Answers To

Frequently Asked Questions About Today's Cryptography

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

Version 2.0, draft 2f.
Last update: September 20, 1993.
Copyright © 1993 RSA Laboratories, a division of RSA Data Security, Inc.
All rights reserved.
Part #002-903002-200-02f-000

Contents

1 General **1**

1.1 *What is encryption?* 1
1.2 *What is authentication? What is a digital signature?* 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.?* 4
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 *How fast is RSA?* 7
2.4 *How much extra message length is caused by using RSA?* 7
2.5 *What would it take to break RSA?* 7
2.6 *Are strong primes necessary in RSA?* 8
2.7 *How large a modulus (key) should be used in RSA?* 9
2.8 *How large should the primes be?* 10
2.9 *How does one find random numbers for keys?* 10
2.10 *What if users of RSA run out of distinct primes?* 11
2.11 *How do you know if a number is prime?* 11
2.12 *How is RSA used for encryption in practice?* 11
2.13 *How is RSA used for authentication in practice?* 11
2.14 *Does RSA help detect altered documents and transmission errors?* 12
2.15 *What are alternatives to RSA?* 12
2.16 *Is RSA currently in use today?* 13
2.17 *Is RSA an official standard today?* 14
2.18 *Is RSA a de facto standard? Why is a de facto standard important?* 14
2.19 *Is RSA patented?* 15
2.20 *Can RSA be exported from the U.S.?* 15

3 Key Management **15**

3.1 *What key management issues are involved in public-key cryptography?* 15
3.2 *Who needs a key?* 16
3.3 *How does one get a key pair?* 16
3.4 *Should a public key or private key be shared among users?* 17
3.5 *What are certificates?* 17
3.6 *How are certificates used?* 18
3.7 *Who issues certificates and how?* 18
3.8 *What is a CSU, or, How do certifying authorities store their private keys?* 19

3.9 *Are certifying authorities susceptible to attack?* 20
3.10 *What if the certifying authority's key is lost or compromised?* . 21
3.11 *What are Certificate Revocation Lists (CRLs)?* 21
3.12 *What happens when a key expires?* 22
3.13 *What happens if I lose my private key?* 23
3.14 *What happens if my private key is compromised?* 23
3.15 *How should I store my private key?* 23
3.16 *How do I find someone else's public key?* 24
3.17 *How can signatures remain valid beyond the expiration dates of their keys, or, How do you verify a 20-year-old signature?* 24
3.18 *What is a digital time-stamping service?* 25

4 Factoring and Discrete Log **26**
4.1 *What is a one-way function?* 26
4.2 *What is the significance of one-way functions for cryptography?* . 26
4.3 *What is the factoring problem?* 27
4.4 *What is the significance of factoring in cryptography?* 27
4.5 *Has factoring been getting easier?* 27
4.6 *What are the best factoring methods in use today?* 28
4.7 *What are the prospects for theoretical factoring breakthroughs?* . . 29
4.8 *What is the RSA Factoring Challenge?* 29
4.9 *What is the discrete log problem?* 30
4.10 *Which is easier, factoring or discrete log?* 30

5 DES **31**
5.1 *What is DES?* 31
5.2 *Has DES been broken?* 31
5.3 *How does one use DES securely?* 32
5.4 *Can DES be exported from the U.S.?* 33
5.5 *What are the alternatives to DES?* 33
5.6 *Is DES a group?* 33

6 Capstone, Clipper, and DSS **34**
6.1 *What is Capstone?* 34
6.2 *What is Clipper?* 34
6.3 *How does the Clipper chip work?* 35
6.4 *Who are the escrow agencies?* 35
6.5 *What is Skipjack?* 36
6.6 *Why is Clipper controversial?* 36
6.7 *What is the current status of Clipper?* 37
6.8 *What is DSS?* 37
6.9 *Is DSS secure?* 38
6.10 *Is use of DSS covered by any patents?* 38
6.11 *What is the current status of DSS?* 39

7 NIST and NSA **39**
7.1 *What is NIST?* 39
7.2 *What role does NIST play in cryptography?* 39
7.3 *What is the NSA?* 40
7.4 *What role does the NSA play in commercial cryptography?* 40

8 Miscellaneous **41**
8.1 *What is the legal status of documents signed with digital signatures?* 41
8.2 *What is a hash function? What is a message digest?* 42
8.3 *What are MD2, MD4 and MD5?* 43
8.4 *What is SHS?* 43
8.5 *What is Kerberos?* 44
8.6 *What are RC2 and RC4?* 44
8.7 *What is PEM?* 45
8.8 *What is RIPEM?* 45
8.9 *What is PKCS?* 45
8.10 *What is RSAREF?* 46

9 Acknowledgements **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 information hidden from anyone for whom it is not intended, even those who can see the encrypted data. For example, one may wish to encrypt files on a hard disk to prevent an intruder from reading them.

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, called the ciphertext, is sent to Bob. 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. In a symmetric cryptosystem, a single key serves as both the encryption and decryption keys.

Cryptography has been around for millennia; see Kahn [37] for a good history of cryptography; see Rivest [69] and Brassard [10] for an introduction to modern cryptography.

1.2 *What is authentication? What is a digital signature?*

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 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 piece of data asserting that a named person 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 digital signature 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. 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 being communicated. 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 [29] in order to solve the key management problem. In the new system, each person gets a pair of keys, called the public key and the private key. Each person's public key is published while the private key is kept 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 some communications channel to be secure against eavesdropping or betrayal. Anyone can send a confidential message just using public information, but it can only be decrypted with a private key that is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used for authentication (digital signatures) as well as for privacy (encryption).

Here's how it works for encryption: when Alice wishes to send a message to Bob, she looks up Bob's public key in a directory, uses it to encrypt the 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 the 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 private key and the message itself; the output is called the digital 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 [27].

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 private keys do not ever need to be transmitted or revealed to anyone. In a secret-key system, by contrast, 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 then repudiate a previously signed message by claiming that the shared secret was somehow compromised by one of the parties sharing the secret. For example, the Kerberos secret-key authentication system [79] involves a central database that keeps copies of the secret keys of all users; a Kerberos-authenticated message would most likely not be held legally binding, since an attack on the database would allow widespread forgery. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

Furthermore, digitally signed messages can be proved authentic to a third party, such as a judge, thus allowing such messages to be legally binding. Secret-key authentication systems such as Kerberos were designed to authenticate access to network resources, rather than to authenticate documents, a task which is better achieved via digital signatures.

A disadvantage of using public-key cryptography for encryption is speed: there are popular secret-key encryption methods which are significantly faster than any currently available public-key encryption method. But public-key cryptography can share the burden with secret-key cryptography to get the best of both worlds.

For encryption, the best solution is to combine public- and secret-key systems in order 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 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 2.12 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 [29]; 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 encryption 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 (see Question 5.1). The basic ideas of public-key cryptography are contained in U.S. Patent 4,200,770, by M. Hellman, W. Diffie, and R. Merkle, issued 4/29/80 and in U.S. Patent 4,218,582, by M. Hellman and R. Merkle, issued 8/19/80; similar patents have been issued throughout the world. The exclusive licensing rights to both patents are held by Public Key Partners (PKP), of Sunnyvale, California, which also holds the rights to the RSA patent (see Question 2.19). Usually all of these public-key patents are licensed together.

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.

Some patent applications for cryptosystems have been blocked by intervention by the NSA (see Question 7.3) 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 [46] for some recent cases related to cryptography.

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 basic level (not publicly stated).

Under current regulations, a vendor seeking to export a product using cryptography first submits an request to the State Department's Defense Trade Control office. Export jurisdiction may then be passed to the Department of Commerce, whose export procedures are generally simple and efficient. If jurisdiction remains with the State Department, further review, perhaps lengthy, is required before export is either approved or denied; the National Security Agency (NSA, see Question 7.3) may become directly involved at this point. The details of the export approval process change frequently.

The NSA has de facto control over export of cryptographic products. The State Department will not grant a license without NSA approval and routinely grants licenses whenever NSA does approve. Therefore, the policy 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 cannot be easily

modified for 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 for 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, as many software and hardware vendors consider current export regulations overly restrictive and burdensome. 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 8.6), when the key size is limited. Also, export policy is less restrictive for foreign subsidiaries and overseas offices of U.S. companies.

In March 1992, the Computer Security and Privacy Advisory Board voted unanimously to recommend a national review of cryptography policy, including export policy. The Board is an official advisory board to NIST (see Question 7.1) 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 traditionally been decided solely by agencies concerned with national security, without much input from those who wish to encourage commerce in cryptography. U.S. export policy may undergo significant change in 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 [74]. 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.

It is difficult (presumably) to obtain the private key d from the public key (n, e). If one could factor n into p and q, however, then one could obtain the private key d. Thus the entire security of RSA is predicated on the assumption that factoring is difficult; an easy factoring method would "break" RSA (see Questions 2.5 and 4.4).

