Answers To

Frequently Asked Questions About Today's Cryptography

Paul Fahn
RSA Laboratories
100 Marine Parkway
Redwood City, CA 94065

Version 1.0, draft 1e.
Last update: September 14, 1992.
Copyright © 1992 RSA Laboratories, a division of RSA Data Security Inc.
All rights reserved.
Part #999-100002-100-000-000

Contents

1 General **1**

1.1 *What is encryption?* 1
1.2 *What is authentication?* 1
1.3 *What is public-key cryptography?* 2
1.4 *What are the advantages and disadvantages of public-key cryptography over secret-key cryptography?* 3
1.5 *Is cryptography patentable in the U.S.?* 3
1.6 *Is cryptography exportable from the U.S.?* 4

2 RSA **5**

2.1 *What is RSA?* 5
2.2 *Why use RSA rather than DES?* 6
2.3 *Are there hardware implementations of RSA?* 6
2.4 *How fast is RSA?* 6
2.5 *How much extra message length is caused by using RSA?* 7
2.6 *What would it take to break RSA?* 7
2.7 *Are strong primes necessary in RSA?* 8
2.8 *How large a modulus should be used in RSA?* 9
2.9 *How large should the primes be?* 10
2.10 *How does one find random numbers for keys?* 10
2.11 *What if users of RSA run out of distinct primes?* 11
2.12 *How do you know if a number is prime?* 11
2.13 *How is RSA used for encryption in practice?* 11
2.14 *How is RSA used for authentication in practice?* 11
2.15 *Does RSA help detect transmission errors?* 12
2.16 *Does RSA help protect against computer viruses?* 12
2.17 *What are alternatives to RSA?* 13
2.18 *Is RSA currently in use today?* 14
2.19 *Is RSA an official standard today?* 15
2.20 *Is RSA a de facto standard today?* 15
2.21 *What will happen to RSA if NIST adopts another method to be the official standard?* 16
2.22 *Is RSA patented?* 16
2.23 *Can RSA be exported from the U.S.?* 17

3 Key Management **17**

3.1 *What key management issues are involved in public-key cryptography?* 17
3.2 *Who needs a key?* 18
3.3 *How does one get an RSA key pair?* 18
3.4 *Should a public key or private key be shared among users?* 18
3.5 *What are certificates?* 19

3.6 *How are certificates used?* 19
3.7 *Who issues certificates and how?* 20
3.8 *What is a CSU, or, How do certifying authorities store their private keys?* 21
3.9 *Are certifying authorities susceptible to attack?* 21
3.10 *What if the certifying authority's key is lost or compromised?* . 23
3.11 *For how long is a key pair valid?* 23
3.12 *What happens if I lose my private key?* 24
3.13 *What happens if my private key is compromised?* 24
3.14 *What are Certificate Revocation Lists (CRLs)?* 25
3.15 *How should I store my private key?* 25
3.16 *How does one find someone else's public key?* 26
3.17 *How can signatures remain valid beyond the expiration dates of their keys, or, How do you verify a 20-year-old signature?* 26
3.18 *What is a digital timestamping service?* 27
3.19 *What other digital services will accompany widespread use of RSA?* 28

4 Factoring and Discrete Log **29**
4.1 *What is a one-way function?* 29
4.2 *What is the significance of one-way functions for cryptography?* . 29
4.3 *What is the factoring problem?* 30
4.4 *What is the significance of factoring in cryptography?* 30
4.5 *Has factoring been getting easier?* 31
4.6 *What are the best factoring methods in use today?* 31
4.7 *What are the prospects for theoretical factoring breakthroughs?* . . 32
4.8 *What is the RSA Factoring Challenge?* 33
4.9 *What is the discrete log problem?* 33
4.10 *Which is easier, factoring or discrete log?* 33

5 DES **34**
5.1 *What is DES?* 34
5.2 *Has DES been broken?* 34
5.3 *How does one use DES securely?* 35
5.4 *Can DES be exported from the U.S.?* 36
5.5 *What are the alternatives to DES?* 36
5.6 *Is DES a group?* 36

6 Miscellaneous **37**
6.1 *What is the legal status of documents signed with digital signatures?* 37
6.2 *What is a hash function? What is a message digest?* 38
6.3 *What are MD2, MD4 and MD5?* 38
6.4 *What is SHS?* 39
6.5 *What are RC2 and RC4?* 40
6.6 *What is PEM?* 40

6.7 *What is PKCS?* 41
6.8 *What is RSAREF?* 41

7 DSS **42**
7.1 *What is DSS?* 42
7.2 *Is DSS secure?* 42
7.3 *Is use of DSS covered by any patents?* 43
7.4 *What is the current status of DSS?* 44

8 NIST and NSA **44**
8.1 *What is NIST?* 44
8.2 *What role does NIST play in cryptography?* 45
8.3 *What are NIST's plans for the future of cryptography?* 45
8.4 *What is the NSA?* 45
8.5 *What role does NSA play in commercial cryptography?* 46

1 General

1.1 *What is encryption?*

Encryption is the transformation of data into a form unreadable by anyone without a secret decryption key. Its purpose is to ensure privacy by keeping the content of the information hidden from anyone for whom it is not intended, even those with access to the (encrypted) data.

In a multi-user setting, encryption allows secure communication over an insecure channel. The general scenario is as follows: Alice wishes to send a message to Bob so that no one else besides Bob can read it. Alice encrypts the message, which is called the plaintext, with an encryption key; the encrypted message is called the ciphertext. Bob decrypts the ciphertext with the decryption key and reads the message. An attacker, Charlie, may either try to obtain the secret key or to recover the plaintext without using the secret key. In a secure cryptosystem, the plaintext cannot be recovered from the ciphertext except by using the decryption key.

Encryption can also be used in a single-user setting, say by encrypting files on a hard disk to prevent an intruder from reading them. This is one example of bulk encryption, which refers to encryption of large amounts of data.

Cryptography has been around for millennia; see [33] for a good history of cryptography; see [61] and [10] for an introduction to modern cryptography.

1.2 *What is authentication?*

Authentication in a digital setting is a process whereby the receiver of a digital message can be confident of the identity of the sender and/or the integrity of the message. Authentication protocols can be based on either conventional secret-key cryptosystems like DES (MIT's Kerberos system is an example) or on public-key systems like RSA; authentication in public-key systems uses digital signatures.

In this document, authentication will generally refer to the use of digital signatures, which play a function for digital documents similar to that played by handwritten signatures for printed documents: the signature is an unforgeable message asserting that a named person (or other entity) either wrote or otherwise agreed to the document to which the signature is attached. The recipient, as well as a third party, can verify both that the document did indeed originate from the person whose signature is attached and that the document has not been altered since it was signed. A secure authentication system thus consists of two parts: a method of signing a document such that forgery is infeasible, and a method of verifying that a signature was actually generated by whomever it represents. Furthermore, secure digital signatures cannot be repudiated; i.e., the signer of a document cannot later disown it by claiming it was forged.

Unlike encryption, digital signatures are a recent development, the need for which has arisen with the proliferation of digital communications.

1.3 *What is public-key cryptography?*

Traditional cryptography is based on the sender and receiver of a message knowing and using the same secret key; the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as secret-key cryptography, or symmetric cryptography. The main problem is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, or a phone system, or some other transmission system to not disclose the secret key they are communicating. Anyone who overhears or intercepts the key in transit can later read all messages encrypted using that key. The generation, transmission and storage of keys is called key management; all cryptosystems must deal with key management issues. Secret-key cryptography often has difficulty providing secure key management.

Public-key cryptography was invented in 1976 by Whitfield Diffie and Martin Hellman [24] in order to solve the key management problem. In the new system, every person gets a pair of keys, one public, one private. Everyone publishes his public key and keeps his private key secret. The need for sender and receiver to share secret information is eliminated: all communications involve only public keys, and no private key is ever transmitted or shared. No longer is it necessary to trust the security of some communications channel. Anyone can send a confidential message using public information only; it can only be decrypted with a secret key which is in the sole possession of the intended recipient.

Furthermore, it was realized that public-key cryptography, unlike secret-key cryptography, can be used for authentication (digital signatures) as well as for privacy (encryption).

Here's how it works for encryption: If Alice wishes to send a message to Bob, she looks up Bob's public key in a directory, uses it to encrypt to message and sends it off. Bob then uses his private key to decrypt the message and read it. No one listening in can decrypt the message. Anyone can send an encrypted message to Bob but only Bob can read it. Clearly, one requirement is that no one can figure out a private key from the corresponding public key.

Here's how it works for authentication: Alice, to sign a message, does a computation involving both her secret key and the message itself; the output is the signature and is attached to the message which is then sent. Bob, to verify the signature, does some computation involving the message, the purported signature, and Alice's public key. If the results properly hold in a simple mathematical relation, the signature is verified as genuine; otherwise, the signature may be fraudulent or the message altered, and they are discarded.

A good history of public key cryptography, by one of its inventors, is given by Diffie [22].

1.4 *What are the advantages and disadvantages of public-key cryptography over secret-key cryptography?*

The primary advantage of public-key cryptography is increased security: the secret keys do not need to be transmitted or communicated to anyone; no one else needs to be trusted. In a secret-key system, there is always a chance that an enemy could discover the secret key while it is being transmitted.

Another major advantage of public-key systems is that they can provide a method for digital signatures. Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well; a sender can repudiate a previously signed message by claiming that the shared secret was somehow compromised. Public-key authentication provides non-repudiation and digitally signed messages can be proved authentic to a third party, such as a judge. This allows public-key authentication to be used for legally binding documents; secret-key authentication cannot be so used.

The primary disadvantage of public-key cryptography is speed: there are popular secret-key encryption methods which are significantly faster than any currently available public-key method. The issue of speed in the case of RSA public-key encryption is addressed in Question 2.4.

For encryption, the best solution is to combine public- and secret-key systems to get both the security advantages of public-key systems and the speed advantages of secret-key systems. The public-key system can be used only to encrypt a secret key which is then used to encrypt the bulk of a file or message. This is explained in more detail in Question 2.13 in the case of RSA. Public-key cryptography is not meant to replace secret-key cryptography, but rather to supplement it to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise secret-key system [24]; this is still one of its primary functions.

Secret-key cryptography remains extremely important and is the subject of much ongoing study and research. Some secret-key systems are discussed in Questions 5.1 and 5.5.

1.5 *Is cryptography patentable in the U.S.?*

Cryptographic systems are patentable. Many secret-key cryptosystems have been patented, including DES; also, NIST has applied for a patent for its recently proposed digital signature standard. Some patent applications for cryptosystems have been blocked by intervention by the National Security Agency (NSA) or other intelligence or defense agencies, under the authority of the Invention Secrecy Act of 1940 and the National Security Act of 1947; see Landau [41] for some recent cases related to cryptography.

The basic ideas of public-key cryptography are contained in US Patent 4,200,770, by M.Hellman, W.Diffie, and R. Merkle, issued 4/29/80 and in US Patent 4,218,582, by M.Hellman and R.Merkle, issued 8/19/80. The exclusive

licensing rights to both patents are held by Public Key Partners (PKP), of Sunnyvale, California, which also holds the RSA patent (see Question 2.22). Usually all of these patents are licensed together. The authors of patent 4,218,582 claim that it applies to all uses of public-key cryptography. It has been patented throughout the world.

All legal challenges to public-key patents have been settled before judgment. In a recent case, for example, PKP brought suit against the TRW Corporation which was using public-key cryptography (the ElGamal system) without a license; TRW claimed it did not need to license. In June 1992 a settlement was reached in which TRW agreed to license to the patents.

1.6 *Is cryptography exportable from the U.S.?*

All cryptographic products need export licenses from the State Department, acting under authority of the International Traffic in Arms Regulation (ITAR), which defines cryptographic devices (including software) as munitions. The U.S. government has historically been reluctant to grant export licenses for encryption products stronger than some level (usually not defined publicly); it does grant licenses for encryption products that are less strong and for authentication products, no matter how strong.

The National Security Agency (NSA) has de facto control over export of cryptographic products. A vendor seeking to export first must submit the product to the NSA for approval, then submit the export license application to the State Department. Upon approval by the State Department, the export will go under the jurisdiction of the Commerce Department, which has never put serious obstacles in the way of exporting cryptography. However, the State Department will not grant a license without NSA approval and routinely grants licenses whenever NSA does approve. Therefore, the decisions over exporting cryptography ultimately rest with the NSA.

