



**Postgraduate Diploma
in
Strategic Business Information Technology**

**Module 4
Computer Networking and Management**

**Chapter 7
Network Security**

© NCC Education Limited, 2003

Modification History

Revision	Date	Revision Description
V1.0	January 2003	For issue

© NCC Education Limited, 2003

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Contents

Introduction – Network Security	5
Security in Computer Networks	5
Secure Communication	6
Secrecy.....	7
Authentication	7
Message Integrity	7
Network Security Considerations in the Internet	8
Principles of Cryptography.....	10
Symmetric Key Cryptography.....	11
Data Encryption Standard (DES)	12
Public Key Encryption	14
Why Does RSA Work?	17
Generating Digital Signatures	20
Certification Authorities	23
The Roles of a CA	23
Principles of Secure E-Mail.....	24
Case History	27
Phil Zimmermann and PGP	27
Secure Sockets Layer.....	28
Features Provided by SSL	29
How SSL Works.....	29
The Limitations of SSL in Internet Commerce	29
Secure Electronic Transactions (SET).....	30
Summary.....	32

Introduction – Network Security

Visual 1 (NCC Course Title Visual)

Visual 2

Network security

Foundations:

- What is security?
- Cryptography
- Authentication
- Message integrity
- Key distribution and certification

Security in practice:

- Application layer: secure e-mail
- Transport layer: internet commerce, SSL, SET
- Network layer: IP security

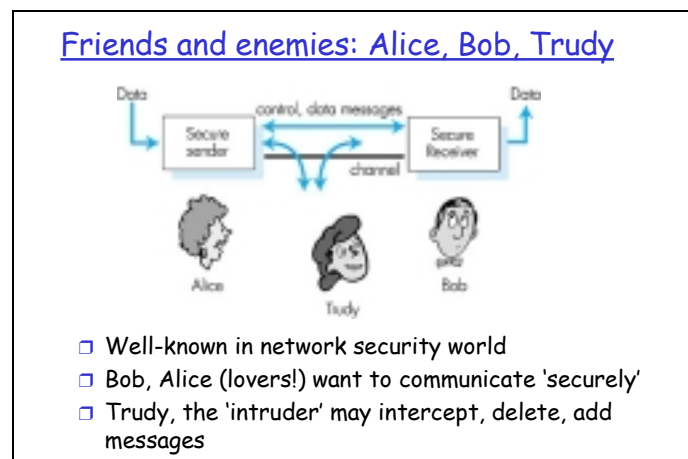
Security in Computer Networks

The first part presents various principles underlying secure communication. We cover cryptographic techniques for coding and decoding data, including both symmetric key cryptography and public key cryptography. We examine DES as a case study for symmetric key cryptography and RSA as a case study for public key cryptography.

We examine authentication, and develop a series of increasingly sophisticated authentication protocols to ensure that a conversant is indeed who he/she claims to be. We find that symmetric and public key cryptography play an important role in authentication as well as in secrecy.

We examine digital signatures and message digests – a shorthand way for signing digital documents. We then look at a series of security applications that make use of these cryptographic techniques. We discuss how these techniques can provide sender authentication and secrecy in e-mail. We examine the PGP protocol as a case study for secure e-mail.

We then look at two popular electronic commerce security protocols, SSL and SET. We conclude this chapter by discussing an emerging network-layer security protocol, IPsec.

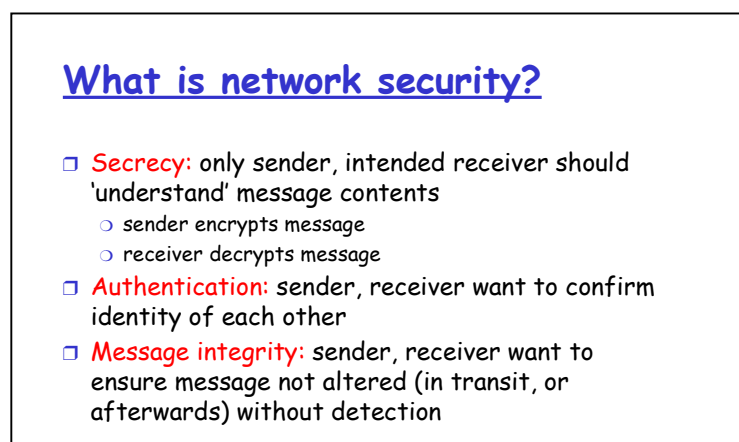
Visual 3

Let us introduce Alice and Bob, two people who want to communicate 'securely'. This being a networking text, we should remark that Alice and Bob may be:

- two routers that want to securely exchange routing tables;
- two hosts that want to establish a secure transport connection; or
- two e-mail applications that want to exchange secure e-mail.

Alice and Bob are well-known fixtures in the security community, perhaps because their names are more fun than a generic entity named 'A' that wants to securely communicate with a generic entity named 'B'. Intimate personal affairs, wartime communication, and business transactions are the commonly cited human needs for secure communications; preferring the first to the latter two, we're happy to use Alice and Bob as our sender and receiver, and imagine them in this first scenario.

Secure Communication

Visual 4

We said that Alice and Bob want to communicate 'securely,' but what precisely does this mean? Certainly, Alice wants only Bob to be able to understand a message that she has sent, even though they are communicating over an 'insecure' medium where an intruder

(Trudy, the intruder) may intercept, read, and perform computations on whatever is transmitted from Alice to Bob. Bob also wants to be sure that the message that he receives from Alice was indeed sent by Alice, and Alice wants to make sure that the person with whom she is communicating is indeed Bob. Alice and Bob also want to make sure that the contents of Alice's message have not been altered in transit. Given these considerations, we can identify the following desirable properties of secure communication:

Secrecy

Only the sender and intended receiver should be able to understand the contents of the transmitted message. Because eavesdroppers may intercept the message, this necessarily requires that the message be somehow encrypted (its data disguised) so that an intercepted message cannot be decrypted (understood) by an interceptor. This aspect of secrecy is probably the most commonly perceived meaning of the term 'secure communication'. Note, however, that this is not only a restricted definition of secure communication but a rather restricted definition of secrecy as well.