Here is how RSA can be used for privacy and authentication (in practice, actual use is slightly different; see Questions 2.12 and 2.13):

RSA privacy (encryption): 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 decrypt, Bob also exponentiates: $m = c^d \bmod n$, and recovers the original message m; the relationship between e and d ensures that Bob correctly recovers m. Since only Bob knows d, only Bob can decrypt.

RSA authentication: suppose Alice wants to send a signed document m to Bob. Alice creates a digital 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 or her own private key. Anyone can send an encrypted message or verify a signed message, using only public keys, but only someone in possession of the correct private key can decrypt or sign a message.

2.2 *Why use RSA rather than DES?*

RSA is not an alternative or replacement for DES; rather it supplements DES (or any other fast 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 encrypting messages, 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. This protocol is known as an RSA digital envelope.

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 (see Question 2.3).

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 usually not necessary in a single-user environment; for example, if you want to keep your personal files encrypted, just do so with DES using, say, your personal password as the DES key. RSA, and public-key cryptography in general, is best suited for a multi-user environment. Also, any system in which digital signatures are desired needs RSA or some other public-key system.

2.3 *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 exponent. This makes encryption faster than decryption and verification faster than signing. Algorithmically, public-key operations take $O(k^2)$ steps, private key operations take $O(k^3)$ steps, and key generation takes $O(k^4)$ steps, where k is the number of bits in the modulus; O-notation refers to the an upper bound on the asymptotic running time of an algorithm [22].

There are many commercially available hardware implementations of RSA, and there are frequent announcements of newer and faster chips. The fastest current RSA chip [76] has a throughput greater than 600 Kbits per second with a 512-bit modulus, implying that it performs over 1000 RSA private-key operations per second. It is expected that RSA speeds will reach 1 Mbit/second within a year or so.

By comparison, DES is much faster than RSA. In software, DES is generally at least 100 times as fast as RSA. In hardware, DES is between 1,000 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.

2.4 *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, usually 64 bits, since RSA is usually combined with a secret-key block cipher such as DES (see Question 2.12). Encrypting the DES key takes as many additional bits as the size of the RSA modulus.

For authentication, an RSA digital signature is appended to a document. An RSA signature, including information such as the name of the signer, is typically a few hundred bytes long. One or more certificates (see Question 3.5) may be included as well; certificates can be used in conjunction with any digital signature method. A typical RSA certificate is a few hundred bytes long.

2.5 *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 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.

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 this guess with the public key of the recipient; by comparison with the actual ciphertext, the attacker knows whether or not the guess was correct. This attack can be thwarted by appending some random bits to the message. Another single-message attack can occur if someone sends the same message m to three others, who each have public exponent $e = 3$. An attacker who knows this and sees the three messages will be able to recover the message m; this attack and ways to prevent it are discussed by Hastad [35]. There are also some "chosen ciphertext" attacks, in which the attacker creates some ciphertext and gets to see the corresponding plaintext, perhaps by tricking a legitimate user into decrypting a fake message; Davida [23] gives some examples.

Of course, there are also 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.

2.6 *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; such 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.

However, recent advances in factoring (see Question 4.6) appear to have obviated the advantage of 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 significantly increase resistance to attacks. So for now the answer is negative: strong primes are not necessary when 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.

2.7 *How large a modulus (key) should be used in RSA?*

The best size for an RSA modulus depends on one's 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, first, of one's security needs, such as the value of the protected data and how long it needs to be protected, and, second, of how powerful one's potential enemy is. It is also possible that a larger key size will allow a digitally signed document to be valid for a longer time; see Question 3.17.

A good analysis of the security obtained by a given modulus length is given by Rivest [72], in the context of discrete logarithms modulo a prime, but it applies to RSA as well. Rivest's estimates imply that a 512-bit modulus can be factored with an $8.2 million effort, less in the future. It may therefore be advisable to use a longer modulus, perhaps 768 bits in length. Those with extremely valuable data (or large potential damage from digital forgery) may want to use a still longer modulus. A certifying authority (see Question 3.5) might use a modulus of length 1000 bits or more, because the validity of so many other key pairs depends on the security of the one central key.

The key of an individual user will expire after a certain time, say, two years (see Question 3.12). Upon expiration, the user will generate a new key which should be at least 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 RSA moduli, may 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 small chance that the attacker may get lucky and factor it quickly.

As for the slowdown caused by increasing the key size (see Question 2.3), doubling the modulus length would, on average, increase the time required for public-key operations (encryption and signature verification) by a factor of 4, and increase the time taken by private key operations (decryption and signing) by a factor of 8. The reason that public-key operations are affected less than private-key operations is that the public exponent can remain fixed when the modulus is increased, whereas the private exponent increases proportionally. Key generation time would increase by a factor of 16 upon doubling the modulus, but this is a relatively infrequent operation for most users.

2.8 *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. Thus if one chooses to use a 512-bit modulus, the primes should each have length approximately 256 bits.

2.9 *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 8.2) 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 [41] 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.

2.10 *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 known universe.

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

It is generally recommended to use probabilistic primality testing, which is much quicker than actually proving a number prime. One can use a probabilistic test that decides if a number is prime with probability of error less than 2^{-100}. For further discussion of some primality testing algorithms, see the papers in the bibliography of [5]. For some empirical results on the reliability of simple primality tests see Rivest [70]; 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 Damgård [9].

2.12 *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-encrypted message and the RSA-encrypted 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.

2.13 *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 8.2) to create a message digest, which serves as a "digital fingerprint" of the message. She then encrypts the message digest with her RSA private key; this is the 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 after it was signed, and he rejects the message. Note that for authentication, the roles of the public and

private keys are converse to their roles in encryption, where the public key is used to encrypt and the private key to decrypt.

In practice, the public exponent is usually much smaller than the private exponent; this means that the 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 either were feasible, an intruder could attach a false message onto Alice's signature. Hash functions such as MD4 and MD5 (see Question 8.3) have been designed specifically to have the property that finding a match is infeasible, and are therefore considered suitable for use in cryptography.

One or more certificates (see Question 3.5) may accompany a digital signature. A certificate is a signed document attesting to the identity and public key of the person signing the message. Its purpose is to prevent someone from impersonating someone else, using a phony key pair. If a certificate is present, the recipient (or a third party) can check the authenticity of the public key, assuming the certifier's public key is itself trusted.

2.14 *Does RSA help detect altered documents and 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 (see Question 8.2) 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. The slightest change in a signed document will cause the digital signature verification process to fail. Thus, RSA authentication allows people to check the integrity of signed documents. Of course, if a signature verification fails, it may be unclear whether there was an attempted forgery or simply a transmission error.

2.15 *What are alternatives to RSA?*

Many other public-key cryptosystems have been proposed, as a look through the proceedings of the annual Crypto and Eurocrypt conferences quickly reveals. A mathematical problem called the knapsack problem was the basis for several systems [52], but these have lost favor because several versions were broken. Another system, designed by ElGamal [30], is based on the discrete logarithm problem. The ElGamal system was, in part, the basis for several later signature methods, including one by Schnorr [75], which in turn was the basis for DSS, the digital signature standard proposed by NIST (see Question 6.8). Because of the NIST proposal, the relative merits of these signature systems versus RSA signatures has received a lot of attention; see [57] 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.

In 1976, before RSA, Diffie and Hellman [29] proposed a system for key exchange only; it permits secure exchange of keys in an otherwise conventional secret-key system. This system is in use today.

Cryptosystems based on mathematical operations on elliptic curves have also been proposed [43, 56], as have cryptosystems based on discrete exponentiation in the finite field $GF(2^n)$. The latter are very fast in hardware; however, doubts have been raised about their security because the underlying problem may be easier to solve than factoring [64, 34]. There are also some probabilistic encryption methods [8, 32], which have the attraction of being resistant to a guessed ciphertext attack (see Question 2.5), but at a cost of data expansion. In probabilistic encryption, the same plaintext encrypted twice under the same key will give, with high probability, two different ciphertexts.

For digital signatures, Rabin [68] 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 a chosen message attack, however, in which the attacker tricks the user into signing messages of a special form. Another signature scheme, by Fiat and Shamir [31], 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 [17] for references. A system is "equivalent to factoring" if recovering the private key is provably as hard as factoring; forgery may be easier than factoring in some of the systems.

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.

2.16 *Is RSA currently in use today?*

The use of RSA is undergoing a period of rapid expansion and may become ubiquitous within a few years. It is currently used in a wide variety of products, platforms and industries around the world. It is found in many commercial software products and planned for many more. RSA is built into current or planned operating systems by Microsoft, Apple, Sun, and Novell. In hardware, RSA can be found in secure telephones, on Ethernet network cards, and on smart cards. RSA is also used internally in many institutions, including branches of the U.S. government, major corporations, national laboratories, and universities.

Adoption of RSA seems to be proceeding more quickly for authentication (digital signatures) than for privacy (encryption), perhaps in part because products for authentication are easier to export than those for privacy (see Question 1.6).