It is the stated policy of the NSA not to restrict export of cryptography for authentication; it is only concerned with the use of cryptography for privacy. A vendor seeking to export a product for authentication only will be granted an export license as long as it can demonstrate that the product can not be easily adapted for use in encryption; this is true even for very strong systems, such as RSA with large key sizes. Furthermore, the bureaucratic procedures are simpler for authentication products than privacy products. An authentication product needs NSA and State Dept. approval only once, whereas an encryption product may need approval for every sale or every product revision.

Export policy is currently a matter of great controversy. The Software Publishers Association (SPA), a software industry group, has recently been negotiating with the government in order to get export license restrictions eased; one agreement was reached that allows simplified procedures for export of two bulk encryption ciphers, RC2 and RC4 (see Question 6.5), when the key size is limited. In March 1992, the Computer Security and Privacy Advisory Board

voted unanimously to recommend a national policy review of cryptographic issues, including export policy. The Board is an official advisory board to NIST (see Question 8.1) whose members are drawn from both the government and the private sector. The Board stated that a public debate is the only way to reach a consensus policy to best satisfy competing interests: national security and law enforcement agencies like restrictions on cryptography, especially for export, whereas other government agencies and private industry want greater freedom for using and exporting cryptography. Export policy has been decided solely by agencies concerned with national security, without significant input from those interested in encourage commerce in cryptography. U.S. export policy may change frequently over the next few years.

2 RSA

2.1 *What is RSA?*

RSA is a public-key cryptosystem for both encryption and authentication; it was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [66]. It works as follows:

Take two large primes, p and q, and find their product $n = pq$; n is called the modulus. Choose a number, e, less than n and relatively prime to $(p-1)(q-1)$, and find its inverse, d, mod $(p-1)(q-1)$, which means that $ed \equiv 1 \bmod (p-1)(q-1)$; e and d are called the public and private exponents, respectively. The public key is the pair (n, e); the private key is d. The factors p and q must be kept secret, or destroyed. All of the above operations are simple to perform. It is (presumably) difficult to obtain the private key d from the public key (n, e). Note that if one could factor n into p and q, one could obtain the secret key d. The entire security of RSA is predicated on the assumption that factoring is difficult. Here is how RSA can be used for privacy and authentication (in practice, actual use is slightly different; see Questions 2.13 and 2.14):

RSA privacy: suppose Alice wants to send a private message, m, to Bob. Alice creates the ciphertext c by exponentiating: $c = m^e \bmod n$, where e and n are Bob's public key. To decipher, Bob also exponentiates: $m = c^d \bmod n$, and recovers the original message m.

RSA authentication: suppose Alice wants to send a signed document m to Bob. Alice creates a signature s by exponentiating: $s = m^d \bmod n$, where d and n belong to Alice's key pair. She sends s and m to Bob. To verify the signature, Bob exponentiates and checks that the message m is recovered: $m = s^e \bmod n$, where e and n belong to Alice's public key.

Thus encryption and authentication take place without any sharing of private keys: each person uses only other people's public keys and his own private key. Anyone can send an encrypted message or verify a signed message using only public keys. Only those in possession of the correct private key can decrypt or

sign a message. The RSA system rests on the belief that factoring is difficult; an easy factoring method would "break" RSA (see Questions 2.6 and 4.4).

2.2 *Why use RSA rather than DES?*

RSA is not an alternative or replacement for DES; rather it supplements DES (or another bulk encryption cipher) and is used together with DES in a secure communications environment. (Note: for an explanation of DES, see Question 5.1.)

RSA allows two important functions not provided by DES: secure key exchange without prior exchange of secrets, and digital signatures. For encrypted messaging, RSA and DES are usually combined as follows: first the message is encrypted with a random DES key, and then, before being sent over an insecure communications channel, the DES key is encrypted with RSA. Together, the DES-encrypted message and the RSA-encrypted DES key are sent. In this protocol, known as an RSA digital envelope, no secret information is sent (unencrypted) over a public channel.

One may wonder, why not just use RSA to encrypt the whole message and not use DES at all? Although this may be fine for small messages, DES (or another cipher) is preferable for larger messages because it is much faster than RSA.

In some situations, RSA is not necessary and DES alone is sufficient. This includes multi-user environments where secure DES-key agreement can take place, for example by the two parties meeting in private. Also, RSA is not necessary in a single-user environment; for example, if you want to keep your personal files encrypted, just do so with DES using your personal password as the DES key. RSA only makes sense in a multi-user environment.

Any system in which digital signatures are desired needs RSA or some other public-key system.

2.3 *Are there hardware implementations of RSA?*

There are many commercially available hardware implementations of RSA, and there are frequent announcements of newer and faster chips. For a survey, see [12]. The fastest current RSA chip has a throughput of 64 Kbits/second; it is expected that RSA speeds will reach 1 Mbit/second within a year or so.

2.4 *How fast is RSA?*

An "RSA operation," whether for encrypting or decrypting, signing or verifying, is essentially a modular exponentiation, which can be performed by a series of modular multiplications.

In practical applications, it is common to choose a small public exponent for the public key; in fact, entire groups of users can use the same public ex-

ponent. This makes encryption faster than decryption and verification faster than signing. Algorithmically, public-key operations are $O(k^2)$ steps, private key operations are $O(k^3)$ steps, and key generation is $O(k^4)$ steps, where k is the number of bits in the modulus.

The fastest hardware implementations of RSA available today (see Question 2.3) achieve throughput greater than 64 Kbits per second with a 512-bit modulus (implying that it performs at least 128 RSA private-key operations per second); one was recently announced at the IEEE Custom Integrated Circuits Conference of May 1992. Chips are being planned that will approach or exceed 1 Mbit per second with a 512-bit modulus.

DES is much faster than RSA. In software, DES is generally at least 100 times as fast as RSA. In hardware, DES is between 1000 and 10,000 times as fast, depending on the implementations. RSA will probably narrow the gap a bit in coming years, as it finds growing commercial markets, but will never match the performance of DES.

2.5 *How much extra message length is caused by using RSA?*

Only a very small amount of data expansion is involved when using RSA. For encryption, a message may be padded to a length that is a multiple of the block length, which is the length of the modulus (currently 512 bits in most applications).

For authentication, the message is not encrypted and thus there is no message expansion; however, the signature itself is appended. An RSA signature is one block length. Sometimes certificates (see Question 3.5) may be included as well; certificates are used in conjunction with any digital signature method. A typical RSA certificate, with 512 bit moduli, is 300 bytes long; a message might include two certificates.

2.6 *What would it take to break RSA?*

There are a few possible interpretations of "breaking RSA". The most damaging would be for an attacker to discover the private key corresponding to a given public key; this would enable the attacker both to read all messages encrypted with the public key and to forge signatures. The obvious way to do this attack is to factor the public modulus, n, into its two prime factors, p and q. From p, q, and e, the public exponent, the attacker can easily get d, the private key. The hard part is factoring n; the security of RSA depends of factoring being difficult. In fact, the task of recovering the private key is equivalent to the task of factoring the modulus: you can use d to factor n, as well as use the factorization of n to find d. See Questions 4.5 and 4.6 regarding the state of the art in factoring. It should be noted that hardware improvements alone will not weaken RSA (as long as appropriate key lengths are used); in fact, hardware improvements should increase the security of RSA (see Question 4.5).

Another way to break RSA is to find a technique to compute e-th roots mod n. Since $c = m^e$, the e-th root of c is the message m. This attack would allow someone to recover encrypted messages and forge signatures even without knowing the private key. This attack is not known to be equivalent to factoring. No methods are currently known that attempt to break RSA in this way. The mere existence of a possible attack along these lines means that breaking RSA is not equivalent to factoring; such an equivalence would be desirable because then it is only necessary to keep track of one hard problem (factoring) and as long as factoring remains difficult the security of RSA would be assured. Other public-key systems have the advantage of being provable equivalent to either factoring or some other hard problem (see Question 2.17). Wiener has suggested attacks that are feasible when the private exponent is short [72]; this situation is easily avoided.

The attacks just mentioned are the only ways to break RSA in such a way as to be able to recover all messages encrypted under a given key. There are other methods, however, which aim to recover single messages; success would not enable the attacker to recover other messages encrypted with the same key. Suppose someone sends the same message m to three others, who each have public exponent $e = 3$. If an attacker knows this and sees the three messages, he will be able to recover the message m. This attack and ways to prevent it are discussed in [30]. There are also some chosen ciphertext attacks, in which the attacker creates his own ciphertext and sees the corresponding plaintext, perhaps by tricking a legitimate user into decrypting a fake message. Davida [21] gives some examples.

The simplest single-message attack is the guessed plaintext attack. An attacker sees a ciphertext, guesses that the message might be "Attack at dawn", and encrypts his guess with the public key of the recipient; by comparison with the actual ciphertext, he knows whether he was correct. This attack can be thwarted by appending some random bits to the message.

There are also, of course, attacks that aim not at RSA itself but at a given insecure implementation of RSA; these do not count as "breaking RSA" because it is not any weakness in the RSA algorithm that is exploited, but rather a weakness in a specific implementation. For example, if someone stores his private key insecurely, an attacker may discover it. One cannot emphasize strongly enough that to be truly secure RSA requires a secure implementation; mathematical security measures, such as choosing a long key size, are not enough. In practice, most successful attacks will likely be aimed at insecure implementations and at the key management stages of an RSA system. See Section 3 for discussion of secure key management in an RSA system.

2.7 *Are strong primes necessary in RSA?*

In the literature pertaining to RSA, it has often been suggested that in choosing a key pair, one should use "strong" primes p and q to generate the modulus n.

Strong primes are those with certain properties that make the product n hard to factor by specific factoring methods; desirable properties have included, for example, the existence of a large prime factor of $p-1$ and a large prime factor of $p+1$. The reason for these concerns is that some factoring methods are especially suited to primes p such that $p-1$ or $p+1$ has only small factors; strong primes are resistant to these attacks. Strong primes were recommended in the original RSA paper [66], by Knuth [36] and in a host of other technical papers; more recently, the X.509 international standard recommended the use of strong primes for RSA in 1989.

However, recent advances in factoring (see Question 4.6) appear to have obviated the need for strong primes; the elliptic curve factoring algorithm is one such advance. The new factoring methods have as good a chance of success on strong primes as on weak primes; therefore, choosing strong primes does not increase resistance to attacks. So for now the answer is negative: strong primes are not necessary or even beneficial in using RSA (although there is no danger in using them, except that it takes longer to generate a key pair). However, new factoring algorithms may be developed in the future which once again target primes with certain properties; if so, choosing strong primes may again help to increase security.

2.8 *How large a modulus should be used in RSA?*

It depends how on your security needs. The larger the modulus, the greater the security but also the slower the RSA operations. One should choose a modulus length upon consideration of one's security needs (e.g., the value of the protected data, how long it needs to be protected for) and also how much money one's potential enemy has. A good analysis of security obtained by a given modulus is given by Rivest [64], in the context of discrete logarithms modulo a prime, but it applies to RSA as well. Rivest estimates that a 512-bit modulus, currently the most common modulus length, can be factored with an $8.2 million effort today, less in the future. Those with extremely valuable data (or large potential damage from digital forgery) should use a larger modulus, say 700 or 800 bits in length. A certifying authority (see Question 3.5) should use a modulus of length 1000 bits or more, because the validity of many other key pairs depends on the security of the one central key.

It may be that the amount of time a signed document is considered legally valid will be a function of the length of the key used to sign it; then the choice of key length should also depend on how far into the future a document needs to remain valid. For example, a job application does not need to be valid for more than two years, so one's ordinary key would suffice. But one might want one's will to be valid for at least twenty years, so a longer key should be used to sign the will. See Question 3.17.

The key of an individual user will expire after a certain time, say, two years (see Question 3.11). Upon expiration, the user will generate a new key which

should be a few digits longer than the old key to reflect the speed increases of computers over the two years. Recommended key length schedules will probably be published by some authority or public body.