For example, Alice might also want the mere fact that she is communicating with Bob (or the timing or frequency of her communications) to be a secret!

Authentication

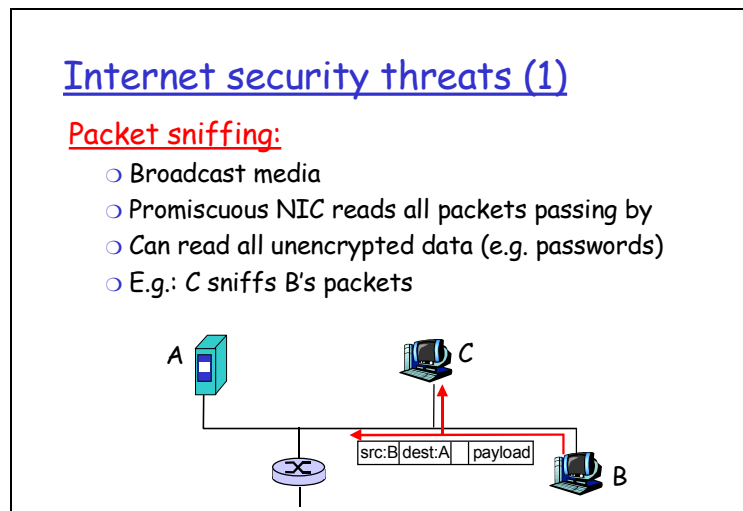
Both the sender and receiver need to confirm the identity of other party involved in the communication – to confirm that the other party is indeed who or what they claim to be. Face-to-face human communication solves this problem easily by visual recognition. When communicating entities exchange messages over a medium where they cannot 'see' the other party, authentication is not so simple. Why, for instance, should you believe that a received e-mail containing a text string saying that the e-mail came from a friend of yours indeed came from that friend? If someone calls you on the phone claiming to be your bank and asking for your account number, secret Personal Identification Number (PIN), and account balances for verification purposes, would you give that information out over the phone? Hopefully not.

Message Integrity

Even if the sender and receiver are able to authenticate each other, they also want to ensure that the content of their communication is not altered, either maliciously or by accident, in transmission.

Having established what we mean by secure communication, let us next consider exactly what is meant by an 'insecure channel'. What information does an intruder have access to, and what actions can be taken on the transmitted data?

Alice, the sender, wants to send data to Bob, the receiver. In order to securely exchange data, while meeting the requirements of secrecy, authentication, and message integrity, Alice and Bob will exchange both control messages and data messages (in much the same way that TCP senders and receivers exchange both control segments and data segments). All or some of these messages will typically be encrypted. A passive intruder can listen to and record the control and data messages on the channel; an active intruder can remove messages from the channel and/or add messages into the channel.

Visual 5**Network Security Considerations in the Internet**

Before delving into the technical aspects of network security in the following sections, let's conclude our introduction by relating our fictitious characters – Alice, Bob, and Trudy – to 'real-world' scenarios in today's Internet.

Let us begin with Trudy, the network intruder.

Can a 'real world' network intruder really listen to and record network messages?

Is it easy to do so?

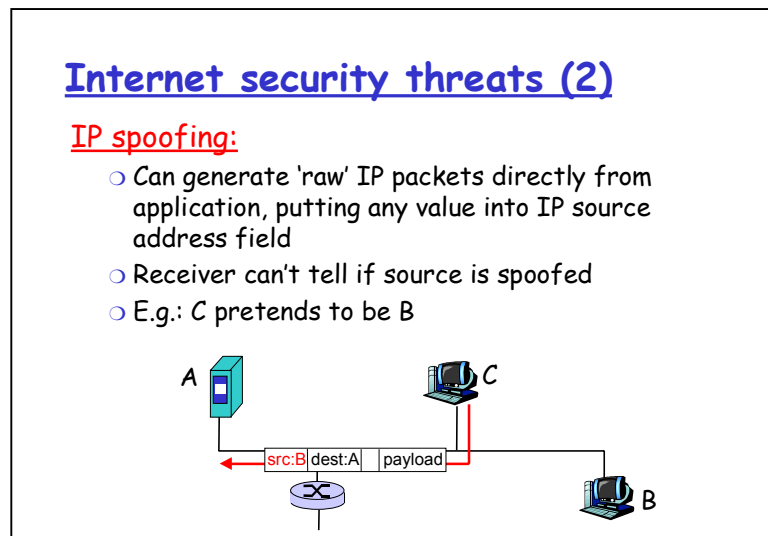
Can an intruder actively inject or remove messages from the network?

The answer to all of these questions is an emphatic yes.

A packet sniffer is a program running in a network-attached device that passively receives all data-link layer frames passing by the device's network interface. In a broadcast environment such as an Ethernet LAN, this means that the packet sniffer receives all frames being transmitted from or to all hosts on the LAN. Any host with an Ethernet card can easily serve as a packet sniffer, as the Ethernet NIC needs only be set to promiscuous mode to receive all passing Ethernet frames. These frames can then be passed on to application programs that extract application-level data.

