

An Overview of Cryptography

Gary C. Kessler

May 1998

(17 November 2006)

A much shorter, edited version of this paper appears in the 1999 Edition of *Handbook on Local Area Networks*, published by Auerbach in September 1998. Since that time, this article has taken on a life of its own...

CONTENTS

- [1. INTRODUCTION](#)
- [2. THE PURPOSE OF CRYPTOGRAPHY](#)
- [3. TYPES OF CRYPTOGRAPHIC ALGORITHMS](#)
 - [3.1. Secret Key Cryptography](#)
 - [3.2. Public-Key Cryptography](#)
 - [3.3. Hash Functions](#)
 - [3.4. Why Three Encryption Techniques?](#)
 - [3.5. The Significance of Key Length](#)
- [4. TRUST MODELS](#)
 - [4.1. PGP Web of Trust](#)
 - [4.2. Kerberos](#)
 - [4.3. Public Key Certificates and Certification Authorities](#)
 - [4.4. Summary](#)
- [5. CRYPTOGRAPHIC ALGORITHMS IN ACTION](#)
 - [5.1. Password Protection](#)
 - [5.2. Some of the Finer Details of Diffie-Hellman Key Exchange](#)
 - [5.3. Some of the Finer Details of RSA Public-Key Cryptography](#)
 - [5.4. Some of the Finer Details of DES, Breaking DES, and DES Variants](#)
 - [5.5. Pretty Good Privacy \(PGP\)](#)
 - [5.6. IP Security \(IPsec\) Protocol](#)
 - [5.7. The SSL "Family" of Secure Transaction Protocols for the World Wide Web](#)
 - [5.8. Elliptic Curve Cryptography](#)
 - [5.9. The Advanced Encryption Standard and Rijndael](#)
 - [5.10. Cisco's Stream Cipher](#)
- [6. CONCLUSION... OF SORTS](#)
- [7. REFERENCES AND FURTHER READING](#)
- [A. SOME MATH NOTES](#)
 - [A.1. The Exclusive-OR \(XOR\) Function](#)
 - [A.2. The *modulo* Function](#)
- [ABOUT THE AUTHOR](#)

FIGURES

1. [Three types of cryptography: secret-key, public key, and hash function.](#)
2. [Sample application of the three cryptographic techniques for secure communication.](#)
3. [Kerberos architecture.](#)
4. [GTE Cybertrust Global Root-issued certificate \(Netscape Navigator\).](#)
5. [Sample entries in Unix/Linux password files.](#)
6. [DES enciphering algorithm.](#)
7. [A PGP signed message.](#)
8. [A PGP encrypted message.](#)
9. [The decrypted message.](#)
10. [IPsec Authentication Header format.](#)
11. [IPsec Encapsulating Security Payload format.](#)
12. [IPsec tunnel and transport modes for AH.](#)
13. [IPsec tunnel and transport modes for ESP.](#)
14. [SSL v3 configuration screen \(Netscape Navigator\).](#)
15. [SSL/TLS protocol handshake.](#)
16. [Elliptic curve addition.](#)
17. [AES pseudocode.](#)

TABLES

1. [Minimum Key Lengths for Symmetric Ciphers.](#)
2. [Contents of an X.509 V3 Certificate.](#)
3. [Other Crypto Algorithms and Systems of Note.](#)
4. [ECC and RSA Key Comparison.](#)

1. INTRODUCTION

Does increased security provide comfort to paranoid people? Or does security provide some very basic protections that we are naive to believe that we don't need? During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with.

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography, which is the focus of this chapter. But it is important to note that while cryptography is *necessary* for secure communications, it is not by itself *sufficient*. The reader is advised, then, that the topics covered in this chapter only describe the first of many steps necessary for better security in any number of situations.

This paper has two major purposes. The first is to define some of the terms and concepts behind basic cryptographic methods, and to offer a way to compare the myriad cryptographic schemes in use today. The second is to provide some real examples of cryptography in use today.

I would like to say at the outset that this paper is very focused on terms, concepts, and schemes in *current* use and is not a treatise of the whole field. No mention is made here about pre-computerized crypto schemes, the difference between a substitution and transposition cipher, cryptanalysis, or other history. Interested readers should check out some of the books in the bibliography below for this detailed — and interesting! — background information.

2. THE PURPOSE OF CRYPTOGRAPHY

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about *any* network, particularly the Internet.

Within the context of any application-to-application communication, there are some specific security requirements, including:

- *Authentication*: The process of proving one's identity. (The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.)
- *Privacy/confidentiality*: Ensuring that no one can read the message except the intended receiver.
- *Integrity*: Assuring the receiver that the received message has not been altered in any way from the original.
- *Non-repudiation*: A mechanism to prove that the sender really sent this message.

Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn (usually) be decrypted into usable plaintext.

In many of the descriptions below, two communicating parties will be referred to as Alice and Bob; this is the common nomenclature in the crypto field and literature to make it easier to identify the communicating parties. If there is a third or fourth party to the communication, they will be referred to as Carol and Dave. Mallory is a malicious party, Eve is an eavesdropper, and Trent is a trusted

third party.

3. TYPES OF CRYPTOGRAPHIC ALGORITHMS

There are several ways of classifying cryptographic algorithms. For purposes of this paper, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms that will be discussed are (Figure 1):

- Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption
- Public Key Cryptography (PKC): Uses one key for encryption and another for decryption
- Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