2.17 *Is RSA an official standard today?*

RSA is part of many official standards worldwide. The ISO (International Standards Organization) 9796 standard lists RSA as a compatible cryptographic algorithm, as does the Consultative Committee in International Telegraphy and Telephony (CCITT) X.509 security standard. RSA is part of the Society for Worldwide Interbank Financial Telecommunications (SWIFT) standard, the French financial industry's ETEBAC 5 standard, and the ANSI X9.31 draft standard for the U.S. banking industry. The Australian key management standard, AS2805.6.5.3, also specifies RSA.

RSA is found in Internet's proposed PEM (Privacy Enhanced Mail) standard (see Question 8.7) and the PKCS standard for the software industry (see Question 8.9). The OSI Implementors' Workshop (OIW) has issued implementers' agreements 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. See [38] for a more comprehensive survey of cryptography standards.

2.18 *Is RSA a de facto standard? Why is a de facto standard important?*

RSA is the most widely used public-key cryptosystem today and has often been called a de facto standard. 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 everywhere 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 evolve completely into a society based on electronic transactions. Digital signatures are the exact tool necessary to convert the most essential paper-based documents to digital electronic media. Digital signatures makes it possible, for example, to have leases, wills, passports, 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 an accepted standard for digital signatures.

2.19 *Is RSA patented?*

RSA is patented under U.S. Patent 4,405,829, issued 9/20/83 and held 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 for special circumstances. 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 partial government funding. RSA is not patented outside North America.

In North America, a license is needed to "make, use or sell" RSA. However, PKP usually allows free non-commercial use of RSA, with written permission, for personal, academic or intellectual reasons. Furthermore, 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, improved and redistributed non-commercially (see Question 8.10).

2.20 *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 used 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 obtain securely a key pair suited to their efficiency and security needs. There must be a way to look up other people's public keys and to publicize one's own key. Users must have confidence in the legitimacy of others' public keys; otherwise an intruder can either change public keys listed in a directory, or impersonate another user. Certificates are used for this purpose. Certificates must be unforgeable, obtainable in a secure manner, and processed 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, so that they will no longer encrypt messages under the invalid public key nor accept messages signed with the invalid private key. Users must be able to store their private keys securely, so that no intruder can find it, yet the keys must be 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. Other entities will also have keys, including electronic entities such as modems, workstations, and printers, as well as organizational entities such as a corporate department, a hotel registration desk, or a university registrar's office.

3.3 *How does one get a key pair?*

Each user should generate his or her own key pair. It may be tempting within an organization to have a single site that generates keys for all members who request one, but this is a security risk because it involves the transmission of private keys over a network as well as catastrophic consequences if an attacker infiltrates the key-generation site. Each node on a network should be capable of local key generation, so that private keys are never transmitted and no external key source need be trusted. Of course, the local key generation software must itself be trustworthy. Secret-key authentication systems, such as Kerberos, often do not allow local key generation but instead use a central server to generate keys.

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, each person should have a unique modulus and private exponent, i.e., a unique private key. The public exponent, on the other hand, can be common to a group of users without security being compromised. Some public exponents in common use today are 3 and $2^{16} + 1$; because these numbers 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 with a common modulus than if every key has a distinct modulus. Thus one should be very cautious about using a common modulus; if a common modulus is chosen, it should be very large.

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 help prevent someone from using a phony key to impersonate someone else.

In their simplest form, certificates contain a public key and a name. As commonly used, they also contain the expiration date of the key, the name of the certifying authority that issued the certificate, the serial number of the certificate, and perhaps other information. Most importantly, it contains the digital signature of the certificate issuer. The most widely accepted format for certificates is defined by the CCITT X.509 international standard [19]; 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 8.9), and the PEM standard (see Question 8.7). A detailed discussion of certificate format can also be found in Kent [40].

A 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?*

A certificate is displayed in order to generate confidence in the legitimacy of a public key. Someone verifying a signature can also verify the signer's certificate, to insure that no forgery or false representation has occurred. These steps can be performed with greater or lesser rigor depending on the context.

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 using the certifying authority's public key and, now confident of the public key of the 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, and to verify, 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 or certificate verifications 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 8.9), every signature points to a certificate that validates the public key of the signer. Specifically, each signature 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 vouch 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. In order to prevent forged certificates, the CA's public key must be trustworthy: 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. The latter solution gives rise to hierarchies of CAs.

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 identification and takes any other steps necessary to assure itself that the request really did come from Alice, and then 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 present this certificate chain whenever desired in order to demonstrate the legitimacy of her public key.

Since the CA must check for proper identification, organizations will find it convenient to act as a CA for its own members and employees. There will also be CAs that issue certificates to unaffiliated individuals.

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 should 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 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.

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. One way to achieve the desired security is to store the key in a tamperproof box; such a box is called a Certificate Signing Unit, or CSU. The CSU would, preferably, destroy its contents if ever opened, and be shielded 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 found in some current implementations. The CSU is 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.

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.

Bolt, Beranek, and Newman (BBN) currently sells a CSU, and RSA Data Security sells a full-fledged certificate issuing system built around the BBN CSU.

3.9 *Are certifying authorities susceptible to attack?*

One can think of many attacks aimed at the certifying authority, which must be prepared against them.

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 then 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, i.e., it must require sufficient evidence that it is actually Alice who is requesting the certificate. The CA may, for example, require Alice to appear in person and show a birth certificate. Some CAs may require very little identification, but the bank should not honor messages authenticated with such low-assurance certificates. Every CA must publicly state its identification requirements and policies; others can then attach an appropriate level of confidence to the certificates.

An attacker who discovers the private key of a certifying authority could then forge certificates. 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, or CSU (see Question 3.8).

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, and should also change keys regularly. Top-level certifying authorities are exceptions: it may not be practical for them to change keys frequently because the key may be written into software used by a large number of verifiers.

In another 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 it authentic 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, in some of today's CSUs, 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, as long as it is properly stored.

Another attack involves forging old documents. 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 raised by this attack is how to authenticate a signed document dated many years ago; this issue is discussed in Question 3.17.

Note that these attacks on certifying authorities do not threaten the privacy of messages between users, as might result from an attack on a secret-key distribution center.

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.

In some CSU designs, encrypted backup copies of the CA's private key are kept. A CA which loses its key can then restore it by loading the encrypted backup into the CSU, which can decrypt it using some unique information stored inside the CSU; the encrypted backup can only be decrypted using the CSU. If the CSU itself is destroyed, the manufacturer may be able to supply another with the same internal information, thus allowing recovery of the key.

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 should be recalled, and then reissued with a new CA key. These measures could be relaxed somewhat if certificates were registered with a digital time-stamping service (see Question 3.18). Note that compromise of a CA key does not invalidate users' keys, but only the certificates that authenticate them. Compromise of a top-level CA's key should be considered catastrophic, since the key may be built into applications that verify certificates.

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

A Certificate Revocation List (CRL) is a list 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. A key might have been compromised. A key might be used professionally by an individual for 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 would not want her to be able to sign messages with that key and therefore the company would place the key on the CRL.

When verifying a signature, one can check the relevant CRL to make sure the signer's key has not been revoked. Whether it is worth the time to perform this check depends on the importance of the signed document.

CRLs are maintained by certifying authorities (CAs) and provide information about revoked keys originally certified by the CA. CRLs only list current keys, since 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 CRLs are maintained in a distributed manner, there may 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.

3.12 *What happens when a key expires?*

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 equivalently, the key length must be long enough to make the chances of factoring before expiration extremely small. The validity period for a key pair may also depend on the circumstances in which the key will be used, although there will also be a standard period. The validity period, together with the value of the key and the estimated strength of an expected attacker, then determines the appropriate key size.

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 and 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 will be cases where it is important that a signed document be considered valid for a much longer period of time; Question 3.17 discusses ways to achieve this.

After expiration, the user chooses a new key, which should be longer than the old key, perhaps by several digits, to reflect both the performance increase of computer hardware and any recent improvements in factoring algorithms. Recommended key length schedules will likely be published. A user may recertify a key that has expired, if it is sufficiently long and has not been compromised. The certifying authority would then issue a new certificate for the same key, and all new signatures would point to the new certificate instead of the old. 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 cryptosystem. Faster hardware has the effect of increasing security, perhaps

vastly, but only if key lengths are increased regularly (see Question 4.5).

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

If your private key is lost or destroyed, but not compromised, you can no longer sign or decrypt messages, but anything previously signed with the lost key is still valid. This can happen, for example, if you forget the password used to access your key, or if the disk on which the key is stored is damaged. You need to choose a new key right away, to minimize the number of messages people send you encrypted under your old key, messages which you can no longer read.

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

If your private key is compromised, that is, if 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 underscores 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 3.11); this will inform people that the key has been revoked. Then choose a new key and obtain the proper certificates for it. You may wish to use the new key to re-sign documents that you had signed with the compromised key; documents that had been time-stamped as well as signed might still be valid. You should also change the way you store your private key, to prevent compromise of the new key.

