksum for it, and then encrypt this checksum using my own private key. I send you the file and the encrypted checksum. You can now edit the file, but to fool the judge you must also edit the encrypted checksum such that it can be decrypted to yield the correct checksum. With a suitably good checksum function, designing a file that yields the same checksum becomes an insurmountable problem. For full security, we need a trusted third party to authenticate the timestamp and associate the private key with me.

- *How can I restrict access to copyrighted material?* – An important emerging application for cryptography is digital rights management for audio and video. A key issue here is speed of decryption, as it must keep up with data transmission or retrieval in real time. Such *stream ciphers* usually involve efficiently generating a stream of pseudorandom bits, say using a shift-register generator. The exclusive-or of these bits with the data stream gives the encrypted sequence. The original data is recovered by exclusive-oring the result with the same stream of pseudorandom bits.

  High-speed cryptosystems have proven to be relatively easy to break. The state-of-the-art solution to this problem involves erecting laws like the Digital Millennium Copyright Act to make it illegal to try to break them.

Implementations: *Nettle* is a comprehensive low-level cryptographic library in C. Cryptographic hash functions include MD5 and SHA-256. Block ciphers include DES, AES, and some more recently developed codes. An implementation of RSA is also provided. *Nettle* is available at *http://www.lysator.liu.se/∼nisse/nettle*.

A comprehensive overview of cryptographic algorithms with assessments of strength is available at *http://www.cryptolounge.org/wiki/Category:Algorithm*. See *http://csrc.nist.gov/groups/ST/toolkit* for related cryptographic resources provided by NIST.

Crypto++ is a large C++ class library of cryptographic schemes, including all we have mentioned in this section. It is available at *http://www.cryptopp.com/*.

Many popular open source utilities employ serious cryptography, and serve as good models of current practice. *GnuPG*, an open source version of PGP, is available at *http://www.gnupg.org/*. *OpenSSL*, for authenticating access to computer systems, is available at *http://www.openssl.org/*.

The *Boost CRC Library* provides multiple implementations of cyclic redundancy check algorithms. It is available at *http://www.boost.org/libs/crc/*.

Notes: The *Handbook of Applied Cryptography* [MOV96] provides technical surveys of all aspects of cryptography, and has been generously made available online at *http://www.cacr.math.uwaterloo.ca/hac/*. Schneier [Sch96] provides a thorough overview of different cryptographic algorithms, with [FS03] as perhaps a better introduction. Kahn [Kah67] presents the fascinating history of cryptography from ancient times to 1967 and is particularly noteworthy in light of the secretive nature of the subject.

Expositions on the RSA algorithm [RSA78] include [CLRS01]. The RSA Laboratories home page *http://www.rsa.com/rsalabs/* is very informative.

Of course, the NSA (National Security Agency) is the place to go to learn the real state of the art in cryptography. The history of DES is well presented in [Sch96]. Particularly controversial was the decision by the NSA to limit key length to 56 bits.

MD5 [Riv92] is the hashing function used by PGP to compute digital signatures. Expositions include [Sch96, Sta06]. Serious problems with the security of MD5 have recently been exposed [WY05]. The SHA family of hash functions appears more secure, particularly SHA-256 and SHA-512.

Related Problems: Factoring and primality testing (see page 420), text compression (see page 637)).