Users should keep in mind that the estimated times to break RSA are averages only. A large factoring effort, attacking many thousands of 512-bit moduli, will likely succeed in factoring at least one in a reasonable time. Although the security of any individual key is still strong, with some factoring methods there is always a chance that the attacker may get lucky and factor it quickly.

Regarding the cost in extra time (see Question 2.4), doubling the modulus length will, on average, increase the time required for public-key operations (encryption and signature verification) by a factor of 4 and increase the time for private key operations (decryption and signing) by a factor of 8; public keys can be kept small and constant. Key generation time will increase by a factor of 16, but this is not a common operation for most users.

2.9 *How large should the primes be?*

The two primes, p and q, which compose the modulus, should be of roughly equal length; this will make the modulus harder to factor than if one of the primes was very small. For example, if one chose to use a 512-bit modulus, one could use primes of lengths 255 and 257.

2.10 *How does one find random numbers for keys?*

One needs a source of random numbers in order to find two random primes to compose the modulus. If one used a predictable method of generating the primes, an adversary could mount an attack by trying to recreate the key generation process.

Random numbers obtained from a physical process are in principle the best. One could use a hardware device, such as a diode; some are sold commercially on computer add-in boards for this purpose. Another idea is to use physical movements of the computer user, such as keystroke timings measured in microseconds. By whichever method, the random numbers may still contain some correlations preventing sufficient statistical randomness. Therefore, it is best to run them through a good hash function (see Question 6.2), such as MD5, before actually using them.

Another approach is to use a pseudorandom number generator fed by a random seed. Since these are deterministic algorithms, it is important to find one that is very unpredictable and also to use a truly random seed. There is a wide literature on the subject of pseudorandom number generators. See Knuth [36] for an introduction.

Note that one does not need random numbers to determine the public and private exponents in RSA, after choosing the modulus. One can simply choose

an arbitrary value for the public exponent, which then determines the private exponent, or vice versa.

2.11 *What if users of RSA run out of distinct primes?*

There are enough prime numbers that RSA users will never run out of them. For example, the number of primes of length 512 bits or less exceeds 10^{150}, according to the prime number theorem; this is more than the number of atoms in the universe.

2.12 *How do you know if a number is prime?*

It is generally recommended to use probabilistic primality testing, which is much quicker than proving a number prime. One can use a probabilistic test to generate a number that is prime with error probability less than 2^{-100}. For further discussion of some primality testing algorithms, see the bibliography of [3]. For some empirical results on the reliability of simple primality tests see [62]; one can perform very fast primality tests and be extremely confident in the results. A simple algorithm for choosing probable primes was recently analyzed by Brandt and Damgard [9].

2.13 *How is RSA used for encryption in practice?*

RSA is combined with a secret-key cryptosystem, such as DES, to encrypt a message by means of an RSA digital envelope.

Suppose Alice wishes to send an encrypted message to Bob. She first encrypts the message with DES, using a randomly chosen DES key. Then she looks up Bob's public key and uses it to encrypt the DES key. The DES-encoded message and the RSA-encoded DES key together form the RSA digital envelope and are sent to Bob. Upon receiving the digital envelope, Bob decrypts the DES key with his private key, then uses the DES key to decrypt to message itself.

2.14 *How is RSA used for authentication in practice?*

Suppose Alice wishes to send a signed message to Bob. She uses a hash function on the message (see Question 6.2) to create a message digest, which serves as a "digital fingerprint" of the message. She encrypts the message digest with her RSA private key; this is her digital signature, which she sends to Bob along with the message itself. Bob, upon receiving the message and signature, decrypts the signature with Alice's public key to recover the message digest. He then hashes the message with the same hash function Alice used and compares the result to the message digest decrypted from the signature. If they are exactly equal, the signature has been successfully verified and he can be confident that the message did indeed come from Alice. If, however, they are not equal, then the message

either originated elsewhere or was altered in transmission, and he rejects the message.

In practice, the public exponent is usually much smaller than the private exponent; this means that verification of a signature is faster than the signing. This is desirable because a message or document will only be signed by an individual once, but the signature may be verified many times.

It must be infeasible for anyone to either find a message that hashes to a given value or to find two messages that hash to the same value. If it were feasible, an intruder could attach a false message onto Alice's signature. Hash functions such as MD4 and MD5 (see Question 6.3) have been designed specifically to have the property that finding a match is infeasible, and are therefore considered suitable for use in cryptography.

A certificate is a signed document attesting to the identity and public key of the person signing the message (see Question 3.5). Its purpose is to prevent someone from impersonating someone else, using a phony key pair. Accompanying a signature will be the name of a certifying authority and a serial number of a certificate; this will allow the recipient (or a third party) to check the authenticity of the public key. It may also be the case that one or more certificates are enclosed with a signed message.

2.15 *Does RSA help detect transmission errors?*

An RSA digital signature is superior to a handwritten signature in that it attests to the contents of a message as well as to the identity of the signer. As long as a secure hash function is used, there is no way to take someone's signature from one document and attach it to another, or to alter the signed message in any way.

For this reason, RSA allows recipients to detect any transmission errors in any messages they receive. Any alteration in the message will cause the verification procedure to fail. Of course, it does not allow the recipient to decide whether the cause of failure was a transmission error or an attempted forgery.

2.16 *Does RSA help protect against computer viruses?*

In the same way that RSA will detect a transmission error, it can be used to detect any alterations in a file stored on a disk. Since viruses act by altering files, they can be detected.

One method is to use RSA to sign every file, and later to verify the signatures; if a signature fails to verify, it indicates that the file has been changed, possibly by a virus. Of course, it could have been changed for other reasons, such as recompilation of source code or physical problems on the hard disk. If the change looks suspicious, one would need to follow up by running a specific virus detection/elimination program.

Another method of virus protection is for commercial software vendors to supply a digital signature with their programs. A user can verify this signature, using the vendor's public key, either when first installing the software or anytime thereafter. Of course, a clever virus might attempt change the public key used to verify the signature and thus escape detection. And the virus-checking program itself would be a likely target for viruses. A more secure method might be to incorporate the virus-checking capability in the operating system, but again, the operating system must contain strong mechanisms to protect itself.

2.17 *What are alternatives to RSA?*

Other public-key cryptosystems have been proposed. A mathematical problem called the knapsack problem was the basis for several systems [46], but these have lost favor because many versions were broken. Another system, designed by ElGamal [25], was the basis for several later signature methods, including one by Schnorr [67], which in turn was the basis for the digital signature standard proposed by NIST [54] (see Question 7.1). Because of the NIST proposal, the relative merits of these signature systems versus RSA signatures has received a lot of attention; see [53] for a discussion. The ElGamal system has been used successfully in applications. It is slower for encryption and verification than RSA and its signatures are larger than RSA signatures.

There have been proposed cryptosystems based on discrete exponentiation in the finite field GF(2^n); their advantage is that they work efficiently in hardware (faster than RSA). However, doubts have been raised about the security of these systems because the underlying problem may be easier to solve than factoring [56, 28].

For key exchange only, Diffie and Hellman [24] proposed a system in 1976, before RSA; it permits secure exchange of keys in an otherwise conventional secret-key system. A group of users share a common modulus; one must be careful to make the common modulus large, because its compromise would compromise all users in the group. This system is in use today.

Probabilistic encryption has the attraction of being resistant to a guessed ciphertext attack (see Question 2.6), but at a cost of data expansion. Interesting probabilistic encryption methods were proposed by Goldwasser and Micali [27] and by Blum and Goldwasser [8]. In probabilistic encryption, the same plaintext encrypted twice under the same key will give, in general, two different ciphertexts.

For digital signatures, Rabin [60] proposed a system which is provably equivalent to factoring; this is an advantage over RSA, where one may still have a lingering worry about an attack unrelated to factoring. Rabin's method is susceptible to an active chosen message attack, however, in which the attacker tricks a signer into signing a certain message. Another signature scheme, by Fiat and Shamir [26], is based on interactive zero-knowledge protocols, but can be adapted for signatures. It is faster than RSA and is provably equivalent to

factoring, but the signatures are much larger than RSA signatures. Other variations, however, lessen the necessary signature length; see [15] for references. These identity-based systems may be better suited to smart card applications than to network communications.

Cryptosystems based on mathematical operations on elliptic curves have been gaining popularity in recent years. Koblitz [38] and Miller [49] have written introductions to this topic.

Major advantages of RSA over other public-key cryptosystems include the fact that it can be used for both encryption and authentication, and that it has been around for many years and has successfully withstood much scrutiny. RSA has received far more attention, study, and actual use than any other public-key cryptosystem, and thus RSA has more empirical evidence of its security than more recent and less scrutinized systems. In fact, a large number of public-key cryptosystems which at first appeared secure were later broken; see [13] for some case histories. RSA's resistance to attack during many years of vigorous attempts to break it is unrivaled by any other public-key system.

2.18 *Is RSA currently in use today?*

RSA is used in a wide variety of products, platforms and industries around the world. It is found in many commercial software products, such as Lotus' Notes and Delrina's PerForm Pro, an electronic forms software package that incorporates RSA digital signatures. RSA is being built into operating systems by Microsoft, IBM, Apple, Sun, Digital and Novell. In hardware, RSA can be found in Secure Telephone Units by Motorola and AT&T, as well as on Xerox Ethernet cards. Many other vendors, such as Wordperfect Corp., have announced plans to incorporate RSA into products.

RSA is used in many branches of the U.S. government, including NASA, the CIA, the Departments of Defense, State and Labor, and the national laboratories, such as Lawrence Livermore and Sandia National Labs; it is not clear how widely used RSA is within these organizations. The government can use RSA without licensing the patent.

Major corporations have chosen RSA for internal use; examples include Boeing, Shell Oil, DuPont, Raytheon, and Citicorp. Research institutions using RSA include the University of California, Bellcore, and the National Science Foundation.

RSA is even more widely used in Europe than in the U.S. For example, RSA is the standard of the European financial community for electronic funds transfer (see Question 2.19), whereas the U.S. banking industry has not yet incorporated RSA into routine use (although it may do so with upcoming official standards).

The use of RSA is undergoing a period of rapid expansion and may become ubiquitous within a few years. Adoption of RSA seems to be proceeding more quickly for authentication (digital signatures) than for privacy (encryption),

perhaps because products for authentication are easier to export than those for privacy.

2.19 *Is RSA an official standard today?*

RSA is part of many official standards worldwide. The ISO (International Standards Organization) 9796 standard [32] lists RSA as an acceptable cryptographic algorithm, as does the Consultative Committee in International Telegraphy and Telephony (CCITT) X.509 security standard [17]. RSA is part of both the Society for Worldwide Interbank Financial Telecommunication (SWIFT) standard and the French financial industry's ETEBAC 5 standard [18]. The Australian digital signature standard, AS2805.6.5.3 [71], also specifies RSA.

RSA is found in Internet's proposed PEM (Privacy Enhanced Mail) standard (see Question 6.6) and the PKCS standard for the software industry (see Question 6.7). The OSI Implementors' Workshop (OIW) has issued implementors' agreements [57] referring to PKCS and PEM, which each include RSA.

A number of other standards are currently being developed and will be announced over the next couple of years; many are expected to include RSA as either an endorsed or a recommended system for privacy and/or authentication.

2.20 *Is RSA a de facto standard today?*

RSA is the most widely used public-key cryptosystem today and has often been called a de facto standard. Its use is widespread in many areas and industries (see Question 2.18) and growing rapidly. Furthermore, use of RSA far outstrips use of any other public-key system (see Question 2.17).

Regardless of the official standards, the existence of a de facto standard is extremely important for the development of a digital economy. If one public-key system is used all over for authentication, then signed digital documents can be exchanged between users in different nations using different software on different platforms; this interoperability is necessary for a true digital economy to develop.

The lack of secure authentication has been a major obstacle in achieving the promise that computers would replace paper; paper is still necessary almost everywhere for contracts, checks, official letters, legal documents, and identification. With this core of necessary paper transaction, it has not been feasible to transform completely into a society based on electronic transactions. Digital signatures and their verifiability are the exact tools necessary to convert the most resistant paper-based documents to digital electronic media. Digital signatures make possible the existence of leases, wills, college transcripts, checks, and voter registration forms that exist only in electronic form; any paper version would just be a "copy" of the electronic original. All of this is enabled by a standard for authentication.