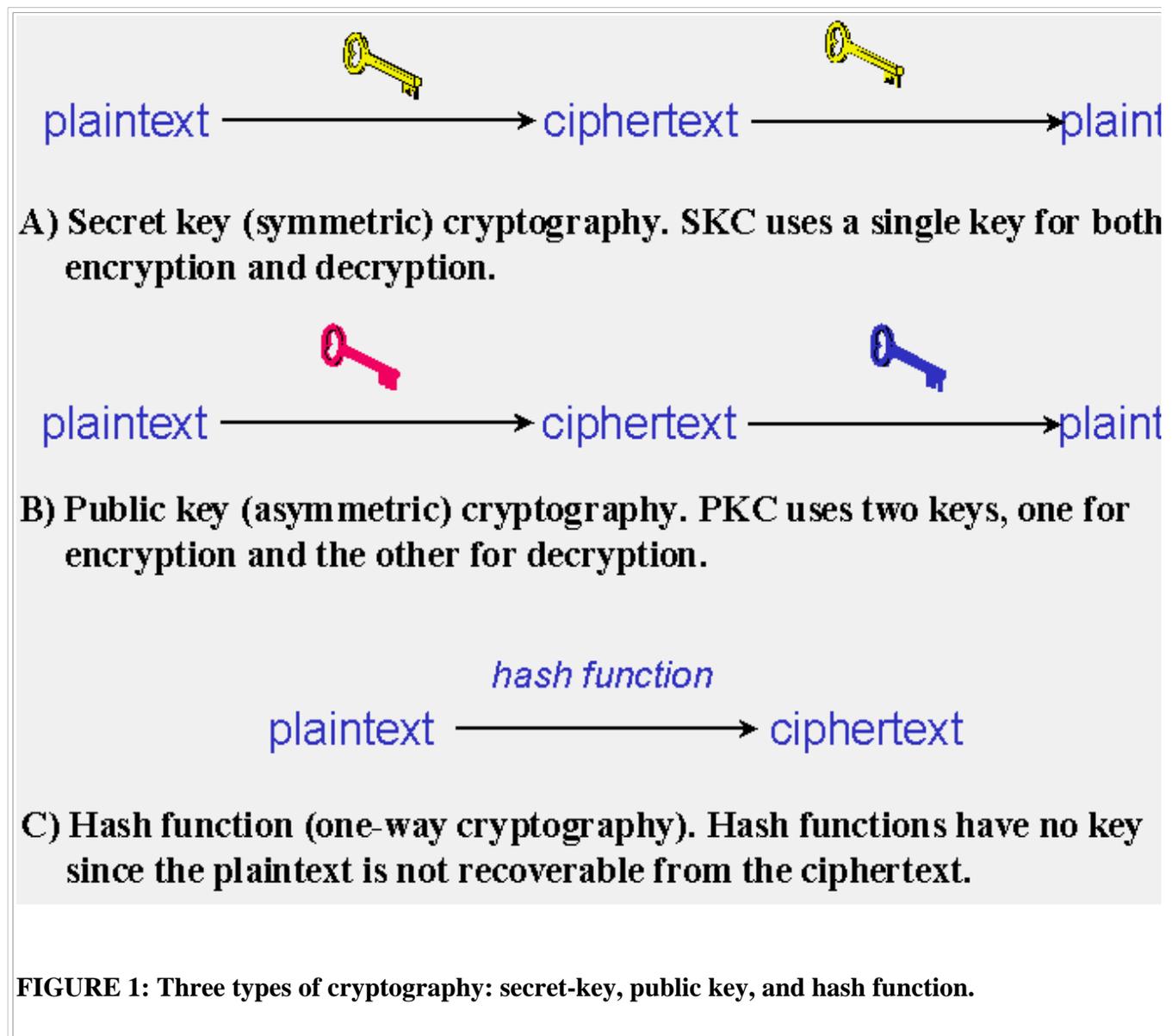


FIGURE 1: Three types of cryptography: secret-key, public key, and hash function.

3.1. Secret Key Cryptography

With *secret key cryptography*, a single key is used for both encryption and decryption. As shown in Figure 1A, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called *symmetric encryption*.

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Secret key cryptography schemes are generally categorized as being either *stream ciphers* or *block ciphers*. Stream ciphers operate on a single bit (byte or computer word) at a time and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

Stream ciphers come in several flavors but two are worth mentioning here. *Self-synchronizing stream ciphers* calculate each bit in the keystream as a function of the previous n bits in the keystream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n -bit keystream it is. One problem is error propagation; a garbled bit in transmission will result in n garbled bits at the receiving side. *Synchronous stream ciphers* generate the keystream in a fashion independent of the message stream but by using the same keystream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the keystream will eventually repeat.

Block ciphers can operate in one of several modes; the following four are the most important:

- *Electronic Codebook (ECB) mode* is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a ciphertext block. Two identical plaintext blocks, then, will always generate the same ciphertext block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.
- *Cipher Block Chaining (CBC) mode* adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous ciphertext block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same ciphertext.
- *Cipher Feedback (CFB) mode* is a block cipher implementation as a self-synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting interactive terminal input. If we were using 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the ciphertext is decrypted and the extra bits in the block (i.e., everything above and beyond the one byte) are discarded.
- *Output Feedback (OFB) mode* is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same ciphertext block by using an internal feedback mechanism that is independent of both the plaintext and ciphertext bitstreams.

Secret key cryptography algorithms that are in use today include:

- *Data Encryption Standard (DES)*: The most common SKC scheme used today, DES was designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards and Technology (NIST)] in 1977 for commercial and unclassified government applications. DES is a block-cipher employing a 56-bit key that operates on 64-bit blocks. DES has a complex set of rules and transformations that were designed specifically to yield fast hardware implementations and slow software implementations, although this latter point is becoming less significant today since the speed of computer processors is several orders of magnitude faster today than twenty years ago. IBM also proposed a 112-bit key for DES, which was rejected at the time by the government; the use of 112-bit keys was considered in the 1990s, however, conversion was never seriously considered.

DES is defined in American National Standard X3.92 and three Federal Information Processing Standards (FIPS):

- [FIPS 46-3: DES](#)
- [FIPS 74: Guidelines for Implementing and Using the NBS Data Encryption Standard](#)
- [FIPS 81: DES Modes of Operation](#)

Information about vulnerabilities of DES can be obtained from the [Electronic Frontier Foundation](#).

Two important variants that strengthen DES are:

- *Triple-DES (3DES)*: A variant of DES that employs up to three 56-bit keys and makes three encryption/decryption passes over the block; 3DES is also described in [FIPS 46-3](#) and is the recommended replacement to DES.
- *DESX*: A variant devised by Ron Rivest. By combining 64 additional key bits to the plaintext prior to encryption, effectively increases the keylength to 120 bits.

More detail about DES, 3DES, and DESX can be found below in [Section 5.4](#).

- *Advanced Encryption Standard (AES)*: In 1997, NIST initiated a very public, 4-1/2 year process to develop a new secure cryptosystem for U.S. government applications. The result, the [Advanced Encryption Standard](#), became the official successor to DES in December 2001. AES uses an SKC scheme called [Rijndael](#), a block cipher designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. The algorithm can use a variable block length and key length; the latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits. NIST initially selected Rijndael in October 2000 and formal adoption as the AES standard came in December 2001. [FIPS PUB 197](#) describes a 128-bit block cipher employing a 128-, 192-, or 256-bit key. The AES process and Rijndael algorithm are described in more detail below in [Section 5.9](#).
- *CAST-128/256*: CAST-128, described in [Request for Comments \(RFC\) 2144](#), is a DES-like substitution-permutation crypto algorithm, employing a 128-bit key operating on a 64-bit block. [CAST-256 \(RFC 2612\)](#) is an extension of CAST-128, using a 128-bit block size and a variable length (128, 160, 192, 224, or 256 bit) key. CAST is named for its developers, Carlisle Adams and Stafford Tavares and is available internationally. CAST-256 was one of the Round 1 algorithms in the AES process.
- *International Data Encryption Algorithm (IDEA)*: Secret-key cryptosystem written by Xuejia Lai and James Massey, in 1992 and patented by Ascom; a 64-bit SKC block cipher using a

128-bit key. Also available internationally.

- *Rivest Ciphers* (aka *Ron's Code*): Named for Ron Rivest, a series of SKC algorithms.
 - *RC1*: Designed on paper but never implemented.
 - *RC2*: A 64-bit block cipher using variable-sized keys designed to replace DES. Its code has not been made public although many companies have licensed RC2 for use in their products. Described in [RFC 2268](#).
 - *RC3*: Found to be breakable during development.
 - *RC4*: A stream cipher using variable-sized keys; it is widely used in commercial cryptography products, although it can only be exported using keys that are 40 bits or less in length.
 - *RC5*: A block-cipher supporting a variety of block sizes, key sizes, and number of encryption passes over the data. Described in [RFC 2040](#).
 - *RC6*: An improvement over RC5, RC6 was one of the AES Round 2 algorithms.
- *Blowfish*: A symmetric 64-bit block cipher invented by Bruce Schneier; optimized for 32-bit processors with large data caches, it is significantly faster than DES on a Pentium/PowerPC-class machine. Key lengths can vary from 32 to 448 bits in length. Blowfish, available freely and intended as a substitute for DES or IDEA, is in use in over 80 products.
- *Twofish*: A 128-bit block cipher using 128-, 192-, or 256-bit keys. Designed to be highly secure and highly flexible, well-suited for large microprocessors, 8-bit smart card microprocessors, and dedicated hardware. Designed by a team led by Bruce Schneier and was one of the Round 2 algorithms in the AES process.
- *Camellia*: A secret-key, block-cipher crypto algorithm developed jointly by Nippon Telegraph and Telephone (NTT) Corp. and Mitsubishi Electric Corporation (MEC) in 2000. Camellia has some characteristics in common with AES: a 128-bit block size, support for 128-, 192-, and 256-bit key lengths, and suitability for both software and hardware implementations on common 32-bit processors as well as 8-bit processors (e.g., smart cards, cryptographic hardware, and embedded systems). Also described in [RFC 3713](#). Camellia's application in IPsec is described in [RFC 4312](#).
- *MISTY1*: Developed at Mitsubishi Electric Corp., a block cipher using a 128-bit key and 64-bit blocks, and a variable number of rounds. Designed for hardware and software implementations, and is resistant to differential and linear cryptanalysis. Described in [RFC 2994](#).
- *Secure and Fast Encryption Routine (SAFER)*: Secret-key crypto scheme designed for implementation in software. Versions have been defined for 40-, 64-, and 128-bit keys.
- *KASUMI*: A block cipher using a 128-bit key that is part of the Third-Generation Partnership Project (3gpp), formerly known as the Universal Mobile Telecommunications System (UMTS). KASUMI is the intended confidentiality and integrity algorithm for both message content and signaling data for emerging mobile communications systems.
- *SEED*: A block cipher using 128-bit blocks and 128-bit keys. Developed by the Korea

Information Security Agency (KISA) and adopted as a national standard encryption algorithm in South Korea. Also described in [RFC 4269](#).