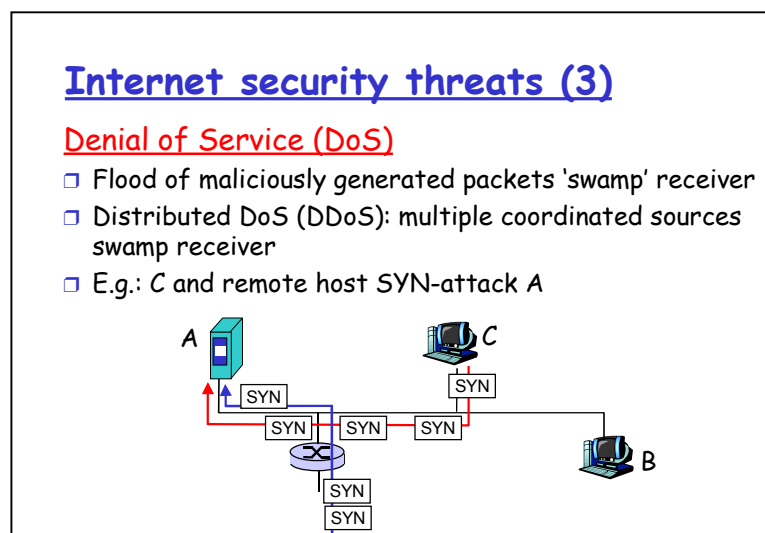
Packet sniffing is a double-edged sword – it can be invaluable to a network administrator for network monitoring and management but also used by the unethical hacker. Packet-sniffing software is freely available at various WWW sites, and as commercial products.

Visual 6



Any Internet-connected device necessarily sends IP datagrams into the network. These datagrams carry the sender's IP address, as well as upper-layer data. A user with complete control over that device's software (in particular its operating system) can easily modify the device's protocols to place an arbitrary IP address into a datagram's Source Address field. This is known as IP spoofing. A user can thus craft an IP packet containing any payload (upper-layer) data it desires and make it appear as if that data was sent from an arbitrary IP host. Packet sniffing and IP spoofing are just two of the more common forms of security 'attacks' on the Internet.

Visual 7



A third broad class of security threats are Denial-of-Service (DoS) attacks. As the name suggests, a DoS attack renders a network, host, or other piece of network infrastructure unusable by legitimate users. Typically, a DoS attack works by creating so much work for the infrastructure under attack that legitimate work cannot be performed. In a so-called SYN flooding attack, the attacker deluges a server with TCP SYN packets, each having a spoofed IP source address. The server, not being able to differentiate between a legitimate SYN and a spoofed SYN, completes the second step of the TCP handshake for a spoofed

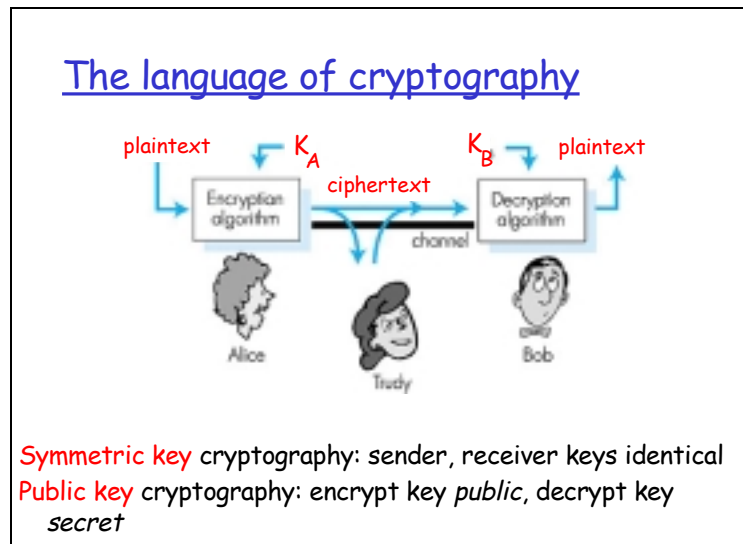
SYN, allocating data structures and state. The third step of the three-way handshake is never completed by the attacker, leaving an ever-increasing number of partially open connections. The load of SYN packets to be processed and depletion of free memory eventually brings the server to its knees.

A ‘smurf’ DoS attack operates by having a large number of innocent hosts respond to ICMP echo-request packets that contain a spoofed source IP address. This results in a large number of ICMP echo-reply packets being sent to the host whose IP address is being spoofed. RFC 2267 and RFC 2644 outline several simple steps that can be taken to help prevent these and other DoS attacks.

Having established that there are indeed real bogeymen (also known as ‘Trudy’) loose in the Internet, what are the Internet equivalents of Alice and Bob, our two friends who need to communicate securely? Certainly, ‘Bob’ and ‘Alice’ might be human users at two end systems, for example, a real Alice and a real Bob who really do want to exchange secure e-mail. They might also be participants in an electronic commerce transaction. For example, a real Alice might want to securely transfer her credit card number to a WWW server to purchase an item online. Similarly, a real Alice might want to interact with her bank online. As noted in RFC 1636, however, the parties needing secure communication might also themselves be part of the network infrastructure. Recall that the Domain Name System (DNS), or routing daemons that exchange routing tables require secure communication between two parties. An intruder that could actively interfere with, control, or corrupt DNS lookups and updates, routing computations, or network management functions could wreak havoc in the Internet.

Principles of Cryptography

Visual 8



Although cryptography has a long history (dating back at least as far as Julius Caesar), modern cryptographic techniques, including many of those used in today’s Internet, are based on advances made in the past thirty years.

For English text, the Caesar cipher would work by taking each letter in the plaintext message and substituting the letter that is k letters later (allowing wraparound; i.e., having the letter ‘z’ followed by the letter ‘a’) in the alphabet. For example if $k = 3$, then the letter ‘a’ in plaintext becomes ‘d’ in ciphertext; ‘b’ in plaintext becomes ‘e’ in ciphertext, and so on. Here, the value of k serves as the key. As an example, the plaintext message “bob, I love you. alice.” becomes “ere, l oryh brx. dolfh.” in ciphertext. While the ciphertext does indeed look like gibberish, it wouldn’t take long to break the code if you knew that the Caesar cipher was being used, as there are only 25 possible key values.