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. The private key should never be stored anywhere in plaintext form. The simplest storage mechanism is to encrypt the private key under a password and store the result on a disk. Of course, the password itself must be maintained with high security, not written down and not easily guessed. Storing the encrypted key on a disk that is not accessible through a computer network, such as a floppy disk or a local hard disk, will make some attacks more difficult. Ultimately, private keys may be stored on portable hardware, such as a smart card. Furthermore, a challenge-response protocol will be more secure than simple password access. Users with extremely high security needs, such as certifying authorities, should use special hardware devices to protect their keys (see Question 3.8).

3.16 *How do I find someone else's public key?*

Suppose you want to find Bob's public key. There are several possible ways. You could call him up and ask him to send you his public key via e-mail; you could request it via e-mail as well. Certifying authorities may provide directory services; if Bob works for company Z, look in the directory kept by Z's certifying authority. Directories must be secure against unauthorized tampering, so that users can be confident that a public key listed in the directory actually belongs to the person listed. Otherwise, you might send private encrypted information to the wrong person.

Eventually, full-fledged directories will arise, serving as online white or yellow pages. If they are compliant with CCITT X.509 standards [19], the directories will contain certificates as well as public keys; the presence of certificates will lower the directories' security needs.

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, say, 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 are 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. A problem with this method is that any compromised key must remain on the relevant CRL until expiration (see Question 3.11); if 50-year keys are routinely placed on CRLs, the CRLs could grow in size to unmanageable proportions. This idea can be modified as follows. Register the long-term key by the normal procedure, i.e., for two years. 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 that the key will be valid for another two years. Now a compromised key only needs to be kept on a CRL for at most two years, not fifty.

One problem with the previous method 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 contract with a digital time-stamping service (see Question 3.18) at the time it is originally signed. If all parties to the contract keep a copy of the time-stamp, then each can prove that the contract was signed with valid keys. In fact, the time-stamp can prove the validity of a contract even if one signer's key gets compromised at some point after the contract was signed. This time-stamping solution can work with all signed digital

documents, not just multi-party contracts.

3.18 *What is a digital time-stamping service?*

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

Here's one way such a system could work. Suppose Alice signs a document and wants it time-stamped. She computes a message digest of the document using a secure hash function (see Question 8.2) and then sends the message digest (but not the document itself) to the DTS, which sends her in return a digital time-stamp consisting of the message digest, the date and time it was received at the DTS, and the signature of the DTS. Since the message digest does not reveal any information about the content of the document, the DTS cannot eavesdrop on the documents it time-stamps. Later, Alice can present the document and time-stamp together to prove when the document was written. A verifier computes the message digest of the document, makes sure it matches the digest in the time-stamp, and then verifies the signature of the DTS on the time-stamp.

To be reliable, the time-stamps must not be forgeable. Consider the requirements for a DTS of the type just described. First, the DTS itself must have a long key if we want the time-stamps to be reliable for, say, several decades. Second, the private key of the DTS must be stored with utmost security, as in a tamperproof box. 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 or perhaps for decades. Fourth, it must be infeasible to create time-stamps without using the apparatus in the tamperproof box.

A cryptographically strong DTS using only software [4] has been implemented by Bellcore; it avoids many of the requirements just described, such as tamperproof hardware. The Bellcore DTS essentially combines hash values of documents into data structures called binary trees, whose "root" values are periodically published in the newspaper. A time-stamp consists of a set of hash values which allow a verifier to recompute the root of the tree. Since the hash functions are one-way (see Question 8.2), the set of validating hash values cannot be forged. The time associated with the document by the time-stamp is the date of publication.

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, say, 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 becomes dissatisfied with the lease, he or she may refuse to cooperate. The solution is to register the lease with the DTS at the time of the original signing; both parties would then receive a copy of the time-stamp, which can be used years later to enforce the integrity of the original lease.

In the future, it is likely that a DTS 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 computer files; digital time-stamps may be the only way to authenticate the find.

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?*

Public-key cryptosystems are based on (presumed) trap-door one-way functions. The public key gives information about the particular instance of the function; the private key gives 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 secure for years, for example, it is necessary to use a trap-door one-way function with inputs large enough that someone without the trap door would need many years to compute the inverse function.

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 that it is theoretically possible that an algorithm will be discovered that can compute the inverse function easily without a trap door; this development would render any cryptosystem based on that one-way function insecure and useless.

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. For example, the factors of 15 are 3 and 5; the factoring problem is to find 3 and 5 when given 15. Prime factorization requires splitting an integer into factors that 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. Factoring an RSA modulus (see Question 2.1) would allow an attacker to figure out the private key; thus, anyone who can factor the modulus can decrypt messages and forge signatures. The security of RSA therefore depends on the factoring problem being difficult. Unfortunately, it has not been proven that factoring must be difficult, and there remains a possibility that a quick and easy factoring method might be discovered (see Question 4.7), although factoring researchers consider this possibility remote.

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

4.5 *Has factoring been getting easier?*

Factoring has become easier over the last fifteen years for two reasons: computer hardware has become more powerful, 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 hardware improvement that allows an attacker to factor a number two digits longer than before will at the same time allow a legitimate RSA user to use a key dozens of digits longer than before; a user can choose a new key a dozen digits longer than the old one without any performance slowdown, yet a factoring attack will become much more difficult. 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. This point holds for other public-key systems as well.

Better factoring algorithms have been more help to the RSA attacker than have hardware improvements. As the RSA system, and cryptography in general, have attracted much attention, so has the factoring problem, and many researchers have found new factoring methods or improved upon others. This has made factoring easier, for numbers of any size and irrespective of the 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, increases in key size (maintaining a constant 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 decreases, the security of RSA will increase, assuming RSA users regularly increase their key size by appropriate amounts. The open question is how much faster factoring algorithms can get; there must be some intrinsic limit to factoring speed, but this limit remains unknown.

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 references and their big-O asymptotic efficiency. O notation measures how fast an algorithm is; it gives an upper bound on 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 [41], [42], [47], and [11]; for a detailed explanation of big-O notation, see [22].

Factoring algorithms come in two flavors, special purpose and general purpose; the efficiency of the former depends on the unknown factors, whereas the efficiency of the latter depends on the number to be factored. Special purpose algorithms are best for factoring numbers with small factors, but the numbers used for the modulus in the RSA system do not have any small factors. Therefore, general purpose factoring algorithms are the more important ones in the context of cryptographic systems and their security.

Special purpose factoring algorithms include the Pollard rho method [66], with expected running time $O(\sqrt{p})$, and the Pollard $p-1$ method [67], with running time $O(p')$, where p' is the largest prime factor of $p-1$. Both of these

take an amount of time that is exponential in the size of p, the prime factor that they find; thus these algorithms are too slow for most factoring jobs. The elliptic curve method (ECM) [50] is superior to these; its asymptotic running time is $O(\exp(\sqrt{2 \ln p \ln \ln p}))$. The ECM is often used in practice to find factors of randomly generated numbers; it is not strong enough to factor a large RSA modulus.

The best general purpose factoring algorithm today is the number field sieve [16], which runs in time approximately $O(\exp(1.9(\ln n)^{1/3}(\ln \ln n)^{2/3}))$. It has only recently been implemented [15], and is not yet practical enough to perform the most desired factorizations. Instead, the most widely used general purpose algorithm is the multiple polynomial quadratic sieve (mpqs) [77], 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 110 digits; a variation known as ppmpqs [49] has been particularly popular.

It is expected that within a few years the number field sieve will overtake the mpqs as the most widely used factoring algorithm, as the size of the numbers being factored increases from about 120 digits, which is the current threshold of general numbers which can be factored, to 130 or 140 digits. A "general number" is one with no special form that might make it easier to factor; an RSA modulus is a general number. Note that a 512-bit number has about 155 digits.

Numbers that have a special form can already be factored up to 155 digits or more [48]. The Cunningham Project [14] keeps track of the factorizations of numbers with these special forms and maintains a "10 Most Wanted" list of desired factorizations. Also, a good way to survey current factoring capability is to look 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 could seriously weaken RSA, 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. Results of the contest may be useful to 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 e-mail to `challenge-info@rsa.com` for information.

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? There are other, more general, formulations as well.

Like the factoring problem, the discrete log problem is believed to be difficult and also to be the hard direction of a one-way function. For this reason, it has been the basis of several public-key cryptosystems, including the ElGamal system and DSS (see Questions 2.15 and 6.8). The discrete log problem bears the same relation to these systems as factoring does to RSA: the security of these systems rests on the assumption that discrete logs are difficult to compute.

The discrete log problem has received much attention in recent years; descriptions of some of the most efficient algorithms can be found in [47], [21], and [33]. The best discrete log problems have expected running times similar to that of the best factoring algorithms. Rivest [72] has analyzed the expected time to solve discrete log both in terms of computing power 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 modulo a 512-bit prime as to factor a 512-bit RSA modulus. One paper [45] cites experimental evidence that the discrete log problem is slightly harder than factoring: the authors 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. 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 and endorsed by the U.S. government in 1977 as an official standard; the details can be found in the official FIPS publication [59]. 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, such as 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 its operation is relatively fast. It works well for bulk encryption, that is, for encrypting a large set of data.