2.21 *What will happen to RSA if NIST adopts another method to be the official standard?*

If NIST adopts a method other than RSA as an official standard, both RSA and the NIST standard would coexist, at least for the foreseeable future; NIST has recently proposed a non-RSA method, DSS, for digital signatures (see Question 7.1). RSA has already established itself as the standard in Europe; that status would not change due to a NIST action, so anyone doing business with or in Europe would need to use RSA. RSA is also the most widely used in commercial software in the U.S., so users wishing to maintain compatibility and interoperability would continue to use RSA. The NIST standard, however, would be used by the U.S. government, so software vendors and others doing business with the government would need to use the NIST standard. The most likely solution for software manufacturers is to provide both RSA and NIST standard capabilities and let users choose whichever method most suits their purposes. Some people have voiced concern that a NIST standard (non-RSA) may be less secure than RSA (see Question 7.2). Although NIST has recently taken steps to reassure the public that its (proposed) standards are secure, those still distrustful would use RSA.

In the long term, one public-key system is likely to become so dominant as to eliminate altogether the use of any other system, leaving a single standard in place. If NIST chooses an alternative to RSA, the establishment of a single global standard will not occur for many years.

2.22 *Is RSA patented?*

RSA is patented under U.S. Patent 4,405,829, issued 9/20/83 and owned by Public Key Partners (PKP), of Sunnyvale, California; the patent expires 17 years after issue, in 2000. RSA is usually licensed together with other public-key cryptography patents (see Question 1.5). PKP has a standard, royalty-based licensing policy which can be modified in the case of special circumstances. PKP has also publicly told the U.S. government that it will make licenses for RSA "available under reasonable terms" if RSA were chosen as an official government standard. If a software vendor, having licensed the public-key patents, incorporates RSA into a commercial product, then anyone who purchases the end product has the legal right to use RSA within the context of that software. The U.S. government can use RSA without a license because it was invented at MIT with government funding. RSA is not patented outside North America.

In North America, a license is needed to "make, use or sell" RSA. However, PKP has a policy that anyone may use RSA non-commercially for a personal, academic or intellectual reason without a license; an example of such use would be the implementation of RSA as a programming project for a computer class. RSA Laboratories has made available (in the U.S. and Canada) at no charge a collection of cryptographic routines in source code, including the RSA algorithm;

it can be used for non-commercial purposes (see Question 6.8).

2.23 *Can RSA be exported from the U.S.?*

Export of RSA falls under the same U.S. laws as all other cryptographic products. See Question 1.6 for details.

RSA used for authentication is more easily exported than when used for privacy. In the former case, export is allowed regardless of key (modulus) size, although the exporter must demonstrate that the product cannot be easily converted to use for encryption. In the case of RSA use for privacy (encryption), the U.S. government generally does not allow export if the key size exceeds 512 bits. Export policy is currently a subject of debate, and the export status of RSA may well change in the next year or two.

Regardless of U.S. export policy, RSA is available abroad in non-U.S. products.

3 Key Management

3.1 *What key management issues are involved in public-key cryptography?*

Secure methods of key management are extremely important. In practice, most attacks on public-key systems will probably be aimed at the key management levels, rather than at the cryptographic algorithm itself. The key management issues mentioned here are discussed in detail in later questions.

Users must be able to get a key pair suited to their security and efficiency needs. There must be a way to look up people's public keys. Users must be confident of the accuracy of someone's public key; otherwise an intruder can either change public keys listed in a directory, or represent himself as another user. Conversely, users must be able to publicize their public keys so that others will have confidence in their accuracy. Certificates are used for this purpose. Certificates must be securely obtainable, not forgeable, and used in such a way that an intruder cannot misuse them. The issuance of certificates must proceed in a secure way, impervious to attack. If someone's private key is lost or compromised, others must be made aware of this and no longer encrypt messages under the public key nor accept messages signed with the compromised private key. Users must be able to store their private keys securely, so that no intruder can find it, yet readily accessible for legitimate use. Keys need to be valid only until a specified expiration date. The expiration date must be chosen properly and publicized securely. Some documents need to have verifiable signatures beyond the time when the key used to sign them has expired.

Although most of these key management issues arise in any public-key cryptosystem, for convenience they are discussed here in the context of RSA.

3.2 *Who needs a key?*

Anyone who wishes to sign messages or to receive encrypted messages must have a key pair. People may have more than one key. For example, someone might have a key affiliated with his or her work and a separate key for personal use.

Furthermore, other entities will have RSA keys. This can include electronic entities such as modems, workstations, and printers. It can also mean organizational entities such as a corporate department, a hotel registration desk, or a university registrar's office.

3.3 *How does one get an RSA key pair?*

Each RSA user should generate his or her own RSA key pair. It may be tempting within an organization to have a single site that generates keys to all members who request one, but this is a security risk because it involves the presence and transmission of private keys over a network as well as catastrophic consequences in the case of an attacker infiltrating the key-generation site. Each node on a network should be capable of local key generation, so that private keys are never transmitted and so no external key source must be trusted. Networks with trusted operating systems may have a central node to perform key generation.

Once generated, a user must register his or her public key with some central administration, called a certifying authority. The certifying authority returns to the user a certificate attesting to the veracity of the user's public key along with other information (see Questions 3.5 and following). Most users should not obtain more than one certificate for the same key, in order to simplify various bookkeeping tasks associated with the key.

3.4 *Should a public key or private key be shared among users?*

In RSA, modulus and private key should be unique to every user. The public exponent can be common to a group of users without security being compromised. Public exponents in use today are 3 and $2^{16} + 1$; because they are small, the public-key operations (encryption and signature verification) are fast relative to the private key operations (decryption and signing). If one public exponent becomes a standard, software and hardware can be optimized for that value.

In public-key systems based on discrete logarithms, such as ElGamal, Diffie-Hellman, or DSS, it has often been suggested that a group of people should share a modulus. This would make breaking a key more attractive to an attacker however, because one could break every key with only slightly more effort than it would take to break a single key. To an attacker, therefore, the average cost to break a key is much lower than if every key had a separate modulus. Thus one should be very cautious about using a common modulus.

3.5 *What are certificates?*

Certificates are digital documents attesting to the binding of a public key to an individual or other entity. They allow verification of the claim that a given public key does in fact belong to a given individual. Certificates aim to prevent someone from misrepresenting himself or herself under another name with a phony key.

Certificates contain information about a public key, including the public key itself, the name of the person to whom it is issued, the expiration date of the key, the length of the modulus, the name of the organization that issued the certificate, and the serial number of the certificate. The certificate may also contain optional information, such as the organization or title of the person to whom it is issued. Most importantly, it contains the digital signature of the issuer. The format of certificates is governed by the CCITT X.509 international standard; thus certificates can be read or written by any application complying with X.509. Further refinements are found in the PKCS set of standards (see Question 6.7), which are extensions of X.509.

The certificate is issued by a certifying authority (see Question 3.7) and signed with the certifying authority's private key.

3.6 *How are certificates used?*

One exhibits one's certificate in order to assure confidence in one's public key. On the verifier's side, the signer's certificate can itself be verified to assure that no forgery or false representation has occurred. These uses can be performed with greater or lesser rigor depending on the context in which RSA is being used.

The most secure use of authentication involves enclosing one or more certificates with every signed message. The receiver of the message would verify the certificate with the certifying authority's public key and, now confident of the public key of the individual sender, verify the message's signature. There may be two or more certificates enclosed with the message, forming a hierarchical chain, wherein one certificate testifies to the authenticity of the previous certificate. At the end of a certificate hierarchy is a top-level certifying authority, which is trusted without a certificate from any other certifying authority. The public key of the top-level certifying authority must be independently known, for example by being widely published.

The more familiar the sender is to the receiver of the message, the less need there is to enclose certificates. If Alice sends messages to Bob every day, Alice can enclose a certificate chain on the first day, which Bob verifies. Bob thereafter stores Alice's public key and no more certificates are necessary. A sender whose company is known to the receiver may need to enclose only one certificate (issued by the company), whereas a sender whose company is unknown to the receiver may need to enclose two certificates. A good rule of thumb is to enclose just

enough of a certificate chain so that the issuer of the highest level certificate in the chain is well-known to the receiver.

According to the PKCS standards for public-key cryptography (see Question 6.7), every signature points to a certificate that validates the public key of the signer. Specifically, it contains the name of the issuer of the certificate and the serial number of the certificate. Thus even if no certificates are enclosed with a message, a verifier can still use the certificate chain to check the status of the public key.

3.7 *Who issues certificates and how?*

Certificates are issued by a certifying authority (CA), which can be any trusted central administration willing to play the role of vouching for the identities of those to whom it issues certificates. A company may issue certificates to its employees, a university to its students, a town to its citizens. Other CAs will be available to issue certificates to unaffiliated individuals. In order to prevent forged certificates, a CA must either publicize its public key or provide a certificate from a higher-level CA attesting to the validity of its public key. In this way, hierarchies of certifying authorities will form.

Certificate issuance proceeds as follows. Alice generates her own key pair and sends the public key to an appropriate CA with some proof of her identification. The CA checks the requester's identification and if the request really did come from Alice, sends her a certificate attesting to the binding between Alice and her public key, along with a hierarchy of certificates verifying the CA's public key. Alice can include this certificate chain whenever desired in order to demonstrate the legitimacy of her public key. In order to simplify bookkeeping operations associated with verifying the signature, Alice should not request a certificate from any other certifying authority.

Since the CA must check for proper identification, it will prove convenient for a local organization to become a CA for the purpose of issuing certificates to its own members and employees. There will also be CAs to issue unaffiliated certificates.

Different CAs may issue certificates with varying levels of identification requirements. One CA may insist on seeing a driver's license, another may want the certificate request form to be notarized, yet another may want fingerprints of anyone requesting a certificate. Each CA must publish its own identification requirements and standards, so that verifiers can attach the appropriate level of confidence in the certified name-key bindings.

An example of a certificate-issuing protocol is Apple Computer's upcoming Open Collaborative Environment (OCE); Apple OCE users can generate a key pair and then request and receive a certificate for the public key. The certificate request must be notarized.

A public key may be recertified upon expiration, if it has not been compromised and if the modulus is long enough to warrant the recertification.

3.8 *What is a CSU, or, How do certifying authorities store their private keys?*

It is extremely important that private keys of certifying authorities are stored securely, because compromise would enable undetectable forgeries. For the highest security, CAs can keep their keys in a CSU, or Certificate Signing Unit. The CSU is a high-security, tamper-proof hardware box, which destroys its contents if ever opened. The CSU must be secure against attacks using electromagnetic radiation. Not even employees of the certifying authority should have access to the private key itself, but only the ability to use the private key in the process of issuing certificates.

There are many possible designs for CSUs; here is a description of one design commonly found in current implementations. The CSUs are activated by a set of data keys, which are physical keys capable of storing digital information. The data keys use secret-sharing technology such that several people must all use their data keys to activate the CSU. This prevents one disgruntled CA employee from producing phony certificates. There may also be separate physical keys which store private keys of certificate issuers in encrypted form. These are used for backup only, and their information can only be read by a CSU box with the correct RSA private key inside.

Note that if the CSU is destroyed, say in a fire, no security is compromised. Certificates signed by the CSU are still valid, as long as the verifier uses the correct public key. Some CSUs will be manufactured so that a lost private key can be restored into a new CSU. See Question 3.10 for discussion of lost CA private keys.

Some smaller CAs may choose to store their private keys in software rather than in hardware. If implemented securely, this is a viable and less expensive alternative for organizations which choose not to purchase CSUs. Eventually, CSUs will be sufficiently inexpensive for all CAs to use.

3.9 *Are certifying authorities susceptible to attack?*

One can think of many attacks aimed at the certifying authority.

Consider the following attack. Suppose Bob wishes to impersonate Alice. If Bob can convincingly sign messages as Alice, he can send a message to Alice's bank saying "I wish to withdraw $10,000 from my account. Please send me the money." To carry out this attack, Bob generates a key pair and sends the public key to a certifying authority saying "I'm Alice. Here is my public key. Please send me a certificate." If the CA is fooled and sends him such a certificate, he can fool the bank, and his attack will succeed. In order to prevent such an attack the CA must verify that a certificate request did indeed come from its purported author. It must require sufficient evidence that it is Alice and not anyone else who is requesting the certificate. It may, for example, require Alice to appear in person and show a birth certificate and take her fingerprints. Every

CA must publicly state its identification requirements and policies. Others can then attach an appropriate level of confidence to the certificates.