- [Skipjack](#): SKC scheme proposed for Capstone. Although the details of the algorithm were never made public, Skipjack was a block cipher using an 80-bit key and 32 iteration cycles per 64-bit block.

3.2. Public-Key Cryptography

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

PKC depends upon the existence of so-called *one-way functions*, or mathematical functions that are easy to compute whereas their inverse function is relatively difficult to compute. Let me give you two simple examples:

1. *Multiplication vs. factorization*: Suppose I tell you that I have two numbers, 9 and 16, and that I want to calculate the product; it should take almost no time to calculate the product, 144. Suppose instead that I tell you that I have a number, 144, and I need you tell me which pair of integers I multiplied together to obtain that number. You will eventually come up with the solution but whereas calculating the product took milliseconds, factoring will take longer because you first need to find the 8 pair of integer factors and then determine which one is the correct pair.
2. *Exponentiation vs. logarithms*: Suppose I tell you that I want to take the number 3 to the 6th power; again, it is easy to calculate $3^6=729$. But if I tell you that I have the number 729 and want you to tell me the two integers that I used, x and y so that $\log_x 729 = y$, it will take you longer to find all possible solutions and select the pair that I used.

While the examples above are trivial, they do represent two of the functional pairs that are used with PKC; namely, the ease of multiplication and exponentiation versus the relative difficulty of factoring and calculating logarithms, respectively. The mathematical "trick" in PKC is to find a *trap door* in the one-way function so that the inverse calculation becomes easy given knowledge of some item of information.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it **does not matter which key is applied first**, but that both keys are required for the process to work (Figure 1B). Because a pair of keys are required, this approach is also called *asymmetric cryptography*.

In PKC, one of the keys is designated the *public key* and may be advertised as widely as the owner wants. The other key is designated the *private key* and is never revealed to another party. It is straight forward to send messages under this scheme. Suppose Alice wants to send Bob a message. Alice encrypts some information using Bob's public key; Bob decrypts the ciphertext using his private key. This method could be also used to prove who sent a message; Alice, for example, could encrypt some plaintext with her private key; when Bob decrypts using Alice's public key, he knows that Alice sent the message and Alice cannot deny having sent the message (*non-repudiation*).

Public-key cryptography algorithms that are in use today for key exchange or digital signatures include:

- **RSA:** The first, and still most common, PKC implementation, named for the three MIT mathematicians who developed it — Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key. The key-pair is derived from a very large number, n , that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an n with roughly twice as many digits as the prime factors. The public key information includes n and a derivative of one of the factors of n ; an attacker cannot determine the prime factors of n (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure. (Some descriptions of PKC erroneously state that RSA's safety is due to the difficulty in *factoring* large prime numbers. In fact, large prime numbers, like small prime numbers, only have two factors!) The ability for computers to factor large numbers, and therefore attack schemes such as RSA, is rapidly improving and systems today can find the prime factors of numbers with more than 200 digits. Nevertheless, if a large number is created from two prime factors that are roughly the same size, there is no known factorization algorithm that will solve the problem in a reasonable amount of time; a 2005 test to factor a 200-digit number took 1.5 years and over 50 years of compute time (see the Wikipedia article on [integer factorization](#).) Regardless, one presumed protection of RSA is that users can easily increase the key size to always stay ahead of the computer processing curve. As an aside, the patent for RSA expired in September 2000 which does not appear to have affected RSA's popularity one way or the other. A detailed example of RSA is presented below in [Section 5.3](#).
- **Diffie-Hellman:** After the RSA algorithm was published, Diffie and Hellman came up with their own algorithm. D-H is used for secret-key key exchange only, and not for authentication or digital signatures. More detail about Diffie-Hellman can be found below in [Section 5.2](#).
- **Digital Signature Algorithm (DSA):** The algorithm specified in NIST's Digital Signature Standard (DSS), provides digital signature capability for the authentication of messages.
- **ElGamal:** Designed by Taher Elgamal, a PKC system similar to Diffie-Hellman and used for key exchange.
- **Elliptic Curve Cryptography (ECC):** A PKC algorithm based upon elliptic curves. ECC can offer levels of security with small keys comparable to RSA and other PKC methods. It was designed for devices with limited compute power and/or memory, such as smartcards and PDAs. More detail about ECC can be found below in [Section 5.8](#). Other references include "[The Importance of ECC](#)" Web page and the "[Online Elliptic Curve Cryptography Tutorial](#)", both from Certicom.
- **Public-Key Cryptography Standards (PKCS):** A set of interoperable standards and guidelines for public-key cryptography, designed by RSA Data Security Inc.
 - [PKCS #1](#): RSA Cryptography Standard (Also [RFC 3447](#))
 - [PKCS #2](#): *Incorporated into PKCS #1.*
 - [PKCS #3](#): Diffie-Hellman Key-Agreement Standard
 - [PKCS #4](#): *Incorporated into PKCS #1.*
 - [PKCS #5](#): Password-Based Cryptography Standard (PKCS #5 V2.0 is also [RFC 2898](#))
 - [PKCS #6](#): Extended-Certificate Syntax Standard (being phased out in favor of

- X.509v3)
 - [PKCS #7](#): Cryptographic Message Syntax Standard (Also [RFC 2315](#))
 - [PKCS #8](#): Private-Key Information Syntax Standard
 - [PKCS #9](#): Selected Attribute Types (Also [RFC 2985](#))
 - [PKCS #10](#): Certification Request Syntax Standard (Also [RFC 2986](#))
 - [PKCS #11](#): Cryptographic Token Interface Standard
 - [PKCS #12](#): Personal Information Exchange Syntax Standard
 - [PKCS #13](#): Elliptic Curve Cryptography Standard
 - PKCS #14: *Pseudorandom Number Generation Standard is no longer available*
 - [PKCS #15](#): Cryptographic Token Information Format Standard
- [Cramer-Shoup](#): A public-key cryptosystem proposed by R. Cramer and V. Shoup of IBM in 1998.
 - [Key Exchange Algorithm \(KEA\)](#): A variation on Diffie-Hellman; proposed as the key exchange method for Capstone.
 - [LUC](#): A public-key cryptosystem designed by P.J. Smith and based on Lucas sequences. Can be used for encryption and signatures, using integer factoring.

For additional information on PKC algorithms, see "[Public-Key Encryption](#)", Chapter 8 in *Handbook of Applied Cryptography*, by A. Menezes, P. van Oorschot, and S. Vanstone (CRC Press, 1996).

A digression: Who invented PKC? I tried to be careful in the first paragraph of this section to state that Diffie and Hellman "first described publicly" a PKC scheme. Although I have categorized PKC as a two-key system, that has been merely for convenience; the real criteria for a PKC scheme is that it allows two parties to exchange a secret even though the communication with the shared secret might be overheard. There seems to be no question that Diffie and Hellman were first to publish; their method is described in the classic paper, "New Directions in Cryptography," published in the November 1976 issue of *IEEE Transactions on Information Theory*. As shown below, Diffie-Hellman uses the idea that finding logarithms is relatively harder than exponentiation. And, indeed, it is the precursor to modern PKC which does employ two keys. Rivest, Shamir, and Adleman described an implementation that extended this idea in their paper "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," published in the February 1978 issue of the *Communications of the ACM (CACM)*. Their method, of course, is based upon the relative ease of finding the product of two large prime numbers compared to finding the prime factors of a large number.