NIST (see Question 7.1) has recertified DES as an official U.S. government encryption standard every five years; DES was last recertified in 1993, by default. NIST has indicated, however, that it may not recertify DES again.

5.2 *Has DES been broken?*

DES has never been "broken", despite the efforts of many researchers over many years. The obvious method of attack is brute-force exhaustive search of the key space; this takes 2^{55} steps on average. Early on it was suggested [28] 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 [36] showed a time-memory trade-off that allows improvement over exhaustive search if memory space is plentiful, after an exhaustive precomputation. These ideas fostered doubts about the security of DES. There were also accusations that the NSA had intentionally weakened DES. Despite these suspicions, no feasible way to break DES faster than exhaustive search was discovered. The cost of a specialized computer to perform exhaustive search has been estimated by Wiener at one million dollars [80].

Just recently, however, the first attack on DES that is better than exhaustive search was announced by Eli Biham and Adi Shamir [6, 7], using a new technique known as differential cryptanalysis. This attack requires encryption of 2^{47} chosen plaintexts, i.e., plaintexts chosen by the attacker. 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. Another attack, known as linear cryptanalysis [51], does not require chosen plaintexts.

The consensus is that DES, when used properly, is secure against all but the most powerful enemies. In fact, triple encryption DES (see Question 5.3) may be secure against anyone at all. Biham and Shamir have stated that they consider DES secure. It is used extensively in a wide variety of cryptographic systems, and in fact, most implementations of public-key cryptography include DES at some level.

5.3 *How does one use DES securely?*

When using DES, there are several practical considerations that can affect the security of the encrypted data. One should change DES keys frequently, in order to prevent attacks that require sustained data analysis. In a communications context, one must also find a secure way of communicating the DES key to both sender and receiver. Use of RSA or some other public-key technique for key management solves both these issues: a different DES key is generated for each session, and secure key management is provided by encrypting the DES key with the receiver's RSA public key. RSA, in this circumstance, can be regarded as a tool for improving the security of DES (or any other secret key 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.

A powerful technique for improving the security of DES is triple encryption, that is, encrypting each message block under three different DES keys in succession. Triple encryption is thought to be equivalent to doubling the key size of DES, to 112 bits, and should prevent decryption by an enemy capable of single-key exhaustive search [53]. Of course, using triple-encryption takes three times as long as single-encryption DES.

Aside from the issues mentioned above, DES can be used for encryption in several officially defined 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. 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. CBC mode helps protect against certain attacks, although not against exhaustive search or differential cryptanalysis. CFB (cipher feedback) mode allows one to use DES with block lengths less than 64 bits. Detailed descriptions of the various DES modes can be found in [60].

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 single DES in CBC mode is usually considered secure enough,

triple encryption is not often 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.

5.5 *What are the alternatives to DES?*

Over the years, various bulk encryption algorithms have been designed as alternatives to DES. One is FEAL (Fast Encryption ALgorithm), a cipher for which attacks have been discovered [6], although new versions have been proposed. Another recently proposed cipher designed by Lai and Massey [44] and known as IDEA seems promising, although it has not yet received sufficient scrutiny to instill full confidence in its security. The U.S. government recently announced a new algorithm called Skipjack (see Question 6.5) as part of its Capstone project. Skipjack operates on 64-bit blocks of data, as does DES, but uses 80-bit keys, as opposed to 56-bit keys in DES. However, the details of Skipjack are classified, so Skipjack is only available in hardware from government-authorized manufacturers.

Rivest has developed the ciphers RC2 and RC4 (see Question 8.6), which can be made as secure as necessary because they use variable key sizes. Faster than DES, at least in software, they have the further advantage of special U.S. government status whereby the export approval is simplified and expedited if the key size is limited to 40 bits.

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 always equivalent to a single encryption under a single key? Algebraically, is DES a group? If so, then DES might be weaker than would otherwise be the case; see [39] for a more complete discussion. However, the answer is no, DES is not a group [18]; this issue was settled only recently, after many years of speculation and circumstantial evidence. This result seems to imply that techniques such as triple encryption do in fact increase the security of DES.

6 Capstone, Clipper, and DSS

6.1 *What is Capstone?*

Capstone is the U.S. government's long-term project to develop a set of standards for publicly-available cryptography, as authorized by the Computer Security Act of 1987. The primary agencies responsible for Capstone are NIST and the NSA (see Section 7). The plan calls for the elements of Capstone to become official U.S. government standards, in which case both the government itself and all private companies doing business with the government would be required to use Capstone.

There are four major components of Capstone: a bulk data encryption algorithm, a digital signature algorithm, a key exchange protocol, and a hash function. The data encryption algorithm is called Skipjack (see Question 6.5), but is often referred to as Clipper, which is the encryption chip that includes Skipjack (see Question 6.2). The digital signature algorithm is DSS (see Question 6.8) and the hash function is SHS (see Question 8.4 about SHS and Question 8.2 about hash functions). The key exchange protocol has not yet been announced.

All the parts of Capstone have 80-bit security: all the keys involved are 80 bits long and other aspects are also designed to withstand anything less than an "80-bit" attack, that is, an effort of 2^{80} operations. Eventually the government plans to place the entire Capstone cryptographic system on a single chip.

6.2 *What is Clipper?*

Clipper is an encryption chip developed and sponsored by the U.S. government as part of the Capstone project (see Question 6.1). Announced by the White House in April, 1993 [65], Clipper was designed to balance the competing concerns of federal law-enforcement agencies with those of private citizens and industry. The law-enforcement agencies wish to have access to the communications of suspected criminals, for example by wire-tapping; these needs are threatened by secure cryptography. Industry and individual citizens, however, want secure communications, and look to cryptography to provide it.

Clipper technology attempts to balance these needs by using escrowed keys. The idea is that communications would be encrypted with a secure algorithm, but the keys would be kept by one or more third parties (the "escrow agencies"), and made available to law-enforcement agencies when authorized by a court-issued warrant. Thus, for example, personal communications would be impervious to recreational eavesdroppers, and commercial communications would be impervious to industrial espionage, and yet the FBI could listen in on suspected terrorists or gangsters.

Clipper has been proposed as a U.S. government standard [62]; it would then be used by anyone doing business with the federal government as well as for communications within the government. For anyone else, use of Clipper

is strictly voluntary. AT&T has announced a secure telephone that uses the Clipper chip.

6.3 *How does the Clipper chip work?*

The Clipper chip contains an encryption algorithm called Skipjack (see Question 6.5), whose details have not been made public. Each chip also contains a unique 80-bit unit key U, which is escrowed in two parts at two escrow agencies; both parts must be known in order to recover the key. Also present is a serial number and an 80-bit "family key" F; the latter is common to all Clipper chips. The chip is manufactured so that it cannot be reverse engineered; this means that the Skipjack algorithm and the keys cannot be read off the chip.

When two devices wish to communicate, they first agree on an 80-bit "session key" K. The method by which they choose this key is left up to the implementer's discretion; a public-key method such as RSA or Diffie-Hellman seems a likely choice. The message is encrypted with the key K and sent; note that the key K is not escrowed. In addition to the encrypted message, another piece of data, called the law-enforcement access field (LEAF), is created and sent. It includes the session key K encrypted with the unit key U, then concatenated with the serial number of the sender and an authentication string, and then, finally, all encrypted with the family key. The exact details of the law-enforcement field are classified.

The receiver decrypts the law-enforcement field, checks the authentication string, and decrypts the message with the key K.

Now suppose a law-enforcement agency wishes to tap the line. It uses the family key to decrypt the law-enforcement field; the agency now knows the serial number and has an encrypted version of the session key. It presents an authorization warrant to the two escrow agencies along with the serial number. The escrow agencies give the two parts of the unit key to the law-enforcement agency, which then decrypts to obtain the session key K. Now the agency can use K to decrypt the actual message.

Further details on the Clipper chip operation, such as the generation of the unit key, are sketched by Denning [26].

6.4 *Who are the escrow agencies?*

It has not yet been decided which organizations will serve as the escrow agencies, that is, keep the Clipper chip keys. No law-enforcement agency will be an escrow agency, and it is possible that at least one of the escrow agencies will be an organization outside the government.

It is essential that the escrow agencies keep the key databases extremely secure, since unauthorized access to both escrow databases could allow unauthorized eavesdropping on private communications. In fact, the escrow agencies

are likely to be one of the major targets for anyone trying to compromise the Clipper system; the Clipper chip factory is another likely target.

6.5 *What is Skipjack?*

Skipjack is the encryption algorithm contained in the Clipper chip; it was designed by the NSA. It uses an 80-bit key to encrypt 64-bit blocks of data; the same key is used for the decryption. Skipjack can be used in the same modes as DES (see Question 5.3), and may be more secure than DES, since it uses 80-bit keys and scrambles the data for 32 steps, or "rounds"; by contrast, DES uses 56-bit keys and scrambles the data for only 16 rounds.