An improvement to the Caesar cipher is the so-called monoalphabetic cipher that also substitutes one letter in the alphabet with another letter in the alphabet. However, rather than substituting according to a regular pattern (for example, substitution with an offset of k for all letters), any letter can be substituted for any other letter, as long as each letter has a unique substitute letter, and vice versa. The substitution rule shows one possible rule for encoding plaintext.

Plaintext

Letter:

Ciphertext

Letter: a b c d e f g h i j k l m n o p q r s t u v w x y z

m n b v c x z a s d f g h j k l p o i u y t r e w q

The plaintext message “bob, I love you. alice.” becomes “nkn, s gktc wky. mgsbc.”

Data Encryption Standard (DES)

Visual 10

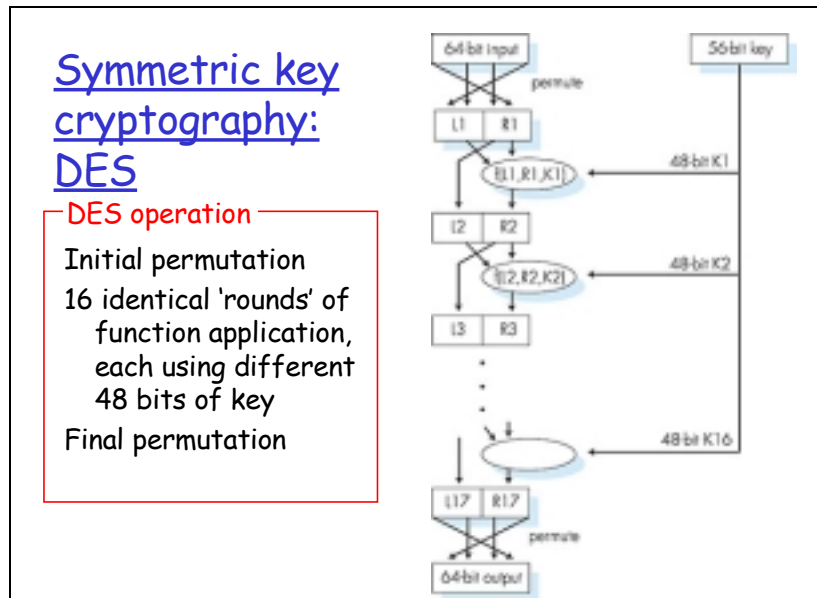
Symmetric key cryptography:
Data Encryption Standard (DES)

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64 bit plaintext input
- How secure is DES?
 - DES Challenge: 56-bit-key-encrypted phrase (“Strong cryptography makes the world a safer place”) decrypted (brute force) in 4 months
 - no known ‘backdoor’ decryption approach
- Making DES more secure
 - use three keys sequentially (3-DES) on each datum
 - use cipher-block chaining

Let us now fast-forward to modern time and examine the Data Encryption Standard, a symmetric-key encryption standard published in 1977 and updated most recently in 1993 by the U.S. National Bureau of Standards for commercial and nonclassified U.S. government use. DES encodes plaintext in 64-bit chunks using a 64-bit key. Actually, 8 of these 64 bits of the key are odd parity bits (there is one parity bit for each of the eight bytes), so the DES key is effectively 56 bits long.

The National Institute of Standards (the successor to the National Bureau of Standards) states the goal of DES as follows: “The goal is to completely scramble the data and key so that every bit of the ciphertext depends on every bit of the data and every bit of the key with a good algorithm, there should be no correlation between the ciphertext and either the original data or key.”

Visual 11



In our discussion we will overview DES operation, leaving the nitty-gritty, bit-level details (there are many) to other sources. The DES consists of two permutation steps (the first and last steps of the algorithm), in which all 64 bits are permuted and there are 16 identical ‘rounds’ of operation in between. The operation of each round is identical, taking the output of the previous round as input. During each round, the rightmost 32 bits of the input are moved to the left 32 bits of the output. Decryption works by reversing the algorithm’s operations.

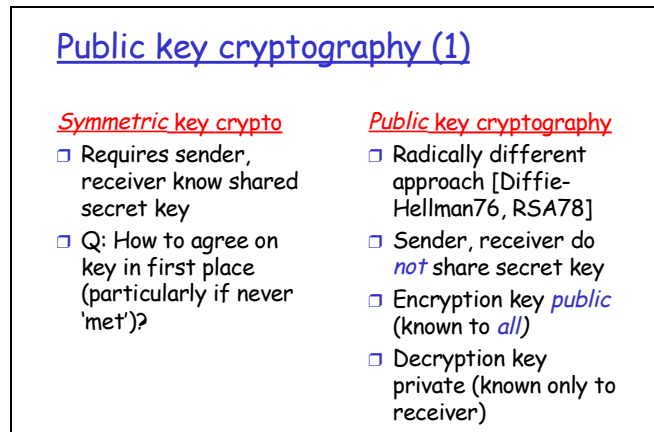
How well does DES work? How secure is it? No one can tell for sure.

A network security company, RSA Data Security Inc., launched a DES Challenge contest to crack (decode) a short phrase it had encrypted using 56 bit DES. The decoded phrase “**Strong cryptography makes the world safer place**” was determined in less than four months by a team which used volunteers throughout the Internet to systematically explore the key space.