Some sources, though, credit Ralph Merkle with first describing a system that allows two parties to share a secret although it was not a two-key system, per se. A *Merkle Puzzle* works where Alice creates a large number of encrypted keys, sends them all to Bob so that Bob chooses one at random and then lets Alice know which he has selected. An eavesdropper will see all of the keys but can't learn which key Bob has selected (because he has encrypted the response with the chosen key). In this case, Eve's effort to break in is the square of the effort of Bob to choose a key. While this difference may be small it is often sufficient. Merkle apparently took a computer science course at UC Berkeley in 1974 and described his method, but had difficulty making people understand it; frustrated, he dropped the course. Meanwhile, he submitted the paper "Secure Communication Over Insecure Channels" which was

published in the *CACM* in April 1978; Rivest et al.'s paper even makes reference to it. Merkle's method certainly wasn't published first, but did he have the idea first?

An interesting question, maybe, but who really knows? For some time, it was a quiet secret that a team at the UK's Government Communications Headquarters (GCHQ) had first developed PKC in the early 1970s. Because of the nature of the work, GCHQ kept the original memos classified. In 1997, however, the GCHQ changed their posture when they realized that there was nothing to gain by continued silence. Documents show that a GCHQ mathematician named James Ellis started research into the key distribution problem in 1969 and that by 1975, Ellis, Clifford Cocks, and Malcolm Williamson had worked out all of the fundamental details of PKC, yet couldn't talk about their work. (They were, of course, barred from challenging the RSA patent!) After more than 20 years, Ellis, Cocks, and Williamson have begun to get their due credit.

And the National Security Agency (NSA) claims to have knowledge of this type of algorithm as early as 1966 but there is no supporting documentation... yet. So this really was a digression...

3.3. Hash Functions

Hash functions, also called *message digests* and *one-way encryption*, are algorithms that, in some sense, use no key (Figure 1C). Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a *digital fingerprint* of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

Hash algorithms that are in common use today include:

- *Message Digest (MD) algorithms*: A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message.
 - *MD2 (RFC 1319)*: Designed for systems with limited memory, such as smart cards.
 - *MD4 (RFC 1320)*: Developed by Rivest, similar to MD2 but designed specifically for fast processing in software.
 - *MD5 (RFC 1321)*: Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. MD5 has been implemented in a large number of products although several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996.
- *Secure Hash Algorithm (SHA)*: Algorithm for NIST's Secure Hash Standard (SHS). SHA-1 produces a 160-bit hash value and was originally published as FIPS 180-1 and [RFC 3174](#). [FIPS 180-2](#) describes five algorithms in the SHS: SHA-1 plus SHA-224, SHA-256, SHA-384, and SHA-512 which can produce hash values that are 224, 256, 384, or 512 bits in length, respectively. SHA-224, -256, -384, and -512 are also described in [RFC 4634](#).
- [RIPEMD](#): A series of message digests that initially came from the RIPE (RACE Integrity

Primitives Evaluation) project. [RIPEMD-160](#) was designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, and optimized for 32-bit processors to replace the then-current 128-bit hash functions. Other versions include RIPEMD-256, RIPEMD-320, and RIPEMD-128.

- [HAVAL \(HAsH of VArIable Length\)](#): Designed by Y. Zheng, J. Pieprzyk and J. Seberry, a hash algorithm with many levels of security. HAVAL can create hash values that are 128, 160, 192, 224, or 256 bits in length.
- [Whirlpool](#): A relatively new hash function, designed by V. Rijmen and P.S.L.M. Barreto. Whirlpool operates on messages less than 2^{256} bits in length, and produces a message digest of 512 bits. The design of this hash function is very different than that of MD5 and SHA-1, making it immune to the same attacks as on those hashes (see below).
- [Tiger](#): Designed by Ross Anderson and Eli Biham, Tiger is designed to be secure, run efficiently on 64-bit processors, and easily replace MD4, MD5, SHA and SHA-1 in other applications. Tiger/192 produces a 192-bit output and is compatible with 64-bit architectures; Tiger/128 and Tiger/160 produce the first 128 and 160 bits, respectively, to provide compatibility with the other hash functions mentioned above.

Hash functions are sometimes misunderstood and some sources claim that no two files can have the same hash value. This is, in fact, not correct. Consider a hash function that provides a 128-bit hash value. There are, obviously, 2^{128} possible hash values. But there are a lot more than 2^{128} possible files. Therefore, there have to be multiple files — in fact, there have to be an infinite number of files! — that can have the same 128-bit hash value.

The difficulty is *finding* two files with the same hash! What is, indeed, very hard to do is to try to create a file that has a given hash value so as to force a hash value collision — which is the reason that hash functions are used extensively for information security and computer forensics applications. Alas, researchers in 2004 found that *practical* collision attacks could be launched on MD5, SHA-1, and other hash algorithms. At this time, there is no obvious successor to MD5 and SHA-1 that could be put into use quickly; there are so many products using these hash functions that it could take many years to flush out all use of 128- and 160-bit hashes. Readers interested in this problem should read the following:

- Burr, W. (2006, Match/April). Cryptographic hash standards: Where do we go from here? *IEEE Security & Privacy*, 4(2), 88-91.
- Gutman, P., Naccache, D., & Palmer, C.C. (2005, May/June). When hashes collide. *IEEE Security & Privacy*, 3(3), 68-71.
- Klima, V. (March 2005) "[Finding MD5 Collisions - a Toy For a Notebook.](#)"
- Thompson, E. (2005, February). MD5 collisions and the impact on computer forensics. *Digital Investigation*, 2(1), 36-40.
- Wang, X., Feng, D., Lai, X., & Yu, H. (August 2004). "[Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.](#)"
- Wang, X., Yin, Y.L., & Yu, H. (February 2005). "[Collision Search Attacks on SHA1.](#)"

An excellent review of the situation with hash collisions can be found in [RFC 4270](#) (by P. Hoffman and B. Schneier, November 2005). And for additional information on hash functions, see David Hopwood's [MessageDigest Algorithms](#) page.

Certain extensions of hash functions are used for a variety of information security and digital forensics applications, such as:

- *Hash libraries* are sets of hash values corresponding to known files. A hash library of known good files, for example, might be a set of files known to be a part of an operating system, while a hash library of known bad files might be of a set of known child pornographic images.
- *Rolling hashes* refer to a set of hash values that are computed based upon a fixed-length "sliding window" through the input. As an example, a hash value might be computed on bytes 1-10 of a file, then on bytes 2-11, 3-12, 4-13, etc.
- *Fuzzy hashes* are an area of intense research and represent hash values that represent two inputs that are similar. Fuzzy hashes are used to detect documents, images, or other files that are close to each other with respect to content.

3.4. Why Three Encryption Techniques?

So, why are there so many different types of cryptographic schemes? Why can't we do everything we need with just one?

The answer is that each scheme is optimized for some specific application(s). Hash functions, for example, are well-suited for ensuring data integrity because any change made to the contents of a message will result in the receiver calculating a different hash value than the one placed in the transmission by the sender. Since it is highly unlikely that two different messages will yield the same hash value, data integrity is ensured to a high degree of confidence.

Secret key cryptography, on the other hand, is ideally suited to encrypting messages. The sender can generate a *session key* on a per-message basis to encrypt the message; the receiver, of course, needs the same session key to decrypt the message.

Key exchange, of course, is a key application of public-key cryptography (no pun intended). Asymmetric schemes can also be used for non-repudiation; if the receiver can obtain the session key encrypted with the sender's private key, then only this sender could have sent the message. Public-key cryptography could, theoretically, also be used to encrypt messages although this is rarely done because secret-key cryptography operates about 1000 times faster than public-key cryptography.