The details of Skipjack are classified. The decision not to make the details of the algorithm publicly available has been widely criticized. Many people are suspicious that Skipjack is not secure, either due to oversight by its designers, or by the deliberate introduction of a secret trapdoor. By contrast, there have been many attempts to find weaknesses in DES over the years, since its details are public. These numerous attempts (and the fact that they have failed) have made people confident in the security of DES. Since Skipjack is not public, the same scrutiny cannot be applied towards it, and thus a corresponding level of confidence may not arise.

Aware of such criticism, the government invited a small group of independent cryptographers to examine the Skipjack algorithm. They issued a report [12] which stated that, although their study was too limited to reach a definitive conclusion, they nevertheless believe that Skipjack is secure.

Another consequence of Skipjack's classified status is that it cannot be implemented in software, but only in hardware by government-authorized chip manufacturers.

6.6 *Why is Clipper controversial?*

The Clipper chip proposal has aroused much controversy and has been the subject of much criticism. Unfortunately two distinct issues have become confused in the large volume of public comment and discussion.

First there is controversy about the whole idea of escrowed keys. Those in favor of escrowed keys see it as a way to provide secure communications for the public at large while allowing law-enforcement agencies to monitor the communications of suspected criminals. Those opposed to escrowed keys see it as an unnecessary and ineffective intrusion of the government into the private lives of citizens. They argue that escrowed keys infringe their rights of privacy and free speech. It will take a lot of time and much public discussion for society to reach a consensus on what role, if any, escrowed keys should have.

The second area of controversy concerns various objections to the specific Clipper proposal, that is, objections to this particular implementation of escrowed keys, as opposed to the idea of escrowed keys in general. Common

objections include: the Skipjack algorithm is not public (see Questions 6.5) and may not be secure; the key escrow agencies will be vulnerable to attack; there are not enough key escrow agencies; the keys on the Clipper chips are not generated in a sufficiently secure fashion; there will not be sufficient competition among implementers, resulting in expensive and slow chips; software implementations are not possible; and the key size is fixed and cannot be increased if necessary.

Micali [55] has recently proposed an alternative system that also attempts to balance the privacy concerns of law-abiding citizens with the investigative concerns of law-enforcement agencies. Called fair public-key cryptography, it is similar in function and purpose to the Clipper chip proposal but users can choose their own keys, which they register with the escrow agencies. Also, the system does not require secure hardware, and can be implemented completely in software.

6.7 *What is the current status of Clipper?*

Clipper is under review. Both the executive branch and Congress are considering it, and an advisory panel recently recommended a full year-long public discussion of cryptography policy. NIST has invited the public to send comments, as part of its own review.

6.8 *What is DSS?*

DSS is the proposed Digital Signature Standard, which specifies a Digital Signature Algorithm (DSA), and is a part of the U.S. government's Capstone project (see Question 6.1). It was selected by NIST, in cooperation with the NSA (see Section 7), 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 is based on the discrete log problem (see Question 4.9) and derives from cryptosystems proposed by Schnorr [75] and ElGamal [30]. It is for authentication only. For a detailed description of DSS, see [63] or [57].

DSS has, for the most part, been looked upon unfavorably by the computer industry, much of which had hoped the government would choose the RSA algorithm as the official standard; RSA is the most widely used authentication algorithm. Several articles in the press, such as [54], discuss the industry dissatisfaction with DSS. Criticism of DSS has focused on a few main issues: it lacks key exchange capability; the underlying cryptosystem is too recent and has been subject to too little scrutiny for users to be confident of its strength; verification of signatures with DSS is too slow; the existence of a second authentication standard 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. Other criticisms were addressed by NIST by modifying the original proposal.

A more detailed discussion of the various criticisms can be found in [57], and a detailed response by NIST can be found in [78].

In the DSS system, signature generation is faster than signature verification, whereas in the RSA system, signature verification is faster than signature generation (if the public and private exponents are chosen for this property, which is the usual case). NIST claims that it is an advantage of DSS that signing is faster, but many people in cryptography think that it is better for verification to be the faster operation.

6.9 *Is DSS secure?*

The most serious criticisms of DSS involve its security. DSS was originally proposed with a fixed 512-bit key size. After much criticism that this is not secure enough, NIST revised DSS to allow key sizes up to 1024 bits. More critical, however, is the fact that DSS has not been around long enough to withstand repeated 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 by Schnorr [75] and has not received much public study. In general, any new cryptosystem could have serious flaws that are only discovered after years of scrutiny by cryptographers. Indeed this has happened many times in the past; see [13] for some detailed examples. RSA has withstood over 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 warned 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 are easily avoided if proper key generation procedures are followed [78].

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

NIST has filed a patent application for DSS and there have been claims that DSS is covered by other public-key patents. NIST recently announced its intention to grant exclusive sublicensing rights for the DSS patent to Public Key Partners (PKP), which also holds the sublicensing rights to other patents that may cover DSS (see Question 1.5). In the agreement between NIST and PKP, PKP publicly stated uniform guidelines by which it will grant licenses to practice DSS. PKP stated that DSS can be used on a royalty-free basis in the case of personal, noncommercial, or U.S. government use. See [61] for details on the agreement and the licensing policy.

6.11 *What is the current status of DSS?*

After NIST issued the DSS proposal in August 1991, there was a period in which comments from the public were solicited; NIST then revised its proposal in light of the comments. DSS may be issued as a FIPS and become the official U.S. government standard, but it is not clear when this might happen. DSS is currently in the process of becoming a standard, along with RSA, for the financial services industry; a recent draft standard [1] contains the revised version of DSS.

7 NIST and NSA

7.1 *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 it aims to promote open systems and interoperability that will spur development of computer-based economic activity. NIST issues standards and guidelines that it hopes will be adopted by all computer systems in the U.S., and also sponsors workshops and seminars. Official standards are published as FIPS (Federal Information Processing Standards) publications.

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

7.2 *What role does NIST play in cryptography?*

NIST issues standards for cryptographic routines; U.S. government agencies are required to use them, and the private sector often adopts them as well. In January 1977, NIST declared DES (see Question 5.1) the official U.S. encryption standard and published it as FIPS Publication 46; DES soon became a de facto standard throughout the U.S.

A few years ago, NIST was asked to choose a set of cryptographic standards for the U.S.; this has become known as the Capstone project (see Section 6). After a few years of rather secretive deliberations, and in cooperation with the NSA, NIST issued proposals for various standards in cryptography, including digital signatures (DSS) and data encryption (the Clipper chip); these are pieces of the overall Capstone project.

NIST has been criticized for allowing the NSA too much power in setting cryptographic standards, since the interests of the NSA conflict with that of the Commerce Department and NIST. Yet, the NSA has much more experience

with cryptography, and many more qualified cryptographers and cryptanalysts, than does NIST; it would be unrealistic to expect NIST to forego such available assistance.

7.3 *What is the NSA?*

The NSA is the National Security Agency, a highly secretive agency of the U.S. government that was created by Harry Truman in 1952; its very existence was kept secret for many years. For a history of the NSA, see Bamford [2]. The NSA has a mandate to listen to and decode all foreign communications of interest to the security of the United States. It has also used its power in various ways (see Question 7.4) to slow the spread of publicly available cryptography, in order to prevent national enemies from employing encryption methods too strong for the NSA to break.

As the premier cryptographic government agency, the NSA has huge financial and computer resources and employs a host of cryptographers. Developments in cryptography achieved at the NSA are not made public; this secrecy has led to many rumors about the NSA's ability to break popular cryptosystems like DES and also to rumors that the NSA has secretly placed weaknesses, called trap doors, in government-endorsed cryptosystems, such as DES. These rumors have never been proved or disproved, and the criteria used by the NSA in selecting cryptography standards have never been made public.

Recent advances in the computer and telecommunications industries have placed NSA actions under unprecedented scrutiny, and the agency has become the target of heavy criticism for hindering U.S. industries that wish to use or sell strong cryptographic tools. The two main reasons for this increased criticism are the collapse of the Soviet Union and the development and spread of commercially available public-key cryptographic tools. Under pressure, the NSA may be forced to change its policies.

7.4 *What role does the NSA play in commercial cryptography?*

The NSA's charter limits its activities to foreign intelligence. However, the NSA is concerned with the development of commercial cryptography because the availability of strong encryption tools through commercial channels could impede the NSA's mission of decoding international communications; in other words, the NSA is worried lest strong commercial cryptography fall into the wrong hands.

The NSA has stated that it has no objection to the use of secure cryptography by U.S. industry. It also has no objection to cryptographic tools used for authentication, as opposed to privacy. However, the NSA is widely viewed as following policies that have the practical effect of limiting and/or weakening the cryptographic tools used by law-abiding U.S. citizens and corporations; see Barlow [3] for a discussion of NSA's effect on commercial cryptography.