For more information : <http://www.itl.nist.gov/fipspubs/fip46-2.htm>

Public Key Encryption

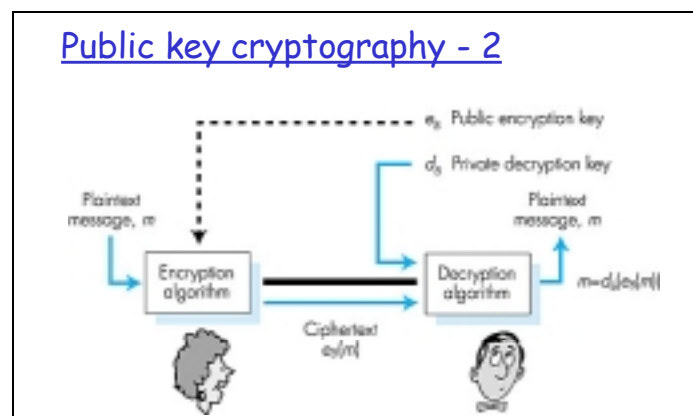
Visual 12



For more than 2000 years (since the time of the Caesar cipher and up to the 1970s), encrypted communication required that the two communicating parties share a common secret – the symmetric key used for encryption and decryption. One difficulty with this approach is that the two parties must somehow agree on the shared key; but to do so requires (presumably secure) communication! In a networked world, however, communicating parties may never meet and may never converse except over the network. Is it possible for two parties to communicate with encryption without having a shared secret key that is known in advance?

In 1976, Diffie and Hellman demonstrated an algorithm (known now as Diffie-Hellman Key Exchange) to do just that – a radically different and marvellously elegant approach toward secure communication that has led to the development of today's public key cryptography systems. Public key cryptography systems also have several wonderful properties that make them useful not only for encryption, but for authentication and digital signatures as well.

Visual 13



The use of public key cryptography is quite simple. Suppose Alice wants to communicate with Bob. Rather than Bob and Alice sharing a single secret key (as in the case of symmetric key systems), Bob (the recipient of Alice's messages) instead has two keys – a public key that is available to everyone in the world (including Trudy the intruder) and a private key that is known only to Bob. In order to communicate with Bob, Alice first

fetches Bob's public key. Alice then encrypts her message to Bob using Bob's public key and a known (for example, standardized) encryption algorithm. Bob receives Alice's encrypted message and uses his private key and a known (for example, standardized) decryption algorithm to decrypt Alice's message. In this manner, Alice can send a secret message to Bob without either of them having to have to distribute any secret key.

Visual 14

Public key encryption algorithms

Two interrelated requirements:

- ① need $d_B(\cdot)$ and $e_B(\cdot)$ such that
$$d_B(e_B(m)) = m$$
- ② need public and private keys
for $d_B(\cdot)$ and $e_B(\cdot)$

RSA: Rivest, Shamir, Adelson algorithm

The use of public key cryptography is thus conceptually simple. But two immediate worries may spring to mind. A first concern is that although an intruder intercepting Alice's encrypted message will only see gibberish, the intruder knows both the key (Bob's public key, which is available for all the world to see) and the algorithm that Alice used for encryption. Trudy can thus mount a chosen plaintext attack, using the known standardized encryption algorithm and Bob's publicly available encryption key to encode any message she chooses!

Trudy might well try, for example, to encode messages, or parts of messages, that she suspects that Alice might send. Clearly, if public key cryptography is to work, key selection and encryption/decryption must be done in such a way that it is impossible (or at least so hard as to be nearly impossible) for an intruder to either determine Bob's private key or somehow otherwise decrypt or guess Alice's message to Bob.

A second concern is that since Bob's encryption key is public, anyone can send an encrypted message to Bob, including Alice or someone claiming to be Alice. In the case of a single shared secret key, the fact that the sender knows the secret key implicitly identifies the sender to the receiver. In the case of public key cryptography, however, this is no longer the case since anyone can send an encrypted message to Bob using Bob's publicly available key. A digital signature, is needed to bind a sender to a message.

While there may be many algorithms and keys that address these concerns, the RSA algorithm (named after its founders, Ron Rivest, Adi Shamir, and Leonard Adleman) has become almost synonymous with public key cryptography. Let us first see how RSA works and then examine why it works. Suppose that Bob wants to receive encrypted messages; there are two interrelated components of RSA:

- Choice of the public key and the private key.
- The encryption and decryption algorithm.

Visual 15

RSA: Choosing keys

1. Choose two large prime numbers p, q .
(e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose e (with $e < n$) that has no common factors with z . (e, z are "relatively prime")
4. Choose d such that $ed-1$ is exactly divisible by z
(in other words: $ed \bmod z = 1$)
5. Public key is (n,e) . Private key is (n,d)

In order to choose the public and private keys, Bob must do the following:

- Choose two large prime numbers, p and q . How large should p and q be? The larger the values, the more difficult it is to break RSA, but the longer it takes to perform the encoding and decoding. RSA Laboratories recommend that the product of p and q be on the order of 768 bits for personal use and 1024 bits for corporate use.
- Compute $n = pq$ and $z = (p - 1)(q - 1)$.
- Choose a number, e , less than n , which has no common factors (other than 1) with z . (In this case, e and z are said to be relatively prime). The letter 'e' is used since this value will be used in encryption.
- Find a number, d , such that $ed - 1$ is exactly divisible (that is, with no remainder) by z . The letter 'd' is used because this value will be used in decryption. Put another way, given e , we choose d such that the integer remainder when ed is divided by z is 1. (The integer remainder when an integer x is divided by the integer n , is denoted $x \bmod n$).
- The public key that Bob makes available to the world is the pair of numbers (n,e) ; his private key is the pair of numbers (n,d) .

Visual 16

RSA: Encryption, decryption

0. Given (n,e) and (n,d) as computed above
1. To encrypt bit pattern, m , compute
 $c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)
2. To decrypt received bit pattern, c , compute
 $m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

Magic happens! $m = (m^e \bmod n)^d \bmod n$

The encryption by Alice, and the decryption by Bob is carried out as follows:

Suppose Alice wants to send Bob a bit pattern, or number, m , such that $m < n$. To encode, Alice performs the exponentiation, m^e , and then computes the integer remainder when m^e is divided by n . Thus, the encrypted value, c , of the plaintext message, m , that Alice sends is:

$$c = m^e \bmod n$$

To decrypt the received ciphertext message, c , Bob computes:

$$m = c^d \bmod n$$

which requires the use of his secret key (n,d) .