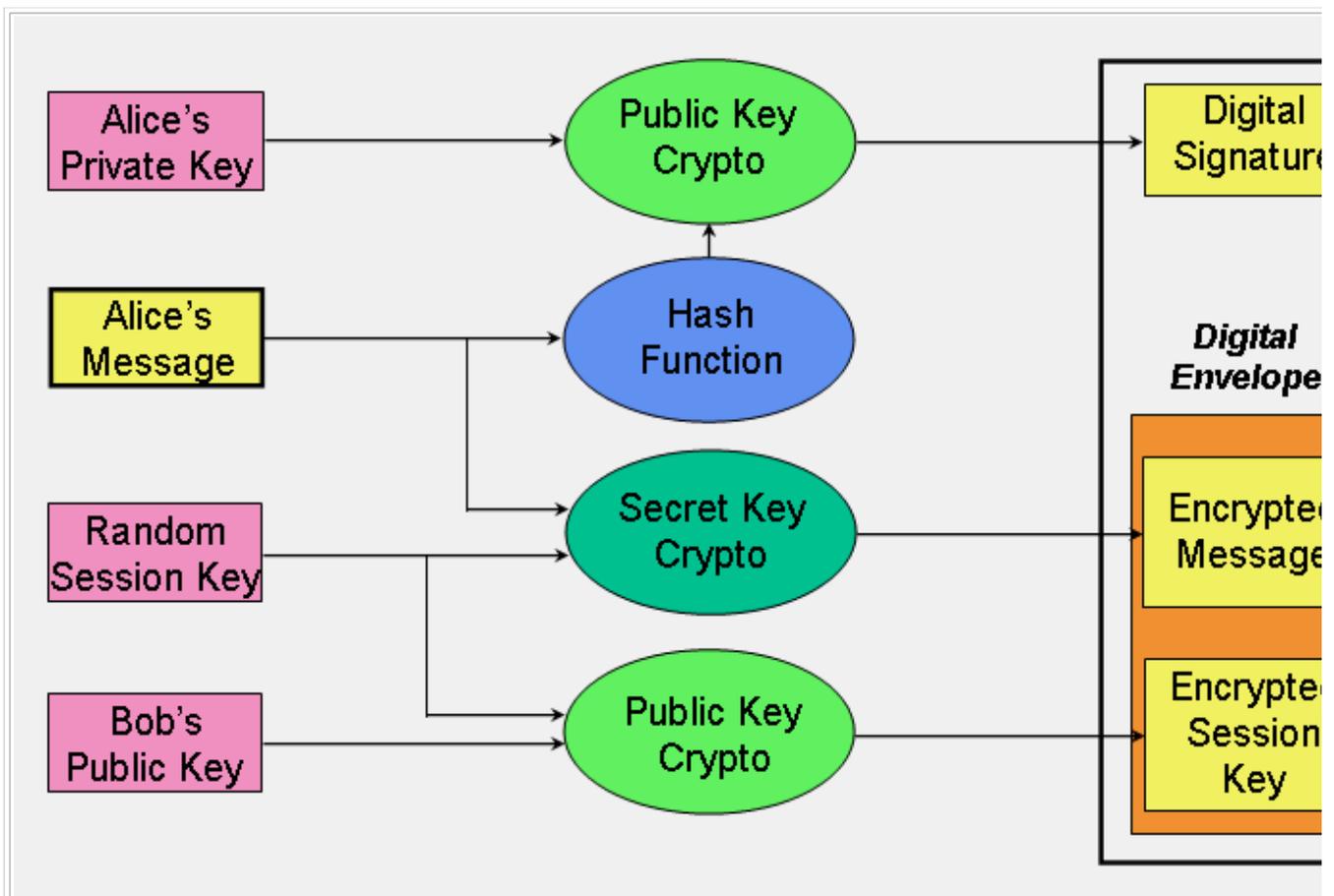


FIGURE 2: Sample application of the three cryptographic techniques for secure communication.

Figure 2 puts all of this together and shows how a *hybrid cryptographic* scheme combines all of these functions to form a secure transmission comprising *digital signature* and *digital envelope*. In this example, the sender of the message is Alice and the receiver is Bob.

A digital envelope comprises an encrypted message and an encrypted session key. Alice uses secret key cryptography to encrypt her message using the *session key*, which she generates at random with each session. Alice then encrypts the session key using Bob's public key. The encrypted message and encrypted session key together form the digital envelope. Upon receipt, Bob recovers the session secret key using his private key and then decrypts the encrypted message.

The digital signature is formed in two steps. First, Alice computes the hash value of her message; next, she encrypts the hash value with her private key. Upon receipt of the digital signature, Bob recovers the hash value calculated by Alice by decrypting the digital signature with Alice's public key. Bob can then apply the hash function to Alice's original message, which he has already decrypted (see previous paragraph). If the resultant hash value is not the same as the value supplied by Alice, then Bob knows that the message has been altered; if the hash values are the same, Bob should believe that the message he received is identical to the one that Alice sent.

This scheme also provides nonrepudiation since it proves that Alice sent the message; if the hash value recovered by Bob using Alice's public key proves that the message has not been altered, then only Alice could have created the digital signature. Bob also has proof that he is the intended

receiver; if he can correctly decrypt the message, then he must have correctly decrypted the session key meaning that his is the correct private key.

3.5. The Significance of Key Length

In a recent article in the industry literature (circa 9/98), a writer made the claim that 56-bit keys do not provide as sufficient protection for DES today as they did in 1975 because computers are 1000 times faster today than in 1975. Therefore, the writer went on, we should be using 56,000-bit keys today instead of 56-bit keys to provide adequate protection. The conclusion was then drawn that because 56,000-bit keys are infeasible (*true*), we should accept the fact that we have to live with weak cryptography (*false!*). The major error here is that the writer did not take into account that the number of possible key values double whenever a single bit is added to the key length; thus, a 57-bit key has twice as many values as a 56-bit key (because 2^{57} is two times 2^{56}). In fact, a 66-bit key would have 1024 times the possible values as a 56-bit key.

But this does bring up the issue, what is the precise significance of key length as it affects the level of protection?

In cryptography, size does matter. The larger the key, the harder it is to crack a block of encrypted data. The reason that large keys offer more protection is almost obvious; computers have made it easier to attack ciphertext by using brute force methods rather than by attacking the mathematics (which are generally well-known anyway). With a brute force attack, the attacker merely generates every possible key and applies it to the ciphertext. Any resulting plaintext that makes sense offers a candidate for a legitimate key. This was the basis, of course, of the EFF's attack on DES.

Until the mid-1990s or so, brute force attacks were beyond the capabilities of computers that were within the budget of the attacker community. Today, however, significant compute power is commonly available and accessible. General purpose computers such as PCs are already being used for brute force attacks. For serious attackers with money to spend, such as some large companies or governments, Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuits (ASIC) technology offers the ability to build specialized chips that can provide even faster and cheaper solutions than a PC. Consider that an AT&T ORCA chip (FPGA) costs \$200 and can test 30 million DES keys per second, while a \$10 ASIC chip can test 200 million DES keys per second (compared to a PC which might be able to test 40,000 keys per second).

The table below shows what DES key sizes are needed to protect data from attackers with different time and financial resources. This information is not merely academic; one of the basic tenets of any security system is to have an idea of *what* you are protecting and *from who* are you protecting it! The table clearly shows that a 40-bit key is essentially worthless today against even the most unsophisticated attacker. On the other hand, 56-bit keys are fairly strong unless you might be subject to some pretty serious corporate or government espionage. But note that even 56-bit keys are declining in their value and that the times in the table (1995 data) are worst cases.

TABLE 1. Minimum Key Lengths for Symmetric Ciphers.