The NSA exerts influence over commercial cryptography in several ways. First, it controls the export of cryptography from the U.S. (see Question 1.6); the NSA generally does not approve export of products used for encryption unless the key size is strictly limited. It does, however, approve for export any products used for authentication only, no matter how large the key size, so long as the product cannot be converted to be used for encryption. The NSA has also blocked encryption methods from being published or patented, citing a national security threat; see Landau [46] for a discussion of this practice. Additionally, the NSA serves an "advisory" role to NIST in the evaluation and selection of official U.S. government computer security standards; in this capacity, it has played a prominent, and controversial, role in the selection of DES and in the development of the group of standards known as the Capstone project (see Section 6), which includes DSS and the Clipper chip. The NSA can also exert market pressure on U.S. companies to produce (or refrain from producing) cryptographic goods, since the NSA itself is often a large customer of these companies.

Cryptography is in the public eye as never before and has become the subject of national public debate. The status of cryptography, and the NSA's role in it, will probably change over the next few years.

8 Miscellaneous

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

If digital signatures are to replace handwritten signatures they must have the same legal status as handwritten signatures, i.e., documents signed with digital signatures must be legally binding. NIST has stated that its proposed Digital Signature Standard (see Question 6.8) 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 requested by NIST also opines that digital signatures will meet the legal standards of handwritten signatures [20].

However, since the validity of documents with digital signatures has never been challenged in court, their legal status is not yet well-defined. Through such challenges, the courts will issue rulings that collectively define which digital signature methods, key sizes, and security precautions are acceptable for a digital signature to be legally binding.

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.

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

8.2 *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 fixed-size string, 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 message digest as a "digital fingerprint" of the larger document. Examples of well-known hash functions are MD4, MD5, and SHS (see Questions 8.3 and 8.4).

Although hash functions in general have many uses in computer programs, in cryptography they are used to generate a small string (the message digest) that can represent securely a much larger string, such as a file or message. Since the hash functions are faster than the signing functions, it is much more efficient to compute a digital signature using a document's message digest, which is small, than using the arbitrarily large document itself. Additionally, a digest can be made public without revealing the contents of the document from which it derives. This is important in digital time-stamping, where, using hash functions, one can get a document time-stamped without revealing its contents to the time-stamping 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 that hashes to a given value and it must be infeasible to find two distinct messages that 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. It would also enable someone to falsely disown a message by claiming that he or she actually signed a different message hashing to the same value, thus violating the non-repudiation property of digital signatures. The ability to find two distinct messages hashing to the same value could enable an attack whereby someone is tricked into signing a message which hashes to the same value as another message with a quite different meaning. The digest must therefore be long enough to prevent an attacker from doing an exhaustive search for a collision. For example, if a hash function produces 100-bit strings, 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.

A digital signature system can be broken by attacking either the difficult mathematical problem on which the signature method is based 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. Actually, attacking the hash function is harder in practice, since it requires a large amount of memory and the ability to trick the victim into signing a special message. With 2^{64} operations, an attacker can find two messages that hash to the same digest under any of the MD hash functions; this effort is comparable to that necessary to break 512-bit RSA; thus 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 [71] would suffice.

8.3 *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 there is no known attack faster than exhaustive search.

MD2 is the slowest of the three; MD4 [71] is the fastest. MD5 [73] has been dubbed "MD4 with safety belts" by Rivest, since it has a more conservative design than MD4; the design gives it increased security against attack, but at a cost of being approximately 33% slower than MD4. MD5 is the most commonly used of the three algorithms. MD4 and MD5 are publicly available for unrestricted use; MD2 is available for use with PEM (see Question 8.7). Details of MD2, MD4, and MD5 with sample C code are available in Internet RFCs (Requests For Comments) 1319, 1320, and 1321, respectively.

No feasible attacks on any of the MD algorithms have been discovered, although some recent theoretical work has found some interesting structural properties [24, 25].

8.4 *What is SHS?*

The Secure Hash Standard (SHS) [58] is a hash function proposed by NIST (see Question 7.1) and adopted as a U.S. government standard. It is designed for use with the proposed Digital Signature Standard (see Question 6.8) and is part of the government's Capstone project (see Question 6.1). SHS produces a 160-bit hash value from a variable-size input. SHS is structurally similar to MD4 and MD5. It is roughly 25% slower than MD5 but may be more secure, because it produces message digests that are 25% longer than those produced by the MD

functions. SHS is currently the only part of Capstone that has been officially adopted as a government standard.

8.5 *What is Kerberos?*

Kerberos is a secret-key network authentication system developed at MIT [79]; it uses DES for encryption and authentication. Unlike a public-key authentication system, it does not produce digital signatures: Kerberos was designed to authenticate requests for network resources rather than to authenticate authorship of documents. Kerberos provides real-time authentication in a distributed environment, but does not provide for future third-party verification of documents.

In a Kerberos system, there is a designated site on the network, called the Kerberos server, which performs centralized key management and administrative functions. The server maintains a database containing the secret keys of all users, generates session keys whenever two users wish to communicate securely, and authenticates the identity of a user who requests certain network services.

Kerberos, like other secret-key systems, requires trust in a third party, in this case the Kerberos server. If the server were compromised, the integrity of the whole system would fall. Public-key cryptography was designed precisely to avoid the necessity to trust third parties or communication lines (see Question 1.4). Kerberos may be adequate for those who do not need the more robust functions and properties of public-key systems.

8.6 *What are RC2 and RC4?*

RC2 and RC4 are variable-key-size cipher functions designed by Ron Rivest for fast bulk encryption. They are alternatives to DES (see Question 5.1) and are as fast or faster than DES. They can be 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 can be used in the same modes as DES (see Question 5.3), including triple encryption. RC2 is approximately twice as fast as DES, at least in software. RC4 is a variable-key-size symmetric stream cipher and is 10 or more times as fast as DES in software. Both RC2 and RC4 are very compact in terms of code size.

An agreement between the Software Publishers Association (SPA) and the U.S. government gives RC2 and RC4 special status by means of which the export approval process is simpler and quicker than the usual cryptographic export process. However, to qualify for quick export approval a product must limit the RC2 and RC4 key sizes to 40 bits; 56 bits is allowed for foreign subsidiaries and overseas offices of U.S. companies. An additional 40-bit string, called a *salt*,

can be used to thwart attackers who try to precompute a large look-up table of possible encryptions. The salt is appended to the encryption key, and this lengthened key is used to encrypt the message; the salt is then sent, unencrypted, with the message. 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 have not been published.

8.7 *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. Designed to work with current Internet e-mail formats, PEM includes encryption, authentication, and key management, and allows use of both public-key and secret-key cryptosystems. Multiple cryptographic tools are supported: for each mail message, the specific encryption algorithm, digital signature algorithm, hash function, and so on are specified in the header. PEM explicitly supports only a few cryptographic algorithms; others may be added later. DES in CBC mode is currently the only message encryption algorithm supported, and both RSA and DES are supported for the key management. PEM also supports the use of certificates, endorsing the CCITT X.509 standard for certificate structure.

The details of PEM can be found in Internet RFCs (Requests For Comments) 1421 through 1424. PEM is likely to be officially adopted by the Internet Activities Board within one year. Trusted Information Systems has developed a free non-commercial implementation of PEM, and other implementations should soon be available as well.

8.8 *What is RIPEM?*

RIPEM is a program developed by Mark Riordan that enables secure Internet e-mail; it provides both encryption and digital signatures, using RSA and DES routines from RSAREF (see Question 8.10). RIPEM is not fully PEM-compatible; for example, it does not currently support certificates. However, future versions will include certificates and will be fully compliant with the PEM standard. RIPEM is available free for non-commercial use in the U.S. and Canada. To get RIPEM, obtain an ftp account at `ripem.msu.edu`.

8.9 *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 8.7) but extends beyond PEM. For example, where PEM can only handle ASCII data, PKCS is designed for binary data as well. PKCS is also compatible with the CCITT X.509 standard.

PKCS includes both algorithm-specific and algorithm-independent implementation standards. Specific algorithms supported include RSA, DES, and Diffie-Hellman key exchange. It also defines algorithm-independent syntax for digital signatures, digital envelopes (for encryption), and certificates; this enables someone implementing any cryptographic algorithm whatsoever to conform to a standard syntax and thus preserve interoperability. Documents detailing the PKCS standards can be obtained by sending e-mail to `pkcs@rsa.com` or by anonymous ftp to `rsa.com`.

8.10 *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; Diffie-Hellman key exchange will be included in a forthcoming version. It includes both low-level subroutines, such as modular exponentiation, and high-level cryptographic functions, such as verification of digital signatures. 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 but cannot be used commercially or sent outside the U.S. and Canada. The RSAREF license contains more details on the usage allowed and disallowed. RSAREF is available on the Internet by sending e-mail to `rsaref@rsa.com` or by ftp to `rsa.com`.

9 Acknowledgements

I would like to thank the following people, who have provided information and helpful suggestions: Burt Kaliski, Jim Bidzos, Matt Robshaw, Steve Dusse, Kurt Stammberger, George Parsons, John Gilmore, Stuart Haber, Dorothy Denning, and Dennis Branstad.

References

[1] American National Standards Institute. *Working Draft: American National Standard X9.30-199X: Public Key Cryptography Using Irreversible*

Algorithms for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA). American Bankers Association, Washington, D.C., March 4, 1993.