As a simple example of RSA, suppose Bob chooses $p = 5$ and $q = 7$. Then $n = 35$ and $z = 24$. Bob chooses $e = 5$, since 5 and 24 have no common factors. Finally, Bob chooses $d = 29$, since $5 * 29 - 1$ (that is, $ed - 1$) is exactly divisible by 24.

Bob makes the two values, $n = 35$ and $e = 5$, public and keeps the value $d = 29$ secret. Observing these two public values, suppose Alice now wants to send the letters 'l' 'o' 'v' and 'e' to Bob. Interpreting each letter as a number between 1 and 26 (with 'a' being 1, and 'z' being 26), Alice and Bob perform the encryption and decryption.

Why Does RSA Work?

The RSA encryption/decryption shown in the following visual appears rather magical. Why should it be that by applying the encryption algorithm and then the decryption algorithm, one recovers the original message? In order to understand why RSA works, we will need to perform arithmetic operations using so-called modulo- n arithmetic. In modular arithmetic, one performs the usual operations of addition, multiplication, and exponentiation. However, the result of each operation is replaced by the integer (whole number) remainder that is left when the result is divided by n . We will take $n = pq$, where p and q are the large prime numbers used in the RSA algorithm.

Visual 17

RSA: Why: $m = (m^e \bmod n)^d \bmod n$

Number theory result: If p, q prime, $n = pq$, then

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$\begin{aligned} (m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\ &= m^{ed \bmod (p-1)(q-1)} \bmod n \\ &\quad \text{(using number theory result above)} \\ &= m^1 \bmod n \\ &\quad \text{(since we chose } ed \text{ to be divisible by } \\ &\quad \text{(} p-1)(q-1 \text{) with remainder 1)} \\ &= m \end{aligned}$$

Recall that under RSA encryption, a message (represented by an integer), m , is first exponentiated to the power e using modulo- n arithmetic to encrypt. Decryption is performed by raising this value to the power d , again using modulo- n arithmetic. The result of an encryption step, followed by a decryption step is thus $(me)d$. Let us now see what we can say about this quantity.

We have:

$$(me)d \bmod n = med \bmod n$$

Although we are trying to remove some of the ‘magic’ about why RSA works, we will need to use a rather magical result from number theory here. Specifically, we will need the result that says:

if p and q are prime, and $n = pq$,

then $xy \bmod n$ is the same as $x(y \bmod (p-1)(q-1)) \bmod n$ [Kaufman 1995].

Applying this result, we have:

$$(me)d \bmod n = m(ed \bmod (p-1)(q-1)) \bmod n$$

But remember that we chose e and d such that $ed - 1$ is exactly divisible (that is, with no remainder) by $(p - 1)(q - 1)$, or equivalently that ed is divisible by $(p - 1)(q - 1)$ with a remainder of 1, and thus $ed \bmod (p - 1)(q - 1) = 1$.

This gives us:

$$(me)d \bmod n = m1 \bmod n = m$$

that is, that

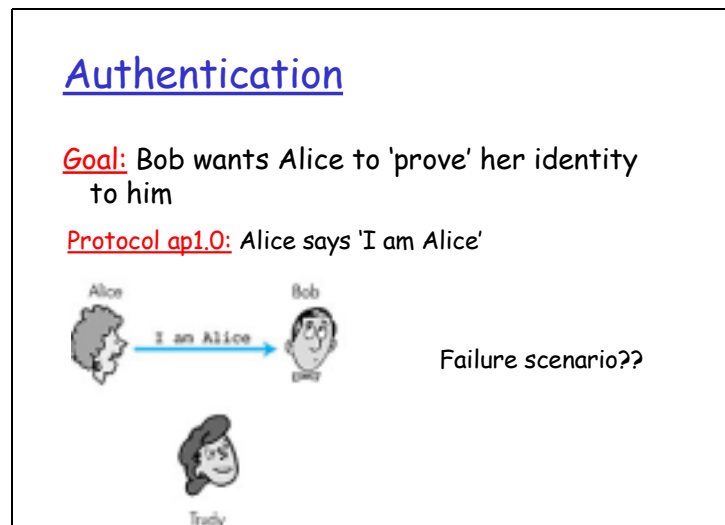
$$(me)d \bmod n = m$$

This is the result we were hoping for! By first exponentiating to the power of e (encrypting) and then exponentiating to the power of d (decrypting), we obtain the original value, m . Even more remarkable is the fact that if we first exponentiate to the power of d and then exponentiate to the power of e – that is, we reverse the order of encryption and decryption, performing the decryption operation first and then applying the encryption operation – we also obtain the original value, m ! (The proof for this result follows the exact same reasoning as above.) We will see shortly that this wonderful property of the RSA algorithm:

$$(me)d \bmod n = m = (md)e \bmod n$$

will be of great use.

The security of RSA relies on the fact that there are no known algorithms for quickly factoring a number – in this case the public value n – into the primes p and q . If one knew p and q , then given the public value e , one could then easily compute the secret key, d . On the other hand, it is not known whether or not there exist fast algorithms for factoring a number, and in this sense the security of RSA is not ‘guaranteed’.

Visual 18

Authentication is the process of proving one's identity to someone else. As humans, we authenticate each other in many ways: we recognize each other's faces when we meet, we recognize each other's voices on the telephone, we are authenticated by the customs official who checks us against the picture on our passport.