If the private key of a certifying authority should become known to an attacker, the attacker could forge certificates, allowing an accomplice to misrepresent himself (see Question 3.10). For this reason, a certifying authority must take extreme precautions to prevent illegitimate access to its private key. The private key should be kept in a high-security box, known as a Certificate Signing Unit (CSU), which destroys its contents if ever opened. See Question 3.8 for details about the CSU.

The certifying authority's public key might be the target of an extensive factoring attack. For this reason, CAs should use very long keys, preferably 1000 bits or longer. A CA should also change its key every year or two. Top-level certifying authorities are exceptions: it may not be practical for them to change keys so frequently because the key may be written into software used by a large number of verifiers.

Consider the following attack. Alice bribes Bob, who works for the certifying authority, to issue to her a certificate in the name of Fred. Now Alice can send messages signed in Fred's name and anyone receiving such a message will believe its authenticity because a full and verifiable certificate chain will accompany the message. This attack can be hindered by requiring the cooperation of two (or more) employees to generate a certificate; the attacker now has to bribe two employees rather than one. For example, is some of today's CSU boxes, three employees must each insert a data key containing secret information in order to authorize the CSU to generate certificates. Unfortunately, there may be other ways to generate a forged certificate by bribing only one employee. If each certificate request is checked by only one employee, that one employee can be bribed and slip a false request into a stack of real certificate requests. Note that a corrupt employee cannot reveal the certifying authority's private key to an attacker, as long as it is properly stored.

Another attack is to steal the CSU box; to succeed, however, the attacker must also steal the correct number of data keys to activate the CSU. More devious is for an attacker to surreptitiously replace the CSU box with another of his own devising. When the data keys are inserted, the fake box can record the secret information that authorizes the real box to produce certificates and then transmit the information back to the attacker, who can now get the real box to produce some phony certificates. A protocol in which the CSU is challenged to sign a random test message before insertion of data keys might block this attack, although an extremely sophisticated fake CSU could surmount even this precaution.

Consider the following attack. Alice tries to factor the modulus of the certifying authority. It takes her 15 years, but she finally succeeds, and she now has the old private key of the certifying authority. The key has long since expired, but she can forge a certificate dated 15 years ago attesting to a phony public key of some other person, say Bob; she can now forge a document with a signature

of Bob dated 15 year ago (perhaps a will leaving everything to Alice). The underlying issue is how to verify a signed document dated many years ago (longer than the key expiration period). This issue is discussed in Question 3.17.

Note that the certifying authority never sees the private keys of those to whom it issues certificates, so it cannot betray its customers in that way.

3.10 *What if the certifying authority's key is lost or compromised?*

If the certifying authority's key is lost or destroyed but not compromised, certificates signed with the old key are still valid, as long as the verifier knows to use the old public key to verify the certificate.

A CA which loses its key can restore it in the following way. The CA first notifies the manufacturer of the CSU, who then supplies a new CSU identical from the one that held the key. The CA then loads the lost key into the new CSU by using a securely encrypted form of the lost key. This secure recovery method depends on each CA storing its key outside its CSU in an extremely secure fashion: the key must be stored in encrypted form, such that it can only be decrypted by a CSU box identical (with the same unique internal information) to that in which the key was generated, and, preferably, the cooperation of several people is required for restoring into a new box.

A compromised CA key is a much more dangerous situation. An attacker who discovers a certifying authority's private key can issue phony certificates in the name of the certifying authority, which would enable undetectable forgeries; for this reason, all precautions must be taken to prevent compromise, including those outlined in Questions 3.8 and 3.9. If a compromise does occur, the CA must immediately cease issuing certificates under its old key and change to a new key. If it is suspected that some phony certificates were issued, all certificates must be recalled, going back until before the compromise. In fact, all certificates signed with the compromised key should be reissued, because any of them could be a forgery, backdated if necessary. This could be relaxed if certificates were registered with a digital timestamping service (see Question 3.18). Compromise of a top-level CA's key should be considered catastrophic.

3.11 *For how long is a key pair valid?*

In order to guard against a long-term factoring attack, every key must have an expiration date after which it is no longer valid. The time to expiration must therefore be much shorter than the expected factoring time, or, from the other perspective, the key length must be long enough to make the chances of factoring within the expiration time extremely small (see Question 2.8).

Currently it is recommended that users' keys expire two years after issue [35]. One should choose a key size appropriate to this period of time (see Question 2.8). After expiration, the user needs to choose a new key, which should be longer than the old key, perhaps by several digits, to reflect the speed increase

of computer hardware, and any improvements in factoring algorithms, during the two years. Recommended key length schedules would be published.

The expiration date of a key accompanies the public key in a certificate or a directory listing. The signature verification program should check for expiration; one should not accept a message signed with an expired key. This means that when one's own key expires, everything signed with it will no longer be considered valid. Of course, there may be cases where it is important that a document is considered valid for a much longer period of time (see Question 3.17). One possibility is to use an extra-long key to sign such a document. This implies that the length of time a signed document is valid should be a function of the key length.

One may also recertify a key that has expired, if the modulus length is sufficient and if the key has not been compromised. The certifying authority would issue a new certificate for the same key. All new signatures would point to the new certificate instead of the old; in fact, old signatures could be altered to point to the new certificate instead of the old (this part of the signature is in plaintext and easily updated). In order to make these solutions practical and efficient, each key should be certified by exactly one certifying authority.

However, the fact that computer hardware continues to improve argues for replacing expired keys with new, longer keys every few years. Key replacement enables one to take advantage of the hardware improvements to increase the security of the RSA system. Faster hardware has the effect of increasing security (perhaps vastly), but only if key lengths are increased regularly (see Question 4.5).

3.12 *What happens if I lose my private key?*

It may happen that your private key may be lost or destroyed, but not compromised; this can happen, for example, if you forget the password used to access your key. In this case, you can no longer sign or decrypt messages, but anything previously signed with the lost key is still valid. You need to choose, certify and publish a new key as quickly as possible to minimize the number of messages people send you encrypted under your old key, messages which you can no longer read.

3.13 *What happens if my private key is compromised?*

If your private key is compromised, that is, you suspect an attacker may have obtained your private key, then you must assume that some enemy can read encrypted messages sent to you and forge your name on documents. The seriousness of these consequences underscore the importance of protecting your private key with extremely strong mechanisms (see Question 3.15).

You must immediately notify your certifying authority and have your old key placed on a Certificate Revocation List (see Question 3.14); this will inform

people that the key has been revoked. Then choose a new key and obtain the proper certificates for it. It may be a good idea to use the new key to resign documents that you signed with the compromised key. You should also increase the security of the device you use to store your private key.

3.14 *What are Certificate Revocation Lists (CRLs)?*

CRLs are lists of public keys that have been revoked before their scheduled expiration date. There are several reasons why a key might need to be revoked and placed on a CRL. The compromise of a key is one case. Also, some keys might be associated with an individual at a company; for example, the official name associated with a key might be "Alice Avery, Vice President, Argo Corp." If Alice were fired, her company might not want her to be able to sign messages with that name and therefore the company would place her key on the CRL. If a government employee were discovered to be a spy, his key would be placed on the CRL.

When verifying a signature, one can check the relevant CRL to be sure the signer's key has not been revoked (which would indicate a possible forgery). Whether it is worth the time to perform this check depends on the importance of the signed document. Signatures on legal contracts, for example, should always be checked against their relevant CRLs.

CRLs are maintained by certifying authorities (CAs) and provide information about revoked keys originally certified by the CA. CRLs only list current keys because expired keys should not be accepted in any case; when a revoked key is past its original expiration date it is removed from the CRL. Although maintained in a distributed manner, there will be central repositories for CRLs, that is, sites on networks containing the latest CRLs from many organizations. An institution like a bank might want an in-house CRL repository to make CRL searches feasible on every transaction.

One reason why a public key should only be certified by exactly one CA is to simplify bookkeeping operations such as a CRL lookup or entering a key in a CRL.

3.15 *How should I store my private key?*

Private keys must be stored securely, since forgery and loss of privacy could result from compromise. For an individual user, keeping the private key encrypted and stored in software should suffice. For example, a password could serve as a DES key for encrypting the private key. The private key should never be stored anywhere in plaintext form. Of course, the password itself must be maintained with high security, not written down, and not easily guessed; the password may prove an easier target for cryptographic attack than factoring the public key. One idea is to keep the password only on a local workstation, one not accessible by a network; unfortunately, this prevents the user from signing messages when

away from his workstation. Ultimately, private keys will be stored in encrypted form on portable hardware, such as a smart card; the move to hardware storage will mean a large jump in security.

Users with particularly high security needs, including certifying authorities, should use special hardware boxes to protect their keys. Features of these boxes include mechanisms that will destroy the contents if ever opened and devices to require two separate keys to use the box for signing or decryption. Some issues surrounding their use are discussed in Question 3.9.

3.16 *How does one find someone else's public key?*

There are many possible ways find someone's key. You could call him up and ask him to send you his public key via email; you could request it via email as well. Certifying authorities may serve as directories; if the person in question works for company Z, look in the directory kept by the Z certifying authority. Directories must be secure against unauthorized tampering, because users of a directory must be confident that a public key listed in the directory actually belongs to the person listed. Otherwise, you might send private encrypted information directly to your enemy.

Eventually, full-fledged directories will arise, serving as online white or yellow pages; you will be able to look up a name and get a public key. If they are compliant with CCITT X.509 standards, the directories will contain certificates along with public keys; the presence of certificates will lower the security needs of the directory.

People might have multiple keys, for example, a work key, a personal key, and a long-modulus key for long-term documents. Each key, however, is certified only once.

3.17 *How can signatures remain valid beyond the expiration dates of their keys, or, How do you verify a 20-year-old signature?*

Normally, a key expires after two years and a document signed with an expired key should not be accepted. However, there are many cases where it is necessary for signed documents to be regarded as legally valid for much longer than two years; long-term leases and contracts are examples. How should these cases be handled? Many solutions have been suggested but it is unclear which will prove the best. Here's some possibilities.

One can have special long-term keys as well as the normal two-year keys. Long-term keys should have much longer modulus lengths and be stored more securely than two-year keys. If a long-term key expires in 50 years, any document signed with it would remain valid within that time. One problem with this method is that any compromised key must remain on the relevant CRL until expiration (see Question 3.14); if 50-year keys are routinely placed on CRLs, the CRLs could grow in size to unmanageable proportions.

The previous idea can be modified as follows. For long-term documents choose a key with a very long modulus. The key will expire in only two years, as normal. At expiration time, if it has not been compromised, the key can be recertified, that is, issued a new certificate by the certifying authority, so the key will be valid for another two years. Depending on the key size, a maximum number of recertifications would be allowed. Now a compromised key only needs to be kept on a CRL for at most 2 years, not 50. Some mechanism will allow a verifying party to know that the key is still valid since a proper certificate chain still validates it (even though it is a different certificate chain than that used to originally sign the document). There is a part of every signature that points to a certificate authenticating the signing key. This certificate name and serial number is in plaintext, as specified by the PKCS standard, and could be updated to point to the new certificate. Another method is to simply update the key expiration date in the certifying authority's key database/directory. A verifier can get the name of the certifying authority from the signature and then check the expiration. One problem with these methods is that someone might try to invalidate a long-term contract by refusing to renew his key. This problem can be circumvented by registering the document with a digital timestamping service (see Question 3.18) at the time it is originally signed.

Another solution is to resign a document with a new key whenever the old key expires. The resigned document includes the previous signatures. This shares with the previous solution the problem that arises if one signer of a multiparty contract refuses to resign; use of a digital timestamping service might enable a better solution.

3.18 *What is a digital timestamping service?*

A digital timestamping service (DTS) issues timestamps which associate a date and time with a digital document in a cryptographically strong way. The digital timestamp (DT) can be used at a later date to prove that an electronic document existed at the time stated on its timestamp. For example, when a physicist has a brilliant idea, he can write about it on his word processor and have the document timestamped. The timestamp and document together can later prove that he deserves the Nobel Prize, even though his arch rival may have been the first to publish.