[2] J. Bamford. *The Puzzle Palace*. Houghton Mifflin, Boston, 1982.

[3] J.P. Barlow. Decrypting the puzzle palace. *Communications of the ACM*, 35(7):25–31, July 1992.

[4] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliablility of digital time-stamping. In R.M. Capocelli, editor, *Sequences '91: Methods in Communication, Security, and Computer Science*, Springer-Verlag, Berlin, 1992.

[5] P. Beauchemin, G. Brassard, C. Crepeau, C. Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *J. of Cryptology*, 1:53–64, 1988.

[6] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, New York, 1993.

[7] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[8] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology — Crypto '84*, pages 289–299, Springer-Verlag, New York, 1985.

[9] J. Brandt and I. Damgård. On generation of probable primes by incremental search. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[10] G. Brassard. *Modern Cryptology*. Volume 325 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1988.

[11] D.M. Bressoud. *Factorization and Primality Testing. Undergraduate Texts in Mathematics*, Springer-Verlag, New York, 1989.

[12] E.F. Brickell, D.E. Denning, S.T. Kent, D.P. Maher, and W. Tuchman. *Skipjack Review, Interim Report: The Skipjack Algorithm*. July 28, 1993.

[13] E.F. Brickell and A.M. Odlyzko. Cryptanalysis: A survey of recent results. *Proceedings of the IEEE*, 76:578–593, 1988.

[14] J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, and S.S. Wagstaff Jr. *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to High Powers*. Volume 22 of *Contemporary Mathematics*, American Mathematical Society, Providence, Rhode Island, 2nd edition, 1988.

[15] J. Buchmann, J. Loho, and J. Zayer. An implementation of the general number field sieve. In *Advances, in Cryptology — Crypto '93*, Springer-Verlag, New York, 1994. To appear.

[16] J.P. Buhler, H.W. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. 1992. To appear.

[17] M.V.D. Burmester, Y.G. Desmedt, and T. Beth. Efficient zero-knowledge identification schemes for smart cards. *Computer Journal*, 35:21–29, 1992.

[18] K.W. Campbell and M.J. Wiener. Proof that DES is not a group. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[19] CCITT (Consultative Committee on International Telegraphy and Telephony). *Recommendation X.509: The Directory—Authentication Framework*. 1988.

[20] Comptroller General of the United States. *Matter of National Institute of Standards and Technology — Use of Electronic Data Interchange Technology to Create Valid Obligations*. December 13, 1991. File B-245714.

[21] D. Coppersmith, A.M. Odlyzko, and R. Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1:1–15, 1986.

[22] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[23] G. Davida. *Chosen signature cryptanalysis of the RSA public key cryptosystem*. Technical Report TR-CS-82-2, Dept of EECS, University of Wisconsin, Milwaukee, 1982.

[24] B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In *Advances in Cryptology — Crypto '91*, pages 194–203, Springer-Verlag, New York, 1992.

[25] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology — Eurocrypt '93*, 1993. Preprint.

[26] Dorothy E. Denning. The Clipper encryption system. *American Scientist*, 81(4):319–323, July–August 1993.

[27] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76:560–577, 1988.

[28] W. Diffie and M.E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10:74–84, 1977.

[29] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[30] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.

[31] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86*, pages 186–194, Springer-Verlag, New York, 1987.

[32] S. Goldwasser and S. Micali. Probabilistic encryption. *J. of Computer and System Sciences*, 28:270–299, 1984.

[33] D.M. Gordon. Discrete logarithms using the number field sieve. March 28, 1991. To appear.

[34] D.M. Gordon and K.S. McCurley. Massively parallel computation of discrete logarithms. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[35] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Computing*, 17:336–241, 1988.

[36] M.E. Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, IT-26:401–406, 1980.

[37] D. Kahn. *The Codebreakers*. Macmillan Co., New York, 1967.

[38] B.S. Kaliski. *A survey of encryption standards*. RSA Data Security, Inc., September 2, 1993.

[39] B.S. Kaliski Jr., R.L. Rivest, and A.T. Sherman. Is the data encryption standard a group? *J. of Cryptology*, 1:3–36, 1988.

[40] S. Kent. *RFC 1422: Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management*. Internet Activities Board, February 1993.

[41] D.E. Knuth. *The Art of Computer Programming*. Volume 2, Addison-Wesley, Reading, Mass., 2nd edition, 1981.

[42] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, New York, 1987.

[43] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[44] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology — Eurocrypt '90*, pages 389–404, Springer-Verlag, Berlin, 1991.

[45] B.A. LaMacchia and A.M. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography*, 1:47–62, 1991.

[46] S. Landau. Zero knowledge and the Department of Defense. *Notices of the American Mathematical Society*, 35:5–12, 1988.

[47] A.K. Lenstra and H.W. Lenstra Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, MIT Press/Elsevier, Amsterdam, 1990.

[48] A.K. Lenstra, H.W. Lenstra Jr., M.S. Manasse, and J.M. Pollard. The factorization of the ninth Fermat number. 1991. To appear.

[49] A.K. Lenstra and M.S. Manasse. Factoring with two large primes. In *Advances in Cryptology — Eurocrypt '90*, pages 72–82, Springer-Verlag, Berlin, 1991.

[50] H.W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649–673, 1987.

[51] M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology — Eurocrypt '93*, Springer-Verlag, Berlin, 1993. To appear.

[52] R.C. Merkle and M.E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24:525–530, 1978.

[53] R.C. Merkle and M.E. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24:465–467, July 1981.

[54] E. Messmer. NIST stumbles on proposal for public-key encryption. *Network World*, 9(30), July 27, 1992.

[55] S. Micali. Fair public-key cryptosystems. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[56] V.S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — Crypto '85*, pages 417–426, Springer-Verlag, New York, 1986.

[57] National Institute of Standards and Technology (NIST). The Digital Signature Standard, proposal and discussion. *Communications of the ACM*, 35(7):36–54, July 1992.

[58] National Institute of Standards and Technology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS).* May 11, 1993.

[59] National Institute of Standards and Technology (NIST). *FIPS Publication 46-1: Data Encryption Standard.* January 22, 1988. Originally issued by National Bureau of Standards.

[60] National Institute of Standards and Technology (NIST). *FIPS Publication 81: DES Modes of Operation*. December 2, 1980. Originally issued by National Bureau of Standards.

[61] National Institute of Standards and Technology (NIST). Notice of proposal for grant of exclusive patent license. *Federal Register*, 58(108), June 8, 1993.

[62] National Institute of Standards and Technology (NIST). A proposed Federal Information Processing Standard for an Escrowed Encryption Standard (EES). *Federal Register*, 58(145), July 30, 1993.

[63] National Institute of Standards and Technology (NIST). *Publication XX: Announcement and Specifications for a Digital Signature Standard (DSS)*. August 19, 1992.

[64] A.M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Advances in Cryptology — Eurocrypt '84*, pages 224–314, Springer-Verlag, Berlin, 1984.

[65] Office of the Press Secretary. *Statement*. The White House, April 16, 1993.

[66] J. Pollard. Monte Carlo method for factorization. *BIT*, 15:331–334, 1975.

[67] J. Pollard. Theorems of factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.

[68] M.O. Rabin. *Digitalized signatures as intractable as factorization*. Technical Report MIT/LCS/TR-212, MIT, 1979.

[69] R.L. Rivest. Cryptography. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, MIT Press/Elsevier, Amsterdam, 1990.

[70] R.L. Rivest. Finding four million random primes. In *Advances in Cryptology — Crypto '90*, pages 625–626, Springer-Verlag, New York, 1991.

[71] R.L Rivest. The MD4 message digest algorithm. In *Advances in Cryptology — Crypto '90*, pages 303–311, Springer-Verlag, New York, 1991.

[72] R.L. Rivest. Response to NIST's proposal. *Communications of the ACM*, 35:41–47, July 1992.

[73] R.L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, April 1992.

[74] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[75] C.P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology — Crypto '89*, pages 239–251, Springer-Verlag, New York, 1990.

[76] M. Shand and J. Vuillemin. Fast implementations of RSA cryptography. In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 252–259, IEEE Computer Society Press, Los Alamitos, CA, 1993.

[77] R.D. Silverman. The multiple polynomial quadratic sieve. *Math. Comp.*, 48:329–339, 1987.

[78] M.E. Smid and D.K. Branstad. Response to comments on the NIST proposed Digital Signature Standard. In *Advances in Cryptology — Crypto '92*, Springer-Verlag, New York, 1993.

[79] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: an authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, Texas, February 1988.

[80] M.J. Wiener. Efficient DES key search. August 20, 1993. Presented at Crypto '93 rump session.

RSA Laboratories is the research and consultation division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

For more information about RSA Laboratories, call or write to

RSA Laboratories
100 Marine Parkway
Redwood City, CA 94065
(415) 595-7703
(415) 595-4126 (fax)

PKCS, RSAREF and RSA Laboratories are trademarks of RSA Data Security, Inc. All other trademarks belong to their respective companies.

Please send comments and corrections to `faq-editor@rsa.com`.