In this section we consider how one party can authenticate another party when the two are communicating over a network. We focus here on authenticating a 'live' party, at the point in time when communication is actually occurring. We will see that this is a subtly different problem from proving that a message received at some point in the past (for example, that may have been archived) did indeed come from that claimed sender. This latter problem is referred to as the digital signature problem.

When performing authentication over the network, the communicating parties cannot rely on biometric information, such as a visual appearance or a voiceprint. Here, authentication must be done solely on the basis of messages and data exchanged as part of an authentication protocol.

Typically, an authentication protocol would run before the two communicating parties run some other protocol (for example, a reliable data-transfer protocol, a routing table exchange protocol, or an e-mail protocol).

The authentication protocol first establishes the identities of the parties to each others' satisfaction; only after authentication do the parties get down to the work at hand.

Visual 19

Digital signatures (1)

Cryptographic technique analogous to hand-written signatures.

- Sender (Bob) digitally signs document, establishing he is document owner/creator.
- **Verifiable, nonforgeable:** recipient (Alice) can verify that Bob, and no one else, signed document.

Simple digital signature for message m :

- Bob encrypts m with his public key d_B , creating signed message, $d_B(m)$.
- Bob sends m and $d_B(m)$ to Alice.

Think of the number of times you have signed your name to a piece of paper during the last week. You sign cheques, credit card statements, legal documents, and letters. Your signature attests to the fact that you (as opposed to someone else) have acknowledged and/or agreed with the document's contents. In a digital world, one often wants to indicate the owner or creator of a document, or to signify one's agreement with a document's content. A digital signature is a cryptographic technique for achieving these goals in a digital world.

Just as with human signatures, digital signing should be done in such a way that digital signatures are verifiable, non-forgeable, and cannot be repudiated. That is, it must be possible to 'prove' that a document signed by an individual was indeed signed by that individual (the signature must be verifiable) and that only that individual could have signed the document (the signature cannot be forged, and a signer cannot later repudiate or deny having signed the document). This is easily accomplished with public key cryptography.

Generating Digital Signatures

Visual 20

Digital signatures (2)

- Suppose Alice receives msg m , and digital signature $d_B(m)$
- Alice verifies m signed by Bob by applying Bob's public key e_B to $d_B(m)$ then checks $e_B(d_B(m)) = m$.
- If $e_B(d_B(m)) = m$, whoever signed m must have used Bob's private key

Alice thus verifies that:

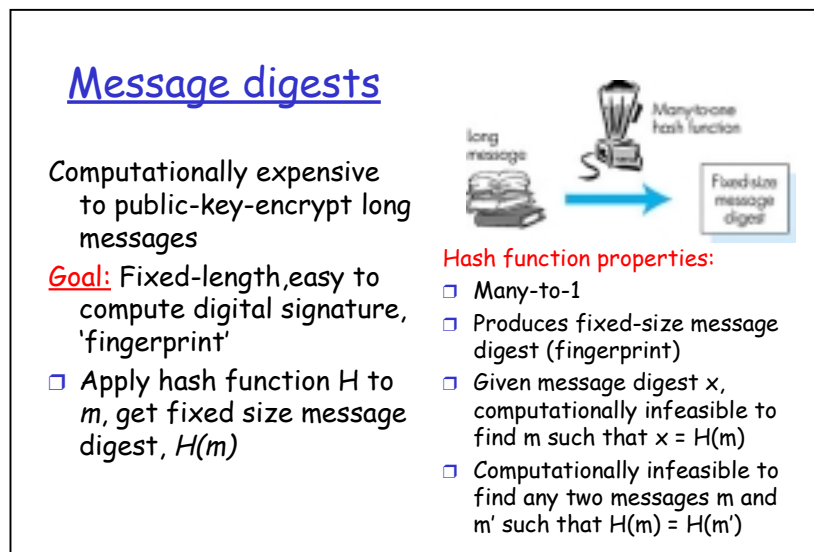
- Bob signed m
- No one else signed m
- Bob signed m and not m'

Non-repudiation:

- Alice can take m , and signature $d_B(m)$ to court and prove that Bob signed m

Suppose that Bob wants to digitally sign a ‘document,’ m . Think of the document as a file or a message that Bob is going to sign and send. To sign this document, Bob simply uses his private decryption key, d_B , to compute $d_B(m)$. At first, it might seem odd that Bob is running a decryption algorithm over a document that hasn’t been encrypted. But recall that ‘decryption’ is nothing more than a mathematical operation (exponentiation to the power of d in RSA and recall that Bob’s goal is not to scramble or obscure the contents of the document, but rather to sign the document in a manner that is verifiable, non-forgable, and non-repudiable. Bob has the document, m , and his digital signature of the document, $d_B(m)$.

Visual 21



We have seen above that public key encryption technology can be used to create a digital signature. One concern with signing data by encryption, however, is that encryption and decryption are computationally expensive. When digitally signing a really important document, say a merger between two large multinational corporations or an agreement with a child to have him/her clean her room weekly, computational cost may not be important. However, many network devices and processes (for example, routers exchanging routing table information and e-mail user agents exchanging e-mail) routinely exchange data that may not need to be encrypted. Nonetheless, they do want to ensure that:

- the sender of the data is as claimed, that is, that the sender has signed the data and this signature can be checked;
- the transmitted data has not been changed since the sender created and signed the data.

Given the overheads of encryption and decryption, signing data via complete encryption/decryption can be overkill. A more efficient approach, using so-called message digests, can accomplish these two goals without full message encryption.

A message digest is in many ways like a checksum. Message-digest algorithms take a message, m , of arbitrary length and compute a fixed length ‘fingerprint’ of the data known as a message digest, $H(m)$. The message digest protects the data in the sense that if m is

changed to m' (either maliciously or by accident) then $H(m)$, computed for the original data (and transmitted with that data), will not match the $H(m)$ computed over the changed data.