Here's how it works. Suppose Alice signs a document and wants it timestamped. She computes a message digest of the document using a secure hash function (see Question 6.2) and then sends the message digest (but not the document itself) to the DTS, which sends her in return a digital timestamp, which is a document consisting of the message digest, the date and time it was received at the DTS, and the signature of the DTS. Note that the message digest does not reveal any information about the content of the document; therefore the DTS cannot eavesdrop on the documents it timestamps. Later, Alice can present the document and timestamp together to prove when it was written. A

verifier computes the message digest of the document, makes sure it matches the digest on the timestamp, and then verifies the signature of the DTS on the timestamp.

To be reliable, the timestamps must not be forgeable. First, the DTS itself must have an extremely long key; if we want the timestamps to be reliable for 100 years, the DTS may need a key several thousand bits long. Second, the private key of the DTS must be stored in utmost security; it should only exist inside a box which erases its memory upon any tampering. Third, the date and time must come from a clock, also inside the tamperproof box, which cannot be reset and which will keep accurate time for years, perhaps for decades. Fourth, the timestamps must only be able to be created by using the tamperproof box with the date, time, and private key supplied from inside the box.

A cryptographically strong DTS using only software has been suggested [29], but it requires other clients of the DTS to save their timestamps and to cooperate in the timestamp verification process. Modified versions may avoid such requirements.

The use of a DTS would appear to be extremely important, if not essential, for maintaining the validity of documents over many years (see Question 3.17). Suppose a landlord and tenant sign a twenty-year lease. The public keys used to sign the lease will expire after two years; solutions such as recertifying the keys or resigning every two years with new keys require the cooperation of both parties several years after the original signing. If one party, perhaps the landlord, becomes dissatisfied with the lease, he or she will refuse to cooperate. What should be done is to register the lease with the DTS at the time of the original signing, and both parties should receive a copy of the timestamp, which can be used years later to enforce the integrity of the original lease.

3.19 *What other digital services will accompany widespread use of RSA?*

Widespread use of public-key cryptography will spur the development of other digital services. In fact, the full integration of digital authentication into an economy and society requires the presence of other services to complement the privacy and authentication of public-key cryptography. This digital infrastructure is just now beginning to emerge. Some of the main digital support services are briefly described below; they are discussed in greater detail elsewhere in this document.

Certifying authorities (CAs) (see Question 3.7) issue certificates testifying to the binding between a public key and a name. They can be used by someone verifying a signature to check that the public key used for the verification does indeed belong to the purported signer. CAs are necessary to prevent forgery through fake keys. Many companies and other organizations will become CAs and issue certificates to their employees and members.

CRL repositories are central, public locations where CRLs (certificate revocation lists) will be stored and maintained (see Question 3.14). They are used by a signature verifier to insure that the public key of the signer has not been compromised and thus is still valid. CRL repositories are necessary to minimize the damage that can result from the theft of a private key.

Directories (see Question 3.16) store lists of public keys and their associated names, organizations and expiration dates, as well as many other attributes about the listed objects. They are analogous to today's phone directories, although expanded in function. Directories will be used by anyone who wishes to send an encrypted message to, or verify a signed message from, another person whose public key is not already known to the first person. Directories are necessary to ensure the global aspect of electronic communications using public-key cryptography; otherwise, such communications would be more or less confined within a local organization or other community.

A digital timestamping service (DTS) (see Question 3.18) issues documents that can be used later to verify that a given message existed at a given time. A DTS is necessary to maintain the long-term integrity of signed digital documents, and will be used for everything from long-term corporate contracts to personal diaries and letters. Today, if an historian discovers some lost letters of Mark Twain, their authenticity is checked by physical means. But a similar find 100 years from now may consist of an author's letters in computer files; digital timestamps may be the only way to authenticate the find.

Many other digital services will appear, such as electronic cash, electronic passports, and electronic journalism. Predictions cannot be made with much accuracy.

4 Factoring and Discrete Log

4.1 *What is a one-way function?*

A one-way function is a mathematical function that is significantly easier to perform in one direction (the forward direction) than in the opposite direction (the inverse direction). One might, for example, compute the function in minutes but only be able to compute the inverse in months or years. A trap-door one-way function is a one-way function where the inverse direction is easy if you know a certain piece of information (the trap door), but difficult otherwise.

4.2 *What is the significance of one-way functions for cryptography?*

Every public-key cryptosystem is based on a (presumed) trap-door one-way function; this was realized by the inventors of public-key cryptography [24]. The public key gives information about the particular instance of the function;

the secret key is information about the trap door. Whoever knows the trap door can perform the function easily in both directions, but anyone lacking the trap door can perform the function only in the forward direction. The forward direction is used for encryption and signature verification; the inverse direction is used for decryption and signature generation.

In almost all public-key systems, the size of the key corresponds to the size of the inputs to the one-way function; the larger the key, the greater the difference between the efforts necessary to compute the function in the forward and inverse directions (for someone lacking the trap door). For a digital signature to be unbreakable for years, it is necessary to use a trap-door function with inputs large enough that someone without the trap door would need many years to compute the inverse function, but also so that anyone can compute in the forward direction in at most a few minutes.

All practical public-key cryptosystems are based on functions that are believed to be one-way, but have not been proven to be so. This means it is theoretically possible that an algorithm will be discovered that can compute the inverse function easily without a trap door; this would render any cryptosystem based on the function insecure and useless. The possibility of such a breakthrough is discussed in Question 4.7, in the context of the factoring problem.

4.3 *What is the factoring problem?*

Factoring is the act of splitting an integer into a set of smaller integers (factors) which, when multiplied together, form the original integer. Prime factorization requires splitting an integer into factors which are prime numbers; every integer has a unique prime factorization. Multiplying two prime integers together is easy, but as far as we know, factoring the product is much more difficult.

4.4 *What is the significance of factoring in cryptography?*

Factoring is the underlying, presumably hard problem upon which several public-key cryptosystems are based, including RSA. It is believed that factoring a large integer is much more difficult than multiplying integers together to form a larger integer. The importance of factoring is discussed in the context of RSA.

In RSA, the one-way function is modular exponentiation; the private key is the trap door which allows one to invert the one-way function. Factoring the modulus would allow recovery of the trap door; the bottom line is that if anyone besides the possessor of the private key could factor the modulus, he could then decrypt messages and forge signatures. Thus the security of RSA depends on the factoring problem being difficult. Unfortunately, it has not been proven that factoring must be difficult, and there remains a remote possibility that a quick and easy factoring method might be discovered (see Question 4.7). There is also a remote possibility of uncovering the trap door without factoring.

Factoring large numbers takes more time than factoring smaller numbers. Therefore, the size of the key pair in RSA determines how secure an actual use of RSA is; the larger the key size, the longer it would take an attacker to factor the public key, and thus the more resistant to attack is RSA.

4.5 *Has factoring been getting easier?*

Factoring has become easier over the last ten years for two reasons: computer hardware has grown faster, and better factoring algorithms have been developed.

Hardware improvement will continue inexorably, but it is important to realize that *hardware improvements make RSA more secure, not less.* This is because a speed improvement that allows an attacker to factor a number two digits longer than previously will allow a legitimate RSA user to use keys dozens of digits longer than previously. Thus although the hardware improvement does help the attacker, it helps the legitimate user much more. This general rule may fail in the sense that factoring may take place using fast machines of the future, attacking RSA keys of the past; in this scenario, only the attacker gets the advantage of the hardware improvement. This consideration argues for using a larger key size today than one might otherwise consider warranted. It also argues for replacing one's RSA key with a longer key every few years, in order to take advantage of the extra security offered by hardware improvements.

Better factoring algorithms have provided much more help to the attacker than hardware improvements. As RSA and cryptography in general have garnered much attention, so has the factoring problem, and many researchers have been searching for ways to improve the factoring process. They have been at least partially successful; the last several years have seen the discovery of new factoring algorithms and modification of others. This has had the effect of making factoring easier irrespective of the size of the modulus length or speed of the hardware. However, factoring is still a very difficult problem.

Overall, any recent decrease in security due to algorithm improvement can be offset by increasing the key size. In fact, between general computer hardware improvements and special-purpose RSA hardware improvements (see Question 2.3), increases in key size (maintaining speed of RSA operations) have kept pace or exceeded increases in algorithm efficiency, resulting in no net loss of security. As long as hardware continues to improve at a faster rate than that at which the complexity of factoring algorithms declines, the security of RSA will increase, assuming RSA users regularly increase their key size by an appropriate amount. The open question is how much faster factoring algorithms can get. There must be some limit to factoring speed, but no one knows where.

4.6 *What are the best factoring methods in use today?*

Factoring is a very active field of research among mathematicians and computer scientists; the best factoring algorithms are mentioned below with some refer-

ences and their big-O asymptotic efficiency. O notation measures how fast an algorithm is; it gives the number of operations (to order of magnitude) in terms of n, the number to be factored, and p, a prime factor of n.

For textbook treatment of factoring algorithms, see [36], [42], [37], and [11].

Factoring algorithms come in two flavors, special purpose and general purpose; the efficiency of the former depend both on the number to be factored and on the unknown factors, whereas the efficiency of the latter depend only on the number to be factored. General purpose algorithms are the most important ones in the context of cryptographic systems and their security.

The best general purpose algorithm today is the multiple polynomial quadratic sieve (mpqs) [68], which has running time $O(\exp(\sqrt{\ln n \ln \ln n}))$. The mpqs (and some of its variations) is the only general purpose algorithm that has successfully factored numbers greater than 100 digits. A variation due to Lenstra and Manasse [44], known as ppmpqs, has been popular.

Special purpose factoring algorithms include the Pollard rho method [58], with running time $O(\sqrt{p})$, and the Pollard $p-1$ method [59], with running time $O(p')$, where p' is the largest prime factor of $p-1$. Both of these take a number of steps that is exponential in the size of n; they thus take too long for most factoring jobs. The elliptic curve method (ECM) [45] is superior to these; its asymptotic running time is the same as mpqs in the worst case, and somewhat better on average. Even though the ECM depends on the size of factors, in many ways it behaves like a general-purpose algorithm.

The recent number field sieve method [43], also a special-purpose algorithm, is superior to all the other methods, but so far it only works for factoring numbers of a narrow class. It is currently being modified to obtain a general-purpose factoring algorithm [14]. The generalized number field sieve may become the top factoring algorithm within a few years, if it proves to be practical.

Rivest estimates [64] that a 512-bit number would need 2.1 million MIPS-years to factor by the best general purpose factoring algorithm today; a MIPS-year is the amount of computation done by a 1 MIPS (million instructions per second) computer in one year. His calculations can be adapted to estimate the MIPS-years required to factor a number of any length.

A good picture of present-day factoring capability can be obtained by looking at recent results of the RSA Factoring Challenge (see Question 4.8).

4.7 *What are the prospects for theoretical factoring breakthroughs?*

Although factoring is strongly believed to be a difficult mathematical problem, it has not been proved so. Therefore there remains a possibility that an easy factoring algorithm will be discovered. This development, which would render RSA useless, would be highly surprising and the possibility is considered extremely remote by the researchers most actively engaged in factoring research.

Another possibility is that someone will prove that factoring is difficult. This negative breakthrough is probably more likely than the positive breakthrough

discussed above, but would also be unexpected at the current state of theoretical factoring research. This development would guarantee the security of RSA beyond a certain key size.

4.8 *What is the RSA Factoring Challenge?*

RSA Data Security Inc. (RSADSI) administers a factoring contest with quarterly cash prizes. Those who factor numbers listed by RSADSI earn points toward the prizes; factoring smaller numbers earns more points than factoring larger numbers. Send email to `challenge-info@rsa.com` for information.

Results of the contest are useful for those who wish to know the state of the art in factoring. The results show the size of numbers factored, which algorithms are used, and how much time was required to factor each number. Send email to `challenge-info@rsa.com` for information about how to obtain the successful factoring results.

4.9 *What is the discrete log problem?*

The discrete log problem, in its most common formulation, is to find the exponent x in the formula $y = g^x \bmod p$; in other words, it seeks to answer the question, To what power must g be raised in order to obtain y, modulo the prime number p?