Type of Attacker	Budget	Tool	Time and Cost Per Key Recovered		Key Length Needed For Protection In Late-1995
			40 bits	56 bits	
Pedestrian Hacker	Tiny	Scavanged computer time	1 week	Infeasible	45
	\$400	FPGA	5 hours (\$0.08)	38 years (\$5,000)	50
Small Business	\$10,000	FPGA	12 minutes (\$0.08)	18 months (\$5,000)	55
Corporate Department	\$300K	FPGA	24 seconds (\$0.08)	19 days (\$5,000)	60
		ASIC	0.18 seconds (\$0.001)	3 hours (\$38)	
Big Company	\$10M	FPGA	7 seconds (\$0.08)	13 hours (\$5,000)	70
		ASIC	0.005 seconds (\$0.001)	6 minutes (\$38)	
Intelligence Agency	\$300M	ASIC	0.0002 seconds (\$0.001)	12 seconds (\$38)	75

So, how big is big enough? DES, invented in 1975, is still in use today, nearly 25 years later. If we take that to be a design criteria (i.e., a 20-plus year lifetime) and we believe Moore's Law ("computing power doubles every 18 months"), then a key size extension of 14 bits (i.e., a factor of more than 16,000) should be adequate. The 1975 DES proposal suggested 56-bit keys; by 1995, a 70-bit key would have been required to offer equal protection and an 85-bit key will be necessary by 2015.

The discussion above suggests that a 128- or 256-bit key for SKC will suffice for some time because that key length keeps us ahead of the brute force capabilities of the attackers. While a large key is good, a huge key may not always be better. That is, many public-key cryptosystems use 1024- or 2048-bit keys; expanding the key to 4096 bits probably doesn't add any protection at this time but it does add significantly to processing time.

The most effective large-number factoring methods today use a mathematical Number Field Sieve to find a certain number of relationships and then uses a matrix operation to solve a linear equation to produce the two prime factors. The sieve step actually involves a large number of operations of operations that can be performed in parallel; solving the linear equation, however, requires a supercomputer. Indeed, finding the solution to the RSA-140 challenge in February 1999 — factoring a 140-digit (465-bit) prime number — required 200 computers across the Internet about 4 weeks for the first step and a Cray computer 100 hours and 810 MB of memory to do the second

step.

In early 1999, Shamir (of RSA fame) described a new machine that could increase factorization speed by 2-3 orders of magnitude. Although no detailed plans were provided nor is one known to have been built, the concepts of [TWINKLE \(The Weizmann Institute Key Locating Engine\)](#) could result in a specialized piece of hardware that would cost about \$5000 and have the processing power of 100-1000 PCs. There still appear to be many engineering details that have to be worked out before such a machine could be built. Furthermore, the hardware improves the sieve step only; the matrix operation is not optimized at all by this design and the complexity of this step grows rapidly with key length, both in terms of processing time and memory requirements. Nevertheless, this plan conceptually puts 512-bit keys within reach of being factored. Although most PKC schemes allow keys that are 1024 bits and longer, Shamir claims that 512-bit RSA keys "protect 95% of today's E-commerce on the Internet." (See Bruce Schneier's [Crypto-Gram \(May 15, 1999\)](#) for more information, as well as the comments from [RSA Labs.](#))

It is also interesting to note that while cryptography is good and strong cryptography is better, long keys may disrupt the nature of the randomness of data files. Shamir and van Someren ("[Playing hide and seek with stored keys](#)") have noted that a new generation of viruses can be written that will find files encrypted with long keys, making them easier to find by intruders and, therefore, more prone to attack.

Finally, U.S. government policy has tightly controlled the export of crypto products since World War II. Until recently, export outside of North America of cryptographic products using keys greater than 40 bits in length was prohibited, which made those products essentially worthless in the marketplace, particularly for electronic commerce. More recently, the U.S. Commerce Department relaxed the regulations, allowing the general export of 56-bit SKC and 1024-bit PKC products (certain sectors, such as health care and financial, allow the export of products with even larger keys). The Commerce Department's Bureau of Export Administration maintains a [Commercial Encryption Export Controls](#) web page with more information. The potential impact of this policy on U.S. businesses is well beyond the scope of this paper.

Much of the discussion above, including the table, are based on the paper "[Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security](#)" by M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener.

On a related topic, public key crypto schemes can be used for several purposes, including key exchange, digital signatures, authentication, and more. In those PKC systems used for SKC key exchange, the PKC key lengths are chosen so to be resistant to some selected level of attack. The length of the secret keys exchanged via that system have to have at least the same level of attack resistance. Thus, the three parameters of such a system — system strength, secret key strength, and public key strength — must be matched. This topic is explored in more detail in *Determining Strengths For Public Keys Used For Exchanging Symmetric Keys* ([RFC 3766](#)).

4. TRUST MODELS

Secure use of cryptography requires trust. While secret key cryptography can ensure message confidentiality and hash codes can ensure integrity, none of this works without trust. In SKC, Alice and Bob had to share a secret key. PKC solved the secret distribution problem, but how does Alice really know that Bob is who he says he is? Just because Bob has a public and private key, and purports to be "Bob," how does Alice know that a malicious person (Mallory) is not pretending to be Bob?

There are a number of *trust models* employed by various cryptographic schemes. This section will explore three of them:

- The web of trust employed by Pretty Good Privacy (PGP) users, who hold their own set of trusted public keys.
- Kerberos, a secret key distribution scheme using a trusted third party.
- Certificates, which allow a set of trusted third parties to authenticate each other and, by implication, each other's users.

Each of these trust models differs in complexity, general applicability, scope, and scalability.

4.1. PGP Web of Trust

Pretty Good Privacy (described more below in [Section 5.5](#)) is a widely used private e-mail scheme based on public key methods. A PGP user maintains a local keyring of all their known and trusted public keys. The user makes their own determination about the trustworthiness of a key using what is called a "web of trust."

If Alice needs Bob's public key, Alice can ask Bob for it in another e-mail or, in many cases, download the public key from an advertised server; this server might be a well-known PGP key repository or a site that Bob maintains himself. In fact, Bob's public key might be stored or listed in many places. (The author's public key, for example, can be found at http://www.garykessler.net/kumquat_pubkey.html.) Alice is prepared to believe that Bob's public key, as stored at these locations, is valid.

Suppose Carol claims to hold Bob's public key and offers to give the key to Alice. How does Alice know that Carol's version of Bob's key is valid or if Carol is actually giving Alice a key that will allow Mallory access to messages? The answer is, "It depends." If Alice trusts Carol and Carol says that she thinks that her version of Bob's key is valid, then Alice *may* — at *her* option — trust that key. And trust is not necessarily transitive; if Dave has a copy of Bob's key and Carol trusts Dave, it does not necessarily follow that Alice trusts Dave even if she does trust Carol.

The point here is that who Alice trusts and how she makes that determination is strictly up to Alice. PGP makes no statement and has no protocol about how one user determines whether they trust another user or not. In any case, encryption and signatures based on public keys can only be used when the appropriate public key is on the user's keyring.

4.2. Kerberos

Kerberos is a commonly used authentication scheme on the Internet. Developed by MIT's Project Athena, Kerberos is named for the three-headed dog who, according to Greek mythology, guards the entrance of Hades (rather than the exit, for some reason!).

Kerberos employs a client/server architecture and provides user-to-server authentication rather than host-to-host authentication. In this model, security and authentication will be based on secret key technology where every host on the network has its own secret key. It would clearly be unmanageable if every host had to know the keys of all other hosts so a secure, trusted host somewhere on the network, known as a Key Distribution Center (KDC), knows the keys for all of the hosts (or at least some of the hosts within a portion of the network, called a *realm*). In this way, when a new node is brought online, only the KDC and the new node need to be configured with the node's key; keys can be distributed physically or by some other secure means.