Like the factoring problem, the discrete log problem is believed to be difficult and also to be the inverse direction of a one-way function. For this reason, it has been the basis of several public-key cryptosystems, including the ElGamal system and the proposed DSS (see Questions 2.17 and 7.1). The discrete log problem bears the same relation to these systems as factoring does to RSA; in particular, the security of these systems rests on the presumption that discrete logs are difficult to compute.

The discrete log problem has received much attention in recent years from researchers looking for efficient algorithms. For descriptions of today's most efficient algorithms see [42] and [20]. The best discrete log problems have an expected running time of approximately $O(\exp(\sqrt{\ln p \ln \ln p}))$, which is similar to the expected running time of the best general-purpose factoring algorithm. Rivest [64] has analyzed the expected time to solve discrete log both in terms of MIPS-years and money.

4.10 *Which is easier, factoring or discrete log?*

The asymptotic running time of the best discrete log algorithm is approximately the same as for the best general purpose factoring algorithm. Therefore, it requires about as much effort to solve the discrete log problem mod a 512-bit prime as to factor a 512-bit RSA modulus. A recent paper [40] cites experimental evidence that the discrete log problem is slightly harder than factoring: they

suggest that the effort necessary to factor a 110-digit integer is the same as the effort to solve discrete logarithms modulo a 100-digit prime. In practice, this difference is so slight that it should not be a significant consideration when choosing a cryptosystem.

Historically, it has been the case that an algorithmic advance in either problem, factoring or discrete logs, was then applied to the other. This suggests that the degrees of difficulty of both problems are closely linked, and that any breakthrough, either positive or negative, will affect both problems equally.

5 DES

5.1 *What is DES?*

DES is the Data Encryption Standard, an encryption block cipher defined by and endorsed by the U.S. government [50] in 1977 for use within the U.S. It was originally developed at IBM. DES has been extensively studied over the last 15 years and is the most well-known and widely used cryptosystem in the world.

DES is a secret-key, symmetric cryptosystem. When used for communication, both sender and receiver must know the same secret key, which is used both to encrypt and decrypt the message. DES can also be used for single-user encryption, for example, to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography was invented to solve this problem (see Question 1.3).

DES operates on 64-bit blocks with a 56-bit key. It was designed to be implemented in hardware, and is fast. It is very good for bulk encryption, that is, for encrypting a large set of data.

NIST (see Question 8.1) recertifies DES as an official U.S. government encryption standard every five years; DES was last recertified in 1988 [51]. NIST has indicated that it is considering not recertifying DES again [69].

5.2 *Has DES been broken?*

Researchers have been trying to "break" DES for a long time. The obvious method of attack is brute-force exhaustive search of the key space; this takes 2^{54} steps on average. Early on it was suggested [23] that a rich and powerful enemy could build a special-purpose computer capable of breaking DES by exhaustive search in a reasonable amount of time. Later, Hellman [31] showed a time-memory trade-off that allows improvement over exhaustive search if memory space is plentiful, after an exhaustive precomputation. These ideas helped engender doubts about the security of DES. There were also accusations that the NSA intentionally weakened DES. Despite these suspicions, no feasible way to break DES was discovered.

Just recently, however, the first attack on DES that is better than exhaustive search was announced by Eli Biham and Adi Shamir [6], using a new technique known as differential cryptanalysis. This attack requires encryption of 2^{47} chosen plaintexts, i.e., plaintexts chosen by the attacker. Changing keys frequently is not an adequate defense, because the attack tests each possible key as soon as it is generated during the attack; therefore the expected time to success is not affected by key changes (as long as the chosen plaintexts are always encrypted under the current key). Although a theoretical breakthrough, this attack is not practical under normal circumstances because it requires the attacker to have easy access to the DES device in order to encrypt the chosen plaintexts. The attacker also needs a large amount of computing resources, an amount currently available only to powerful organizations; those powerful enough to carry out this attack are probably capable of simple exhaustive search as well.

The consensus is that DES, when used properly, is secure against all but the most powerful enemies. Biham and Shamir have stated that they consider DES secure. It is used extensively in a wide variety of cryptographic systems; most implementations of public-key cryptography include DES at some level.

5.3 *How does one use DES securely?*

One should change DES keys frequently, in order to prevent attacks that require sustained data analysis. In a communications context, one must also provide secure key agreement, that is, find a secure way of communicating the DES key to both sender and receiver. Use of RSA for key management solves both these issues: it generates a different DES key for each message, and it provides secure key management by encrypting the DES key with the receiver's RSA public key. RSA, when used for privacy, can be regarded as a tool for improving the security of DES (or any other bulk encryption cipher).

If one wishes to use DES to encrypt files stored on a hard disk, it is not feasible to frequently change the DES keys, as this would entail decrypting and then re-encrypting all files upon each key change. Instead, one should have a master DES key with which one encrypts the list of DES keys used to encrypt the files; one can then change the master key frequently without much effort.

Another technique for improving security is triple encryption, that is, encrypting each message block under three different DES keys in succession. Triple encryption is equivalent to doubling the key size of DES, and helps prevent decryption by an enemy capable of single-key exhaustive search [47].

Aside from the issues mentioned above, DES can be used for encryption in several different modes. Some are more secure than others. ECB (electronic codebook) mode simply encrypts each 64-bit block of plaintext one after another under the same 56-bit DES key; it is the simplest mode. In CBC (cipher block chaining) mode, each 64-bit plaintext block is XORed with the previous ciphertext block before being encrypted with the DES key. Thus the encryption of each block depends on previous blocks and the same 64-bit plaintext block can

encrypt to different ciphertext depending on its context in the overall message. In CBC mode, the very first plaintext block is XORed with an initialization vector before encryption. CFB (cipher feedback) mode allows one to use DES with block lengths less than 64 bits. Detailed discussion of the various DES modes can be found in [52].

In practice, CBC is the most widely used mode of DES, and is specified in several standards. For additional security, one could use triple encryption with CBC, but since CBC by itself is usually considered secure enough, triple encryption is generally not used.

5.4 *Can DES be exported from the U.S.?*

Export of DES, either in hardware or software, is strictly regulated by the U.S. State Department and the NSA (see Question 1.6). The government rarely approves export of DES, despite the fact that DES is widely available overseas; financial institutions and foreign subsidiaries of U.S. companies are exceptions. RC2 and RC4 (see Question 6.5) are alternatives to DES with special export status designed to ease export approval, at least when used with restricted key size.

5.5 *What are the alternatives to DES?*

When it became apparent that NIST may stop recertifying DES for official use, people started designing alternatives. One is FEAL (Fast Encryption ALgorithm), a cipher for which attacks have been discovered [5], although new versions have been proposed. Another recently proposed cipher [39] seems promising, although it has not yet received sufficient scrutiny to instill full confidence in its security.

Rivest has developed the ciphers RC2 and RC4 (see Question 6.5), which can be made as secure as necessary because they use variable key sizes. Faster than DES, they have the further advantage of special U.S. government status whereby the export approval is expedited and simplified if the key size is limited to 40 bits. For this reason developers looking to export have been adopting RC2 and RC4 as alternatives to DES.

5.6 *Is DES a group?*

It has been frequently asked whether DES encryption is closed under composition; i.e., is encrypting a plaintext under one DES key and then encrypting the result under another key equivalent to a single encryption under a single key? If the answer is yes, it indicates DES to be weaker than if the answer is no.

DES is not a group. Although there has been much speculation and discussion of this issue for years, it was settled only recently [16]. This follows

previously reported experimental evidence to the same effect. For a more complete discussion of the significance of the issue, see [34]. Since DES is not a group, techniques such as triple encryption do in fact increase the security of DES.

6 Miscellaneous

6.1 *What is the legal status of documents signed with digital signatures?*

The purpose of digital signatures is to replace handwritten signatures; ultimately, this means that digital signatures must be as legally binding as handwritten signatures. NIST has stated that its proposed digital signature standard (see Question 7.1) should be capable of "proving to a third party that data was actually signed by the generator of the signature." Furthermore, U.S. federal government purchase orders will be signed by any such standard; this implies that the government will support the legal authority of digital signatures in the courts. Some preliminary legal research has also resulted in the opinion that digital signatures would meet the requirements of legally binding signatures for most purposes, including commercial use as defined in the Uniform Commercial Code (UCC). A GAO (Government Accounting Office) decision request by NIST also opines that digital signatures will meet the legal standards of handwritten signatures [19].

However, since the validity of documents with digital signatures has never been challenged in court, their legal status is not yet well-defined. In order for digital signatures to carry the same authority (or more) as handwritten signatures, they must first be used to sign a legally binding document, such as a contract, and then be challenged by one of the parties. The court would then consider the security of the particular signature scheme and issue a ruling. If this happened several times, lines would be drawn regarding which digital signature methods and how large a key size are required for a digital signature to be legally binding.

Currently, if two people wish to digitally sign a series of contracts, it is recommended that they first sign a paper contract in which they agree for the future to be bound by any contracts digitally signed by them with a given signature method and minimum key size.

Digital signatures have the potential to possess greater legal authority than handwritten signatures. If a ten-page contract is signed by hand on the tenth page, one cannot be sure that the first nine pages have not been altered. If the contract was signed by digital signatures, however, a third party can verify that not one byte of the contract has been altered.

6.2 *What is a hash function? What is a message digest?*

A hash function is a computation that takes a variable size input and returns a string of fixed size, which is called the hash value. If the hash function is one-way, i.e., hard to invert, it is also called a message-digest function, and the result is called a message digest. The idea is that a digest represents concisely the longer message or document from which it was computed; one can think of a digest as a "digital fingerprint" of the larger message. Examples of well-known hash functions are MD4, MD5 (see Question 6.3), and SHS (see Question 6.4).

Although hash functions in general have many uses in computer programs, in cryptography they are used to generate a small string (the digest) that can represent a much larger string (such as a file or message); digital signatures are then computed using the message digest rather than the message itself (see Question 2.14). It is much more efficient to compute a digital signature on a small input like a message digest than on an arbitrarily large input like a message; the hash function is much faster than the signing function. Additionally, a digest can be made public without revealing the contents of the message from which it is derived. This is important in digital timestamping, for example, where one can get a document timestamped while not revealing the document itself to the timestamping service (see Question 3.18).

A hash function used for digital authentication must have certain properties that make it secure enough for cryptographic use. Specifically, it must be infeasible to find a message which hashes to a given value and it must be infeasible to find two distinct messages which hash to the same value. The ability to find a message hashing to a given value would enable an attacker to substitute a fake message for a real message that was signed; a digital signature will verify correctly for any message with the right hash value. It would also enable someone to disown a message he signed by claiming that he actually signed a different message hashing to the same value. The ability to find two distinct messages hashing to the same value could enable an attack whereby someone is tricked into signing one among a series of look-alike messages which hashes to the same value as another message with a quite different meaning.

A proposed hash function mush create digests of a minimum length in order to prevent attacks based on exhaustive search. For example, if a hash function produces 100-bit numbers, exhaustive search would take 2^{100} attempts on average to match a given value, and approximately 2^{50} attempts on average to find two inputs producing the same digest. Of course, just being long enough doesn't guarantee the security of a hash function.

6.3 *What are MD2, MD4 and MD5?*

MD2, MD4 and MD5 (MD stands for Message Digest) are widely used hash functions designed by Ron Rivest specifically for cryptographic use. They produce 128-bit digests and are believed secure against attack, i.e., it is conjectured

that an effort on the order of 2^{128} is necessary to find a message hashing to a given digest and that an effort of 2^{64} is necessary to find two messages hashing to the same value.

MD2 is the slowest of the three. MD4 [63] is the fastest and is part of the SNMP (Secure Network Management Protocol) Internet standard. MD5 [65] has been described by Rivest as "MD4 with safety belts": it has a more conservative design than MD4 and can be considered more secure, but at a cost of being approximately 33% slower. Currently, MD5 is the most often recommended hash algorithm for digital signatures. The Internet Privacy-Enhanced Mail standard (see Question 6.6) lists the MD algorithms as endorsed message digest functions. There is also an extension of MD4 which produces a 256-bit hash value [63].

No serious weaknesses have been discovered in any of the three MD algorithms. An attempt by Berson to apply differential cryptanalysis to MD5 [4] failed to reveal any feasible attack, despite some interesting theoretical analysis. Differential cryptanalysis has in fact found weaknesses in two other proposed hash functions, N-hash and Snefru.