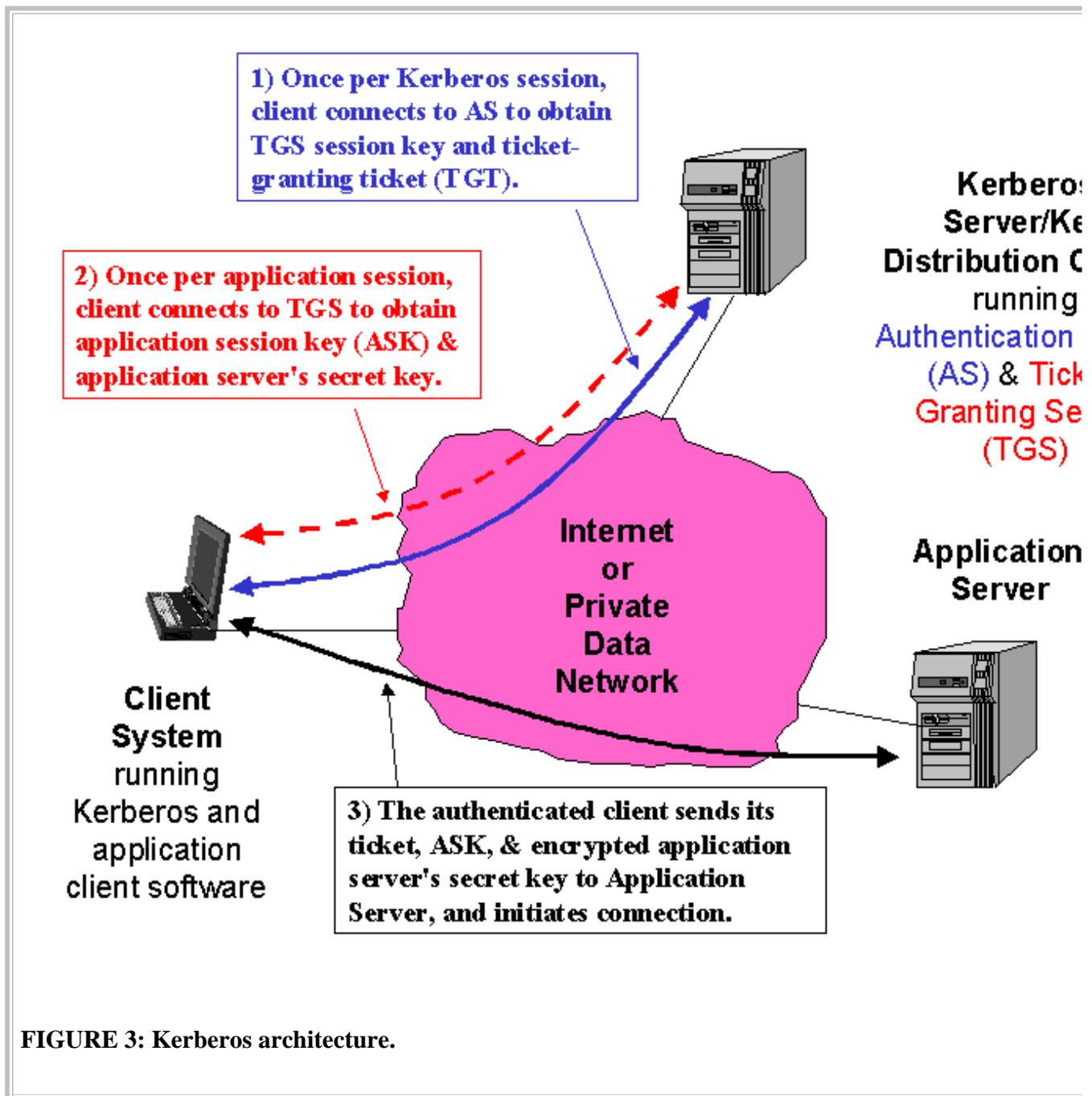


FIGURE 3: Kerberos architecture.

The Kerberos Server/KDC has two main functions (Figure 3), known as the Authentication Server (AS) and Ticket-Granting Server (TGS). The steps in establishing an authenticated session between an application client and the application server are:

1. The Kerberos client software establishes a connection with the Kerberos server's AS function. The AS first authenticates that the client is who it purports to be. The AS then provides the client with a secret key for this login session (the *TGS session key*) and a ticket-granting ticket (TGT), which gives the client permission to talk to the TGS. The ticket has a finite lifetime so that the authentication process is repeated periodically.
2. The client now communicates with the TGS to obtain the Application Server's key so that it (the client) can establish a connection to the service it wants. The client supplies the TGS with the TGS session key and TGT; the TGS responds with an application session key (ASK) and an encrypted form of the Application Server's secret key; this secret key is *never* sent on the network in any other form.

3. The client has now authenticated itself *and* can prove its identity to the Application Server by supplying the Kerberos ticket, application session key, and encrypted Application Server secret key. The Application Server responds with similarly encrypted information to authenticate itself to the client. At this point, the client can initiate the intended service requests (e.g., Telnet, FTP, HTTP, or e-commerce transaction session establishment).

The current shipping version of this protocol is Kerberos V5 (described in [RFC 1510](#)), although Kerberos V4 still exists and is seeing some use. While the details of their operation, functional capabilities, and message formats are different, the conceptual overview above pretty much holds for both. One primary difference is that Kerberos V4 uses only DES to generate keys and encrypt messages, while V5 allows other schemes to be employed (although DES is still the most widely algorithm used).

4.3. Public Key Certificates and Certificate Authorities

Certificates and *Certificate Authorities (CA)* are necessary for widespread use of cryptography for e-commerce applications. While a combination of secret and public key cryptography can solve the business issues discussed above, crypto cannot alone address the trust issues that must exist between a customer and vendor in the very fluid, very dynamic e-commerce relationship. How, for example, does one site obtain another party's public key? How does a recipient determine if a public key really belongs to the sender? How does the recipient know that the sender is using their public key for a legitimate purpose for which they are authorized? When does a public key expire? How can a key be revoked in case of compromise or loss?

The basic concept of a certificate is one that is familiar to all of us. A driver's license, credit card, or SCUBA certification, for example, identify us to others, indicate something that we are authorized to do, have an expiration date, and identify the authority that granted the certificate.

As complicated as this may sound, it really isn't! Consider driver's licenses. I have one issued by the State of Vermont. The license establishes my identity, indicates the type of vehicles that I can operate and the fact that I must wear corrective lenses while doing so, identifies the issuing authority, and notes that I am an organ donor. When I drive outside of Vermont, the other jurisdictions throughout the U.S. recognize the authority of Vermont to issue this "certificate" and they trust the information it contains. Now, when I leave the U.S., everything changes. When I am in Canada and many other countries, they will accept not the Vermont license, *per se*, but *any* license issued in the U.S.; some other countries may not recognize the Vermont driver's license as sufficient *bona fides* that I can drive. This analogy represents the certificate chain, where even certificates carry certificates.

For purposes of electronic transactions, certificates are digital documents. The specific functions of the certificate include:

- *Establish identity*: Associate, or *bind*, a public key to an individual, organization, corporate position, or other entity.
- *Assign authority*: Establish what actions the holder may or may not take based upon this certificate.
- *Secure confidential information* (e.g., encrypting the session's symmetric key for data confidentiality).

Typically, a certificate contains a public key, a name, an expiration date, the name of the authority that issued the certificate (and, therefore, is vouching for the identity of the user), a serial number, any pertinent policies describing how the certificate was issued and/or how the certificate may be used, the digital signature of the certificate issuer, and perhaps other information.

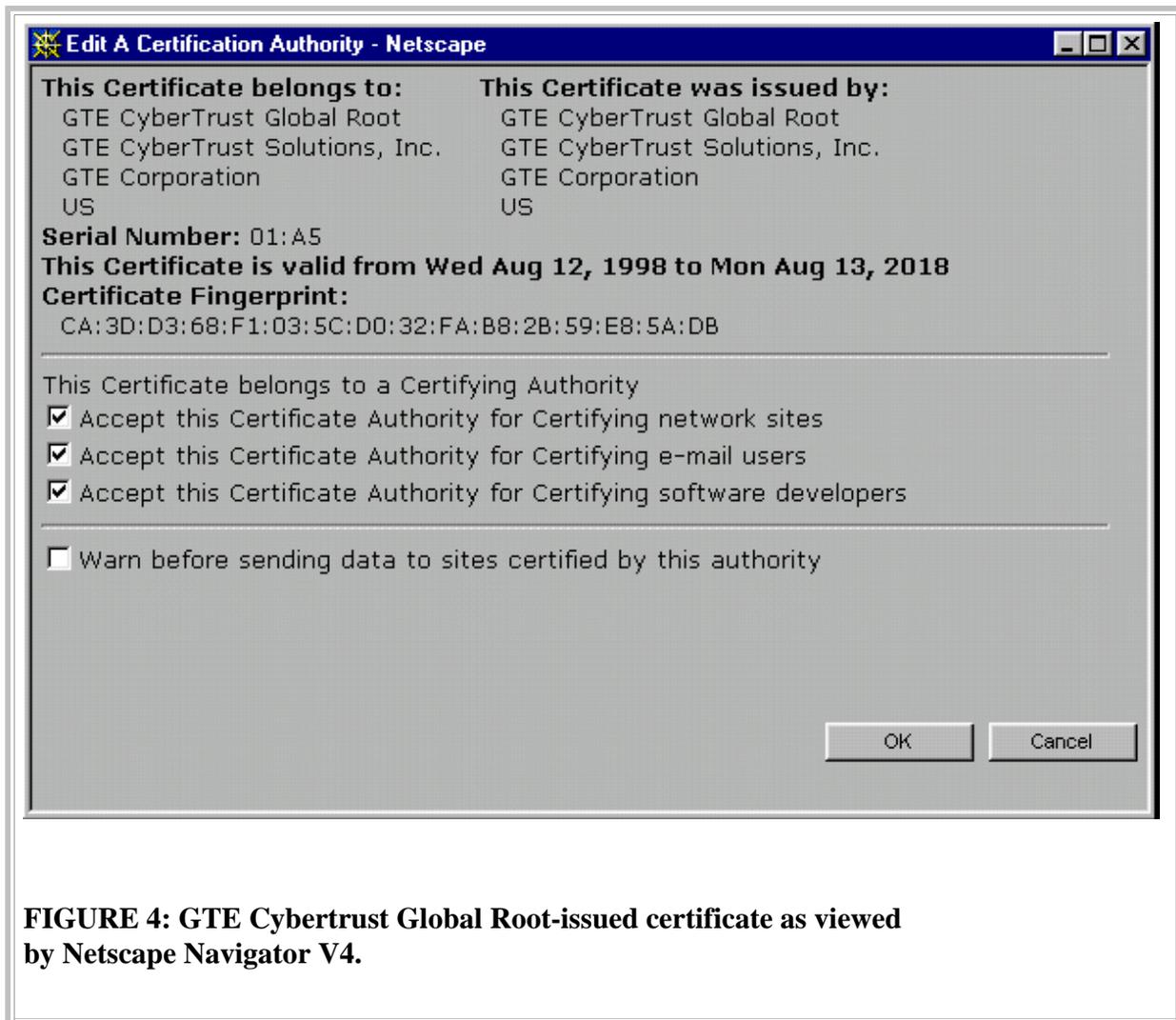


FIGURE 4: GTE Cybertrust Global Root-issued certificate as viewed by Netscape Navigator V4.

A sample abbreviated certificate is shown in Figure 4. This is a typical certificate found in a browser; while this one is issued by GTE Cybertrust, many so-called root-level certificates can be found shipped with browsers. When the browser makes a connection to a secure Web site, the Web server sends its public key certificate to the browser. The browser then checks the certificate's signature against the public key that it has stored; if there is a match, the certificate is taken as valid and the Web site verified by this certificate is considered to be "trusted."

TABLE 2. Contents of an X.509 V3 Certificate.

version number
certificate serial number
signature algorithm identifier
issuer's name and unique identifier
validity (or operational) period
subject's name and unique identifier
subject public key information
standard extensions
certificate appropriate use definition
key usage limitation definition
certificate policy information
other extensions
Application-specific
CA-specific

The most widely accepted certificate format is the one defined in International Telecommunication Union Telecommunication Standardization Sector (ITU-T) Recommendation X.509. Rec. X.509 is a specification used around the world and any applications complying with X.509 can share certificates. Most certificates today comply with X.509 Version 3 and contain the information listed in Table 2.

Certificate authorities are the repositories for public-keys and can be any agency that issues certificates. A company, for example, may issue certificates to its employees, a college/university to its students, a store to its customers, an Internet service provider to its users, or a government to its constituents.

When a sender needs an intended receiver's public key, the sender must get that key from the receiver's CA. That scheme is straight-forward if the sender and receiver have certificates issued by the same CA. If not, how does the sender know to *trust* the foreign CA? One industry wag has noted, about trust: "You are either born with it or have it granted upon you." Thus, some CAs will be trusted because they are known to be reputable, such as the CAs operated by AT&T, BBN, Canada Post Corp., CommerceNet, [GTE Cybertrust](#), MCI, Nortel [EnTrust](#), [Thawte](#), the U.S. Postal Service, and [VeriSign](#). CAs, in turn, form trust relationships with other CAs. Thus, if a user queries a foreign CA for information, the user may ask to see a list of CAs that establish a "chain of trust" back to the user.

One major feature to look for in a CA is their identification policies and procedures. When a user generates a key pair and forwards the public key to a CA, the CA has to check the sender's identification and takes any steps necessary to assure itself that the request is really coming from the advertised sender. Different CAs have different identification policies and will, therefore, be trusted differently by other CAs. Verification of identity is just of many issues that are part of a CA's Certification Practice Statement (CPS) and policies; other issues include how the CA protects the public keys in its care, how lost or compromised keys are revoked, and how the CA protects its own private keys.

4.4. Summary

The paragraphs above describe three very different trust models. It is hard to say that any one is

better than the others; it depend upon your application. One of the biggest and fastest growing applications of cryptography today, though, is electronic commerce (e-commerce), a term that itself begs for a formal definition.

PGP's web of trust is easy to maintain and very much based on the reality of users as people. The model, however, is limited; just how many public keys can a single user reliably store and maintain? And what if you are using the "wrong" computer when you want to send a message and can't access your keyring? How easy it is to revoke a key if it is compromised? PGP may also not scale well to an e-commerce scenario of secure communication between total strangers on short-notice.

Kerberos overcomes many of the problems of PGP's web of trust, in that it is scalable and its scope can be very large. However, it also requires that the Kerberos server have *a priori* knowledge of all client systems prior to any transactions, which makes it unfeasible for "hit-and-run" client/server relationships as seen in e-commerce.

Certificates and the collection of CAs will form a Public Key Infrastructure (PKI). In the early days of the Internet, every host had to maintain a list of every other host; the Domain Name System (DNS) introduced the idea of a distributed database for this purpose and the DNS is one of the key reasons that the Internet has grown as it has. A PKI will fill a similar void in the e-commerce and PKC realm.

While certificates and the benefits of a PKI are most often associated with electronic commerce, the applications for PKI are much broader and include secure electronic mail, payments and electronic checks, Electronic Data Interchange (EDI), secure transfer of Domain Name System (DNS) and routing information, electronic forms, and digitally signed documents. A single "global PKI" is still many years away, that is the ultimate goal of today's work as international electronic commerce changes the way in which we do business in a similar way in which the Internet has changed the way in which we communicate.

5. CRYPTOGRAPHIC ALGORITHMS IN ACTION

The paragraphs above have provided an overview of the different types of cryptographic algorithms, as well as some examples of some available protocols and schemes. Table 3 provides an even longer list of some of the schemes employed today for a variety of functions, most notably electronic commerce. The paragraphs below will show several real cryptographic applications that many of us employ (knowingly or not) everyday; for password protection and private communication.