A digital signature system can be broken by attacking either the difficult cryptographic problem used for signing or the hash function used to create the message digests. When choosing an authentication system, it is generally a good idea to choose a signature method and a hash function that require comparable efforts to break; any extra security in one of the two components is wasted, since attacks will be directed at the weaker component. The MD hash functions can be attacked with 2^{64} operations, which is comparable to the effort necessary to break 512-bit RSA, although the attack on MD is harder in practice, since it requires 2^{64} memory units and the ability to trick the attackee into signing a message of your choice. MD5 is a good choice when using RSA with a 512-bit modulus. However, those with greater security needs, such as certifying authorities, should use a longer modulus and a hash function that produces a longer message digest; either SHS (160-bit digest) or a modified version of MD4 that produces a 256-bit digest [63] would suffice.

The MD algorithms are available for unrestricted use. Details of MD4 and MD5 with sample C code are available as Internet RFCs (Request for Comments) 1320 and 1321 respectively. They can be obtained via anonymous ftp at `ftp.nisc.sri.com` in the `rfc` directory.

6.4 *What is SHS?*

The Secure Hash Standard (SHS) [55] is a hash function proposed by NIST (see Question 8.1); it is designed for use with its proposed Digital Signature Standard (see Question 7.1). It produces a 160-bit hash value from a variable size input. SHS is structurally similar to MD4 and MD5. It is 25% slower than MD5 but may be more secure, because it produces message digests that are 25% longer than the MD functions. SHS has not yet been formally adopted by

NIST as an official government standard.

6.5 *What are RC2 and RC4?*

RC2 and RC4 are variable-key-size cipher functions designed by Ron Rivest and meant for fast bulk encryption. They are alternatives to DES (see Question 5.1) that are as fast or faster than DES and are capable of being more secure than DES because of their ability to use long key sizes; they can also be less secure than DES if short key sizes are used.

RC2 is a variable-key-size symmetric block cipher and can serve as a drop-in replacement for DES, for example in export versions of products otherwise using DES. RC2 is approximately twice as fast as DES, at least in software. RC4 is a variable-key-size symmetric stream cipher and is 10 to 100 times as fast as DES. Both RC2 and RC4 are very compact in terms of code size. Their speeds are independent of key size. To date, they have not been implemented in hardware.

A recent agreement between the Software Publishers Association (SPA) and the U.S. government gives RC2 and RC4 special status by which the export approval process is much simpler and quicker than the general cryptographic export process. However, to qualify for the quick export approval a product must limit the RC2 and RC4 key sizes to 40 bits; this maximum length may be gradually increased over the coming years. RC2 and RC4 have been widely used by developers who want to export their products. DES is almost never approved for export.

RC2 and RC4 are proprietary algorithms of RSA Data Security Inc. Details about them have not been published (including by patenting) in order to maintain their special export status.

6.6 *What is PEM?*

PEM is the Internet Privacy-Enhanced Mail standard, designed, proposed, but not yet officially adopted, by the Internet Activities Board in order to provide secure electronic mail over the Internet. It is designed to be compatible with current Internet email formats, although it requires new email software. PEM includes encryption, authentication, and key management. It is an inclusive standard, and allows use of both public-key and secret-key cryptosystems. Multiple cryptographic tools are supported; for each mail message, the specific hash function, encryption algorithm, signature algorithm, and so on are specified in the header. PEM names certain cryptographic algorithms as acceptably secure; others may be added later. PEM also supports the use of certificates, endorsing the X.509 standard for certificate structure. If the message itself is encrypted, DES in CBC mode is always used and the mail header gives information regarding the method by which the DES session key was encrypted, either RSA or DES. Later versions of PEM will include other encryption algorithms. The use of certificates and the other key management structures is optional.

The details of PEM can be found in an article by Bishop [7] and in RFCs (Requests for Comment) 1113 through 1115; some details have been changed since those publications, but so far the changes exist only in unpublished draft form. PEM is likely to be officially adopted by the Internet Activities Board within six months; after that, free implementations will be made available.

6.7 *What is PKCS?*

PKCS (Public-Key Cryptography Standards) is a set of standards for implementation of public-key cryptography. It has been issued by RSA Data Security Inc. in cooperation with a computer industry consortium, including Apple, Microsoft, DEC, Lotus, Sun and MIT. PKCS has been cited by the OIW (OSI Implementors' Workshop) as a method for implementation of OSI standards. PKCS is compatible with PEM (see Question 6.6) but extends beyond PEM. For example, where PEM can only handle ASCII data, PKCS is intended for binary data as well. PKCS is also compatible with the CCITT X.509 standard and provides implementation details about RSA encryption and authentication that were left out of X.509.

PKCS includes both algorithms-specific and algorithm-independent implementation standards. Specific algorithms supported include RSA, DES (CBC mode), and Diffie-Hellman key exchange. It also details algorithm-independent syntax for digital signatures, digital envelopes (for encryption), and certificates; this enables someone implementing any other cryptographic algorithms to conform to a standard syntax and thus preserve interoperability.

Documents detailing the PKCS standards can be obtained by anonymous ftp to `rsa.com` or by email to `pkcs@rsa.com`.

6.8 *What is RSAREF?*

RSAREF is a collection of cryptographic routines in portable C source code available at no charge from RSA Laboratories, a division of RSA Data Security, Inc. It includes RSA, MD2, MD5, and DES. It includes both low-level subroutines, such as RSA exponentiation, and high-level cryptographic functions, such as digital signature verification. The arithmetic routines can handle multiple-precision integers, and the RSA algorithm routines can handle variable key sizes. RSAREF is fully compatible with the PEM and PKCS standards.

RSAREF is available to citizens of the U.S. or Canada and to permanent residents of the U.S. It can be used in personal, non-commercial applications. It cannot be used commercially and it cannot be sent outside the U.S. or Canada. The RSAREF license contains more details on the usage allowed and disallowed. RSAREF is available through the Internet by sending email to `rsaref@rsa.com`.

7 DSS

7.1 *What is DSS?*

DSS is the proposed Digital Signature Standard, which specifies a Digital Signature Algorithm (DSA). It was selected by NIST (see Question 8.1) to be the digital authentication standard of the U.S. government; whether the government should in fact adopt it as the official standard is still under debate. DSS was chosen after study by NIST in cooperation with various government security and law-enforcement agencies, most prominently the NSA (see Question 8.4). Private industry was not involved in the selection of DSS; since the selection, however, industry has been able to make public comments to NIST and Congress regarding DSS.

DSS is based on the discrete log problem (see Question 4.9) and derives from cryptosystems proposed by Schnorr [67] and ElGamal [25]. It is for authentication only and cannot be used for key exchange or encryption. For a detailed description of DSS, see [54] or [53].

DSS has been looked upon unfavorably by the computer industry, which had hoped the government would choose the RSA algorithm as the official standard; RSA is the most widely used authentication algorithm and is a de facto standard in the private sector. Several recent articles in the press discuss the industry dissatisfaction with DSS; an article by Messmer [48] is one example. Criticism of DSS has focused on a few main issues: it lacks key exchange capability; the key size of DSS, 512 bits, is not variable and/or is too small; there is a lack of guidelines for secure implementation; the underlying cryptosystem is too recent and has been subject to too little scrutiny to be confident of its strength; the existence of two authentication standards will cause hardship to computer hardware and software vendors, who have already standardized on RSA; and that the process by which NIST chose DSS was too secretive and arbitrary, with too much influence wielded by NSA. A more detailed discussion of these criticisms can be found in [53], and a detailed response by NIST to the criticisms can be found in [70].

In the DSS system, signature generation is faster than signature verification; in the RSA system, signature verification is much faster than signature generation (as long as the public and private exponents are chosen to have this property, which is the usual case). NIST claims that it is an advantage that signing is faster, but many people in cryptography think that verification should be faster.

7.2 *Is DSS secure?*

DSS has been greeted with suspicion by many in industry and academia. Their most serious criticisms involve the security of DSS.

DSS was proposed with a fixed 512-bit key size. Although probably secure

enough for most ordinary uses, 512 bits is not secure enough for those with high security needs, such as certifying authorities. Furthermore, in a few years 512 bits may not be secure enough for ordinary needs. What is needed is an ability to handle variable key sizes, so that every user can choose a key size appropriate to his needs. In response to this criticism, NIST has announced that DSS will be revised to allow key sizes up to 1024 bits.

DSS has not been around long enough to withstand attempts to break it; although the discrete log problem is old, the particular form of the problem used in DSS was first proposed for cryptographic use in 1989 [67] and has not received much study. This may be the most powerful argument against the security of DSS. Any new cryptosystem could have serious flaws that are only discovered after a couple of years of scrutiny by cryptographers. Indeed this has happened many times in the past; see [13] for details. RSA has withstood 15 years of vigorous examination for weaknesses. In the absence of mathematical proofs of security, nothing builds confidence in a cryptosystem like sustained attempts to crack it. Although DSS may well turn out to be a strong cryptosystem, its relatively short history will leave doubts for years to come.

Some researchers raised alarm about the existence of "trapdoor" primes in DSS, which could enable a key to be easily broken. These trapdoor primes are relatively rare however, and can be avoided if each person generates his own key. If keys are generated by a central authority, a procedure can followed by which a DSS user can be confident that he was not intentionally given a weak prime [70].

7.3 *Is use of DSS covered by any patents?*

Whether use of DSS infringes any existing patents is a matter of current dispute. NIST claims that DSS is not covered by any existing patents and thus that any private entity can use DSS without licensing or royalty fees; indeed, this was one of the criteria used by NIST when choosing DSS. However, authors of at least three U.S. patents claim that DSS infringes upon their work; the patents are 4,200,770, 4,218,582, and 4,995,082. The government has filed for a patent for DSS; the inventor is a mathematician who works for the NSA. NIST does not plan to charge for licensing its patent.

In the debate over DSS vs. RSA, the claim by NIST that DSS can be used without paying patent licenses or royalties has been the only clear advantage of DSS over RSA. RSA can be used without charge by the U.S. government and is not patented outside North America, but it must be licensed by private industry in the U.S. If it turns out that DSS infringes previous patents after all, acceptance of DSS will be seriously damaged. The question of patent infringement will eventually be settled in the courts in a few years.

7.4 *What is the current status of DSS?*

After NIST issued its proposal to DSS in August 1991, there was a period in which comments from the public were solicited. Most comments were negative. NIST is currently in the process of reviewing and revising its proposal, in light of the comments. A revised DSS will be released and new comments may be solicited. Later, it may be issued as a FIPS and become the official U.S. government standard.

In March 1992, the Computer Security and Privacy Advisory Board voted unanimously that NIST should postpone decision on DSS and sponsor a public debate on DSS and other cryptography policy issues. The board is an official advisory body to NIST; its twelve members are drawn from both the U.S. government and private industry. It said that a national policy review is the only way to resolve the conflicts between competing interests. Government security and law-enforcement agencies want the use of cryptography to be restricted, whereas other government agencies and private industry want cryptographic tools to become more readily available.

A NIST official standard must be used by the U.S. government agencies in almost all cases, and thus would be used by companies doing business with the government as well. Use by anyone outside the government is voluntary. See Question 2.21 for a discussion of cryptography in the presence of two standards, DSS and RSA.

8 NIST and NSA

8.1 *What is NIST?*

NIST is an acronym for the National Institute of Standards and Technology, a division of the U.S. Department of Commerce; it was formerly known as the National Bureau of Standards (NBS). Through its Computer Systems Laboratory (CSL) it aims to promote open systems and interoperability that will spur development of computer-based economic activity. It issues standards and guidelines that it hopes will be adopted by all computer systems in the U.S.; for example, it has issued codes for every county in the U.S. It also sponsors workshops and seminars; it has recently sponsored meetings of the OSI Implementors' Workshop (OIW). Official standards are published as FIPS (Federal Information Processing Standards) publications.

In 1987 Congress passed the Computer Security Act, which gave NIST a mandate to define standards for ensuring the security of sensitive but unclassified information in government computer systems. It authorized NIST to work with other government agencies and private industry in evaluating proposed technology standards.

8.2 *What role does NIST play in cryptography?*

NIST issues standards for cr