While the message digest provides for data integrity, how does it help with signing the message m ? The goal here is that rather than having Bob digitally sign (encrypt) the entire message by computing $dB(m)$, he should be able to sign just the message digest by computing $dB(H(m))$. That is, having m and $dB(H(m))$ together (note that m is not encrypted) should be ‘just as good as’ having a signed complete message, $dB(m)$. This means that m and $dB(H(m))$ together should be non-forgable, verifiable, and non-repudiable. Non-forgability will require that the message-digest algorithm that computes the message digest have some special properties.

Visual 22

Hash function algorithms

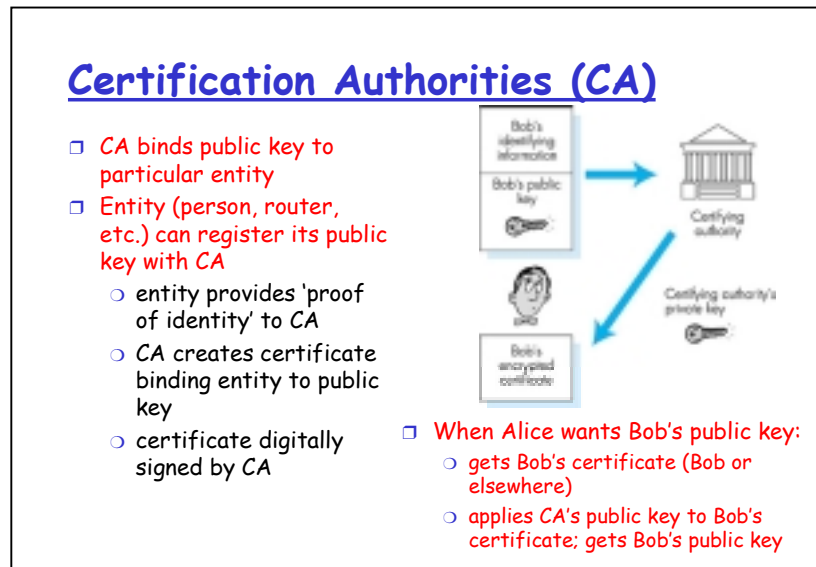
- Internet checksum would make a poor message digest.
 - too easy to find two messages with same checksum.
- MD5 hash function widely used.
 - computes 128-bit message digest in 4-step process.
 - arbitrary 128-bit string x , appears difficult to construct msg m whose MD5 hash is equal to x .
 - SHA-1 is also used
 - US standard
 - 160-bit message digest

The MD5 message digest algorithm by Ron Rivest [RFC 1321] is in wide use today. It computes a 128-bit message digest in a four-step process consisting of a padding step (adding a 1 followed by enough zero's so that the length of the message satisfies certain conditions), an append step (appending a 64-bit representation of the message length before padding), an initialization of an accumulator, and a final looping step in which the message's 16-word blocks are processed (mangled) in four rounds of processing. It is not known whether MD5 actually satisfies the requirements listed above. The author of MD5 claims “It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^{128} operations.” No one has argued with this claim. For a description of MD5 (including a C source code implementation) see RFC 1321.

The second major message-digest algorithm in use today is SHA-1, the Secure Hash Algorithm. This algorithm is based on principles similar to those used in the design of MD4 [RFC 1320], the predecessor to MD5. The Secure Hash Algorithm (SHA-1), a U.S. federal standard, is required for use whenever a secure message digest algorithm is required for federal applications. It produces a 160-bit message digest.

Certification Authorities

Visual 23



In order for public key cryptography to be useful, entities (users, browsers, routers, etc.) need to know for sure that they have the public key of the entity with which they are communicating. For example, when Alice is communicating with Bob using public key cryptography, she needs to know for sure that the public key that is supposed to be Bob's is indeed Bob's. Binding a public key to a particular entity is typically done by a Certification Authority (CA), whose job it is to validate identities and issue certificates.

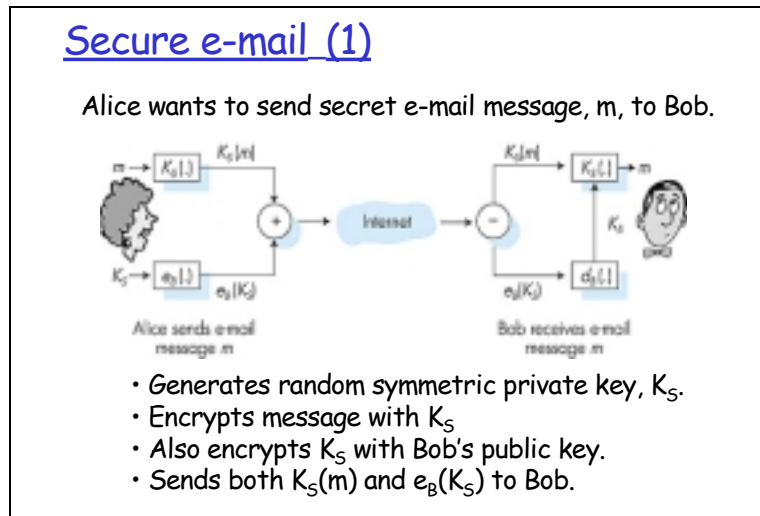
The Roles of a CA

A CA verifies that an entity (a person, a router, and so on) is who it says it is. There are no mandated procedures for how certification is done. When dealing with a CA, one must trust the CA to have performed suitably rigorous identity verification. On the other hand, one might be more willing to trust a CA that is part of a federal or state-sponsored program. One can trust the 'identity' associated with a public key only to the extent that one can trust a CA and its identity-verification techniques.

Once the CA verifies the identity of the entity, the CA creates a certificate that contains the public key and globally unique identifying information about the owner of the public key, for example, a human name or an IP address. The certificate is digitally signed by the CA.

Principles of Secure E-Mail

Visual 24



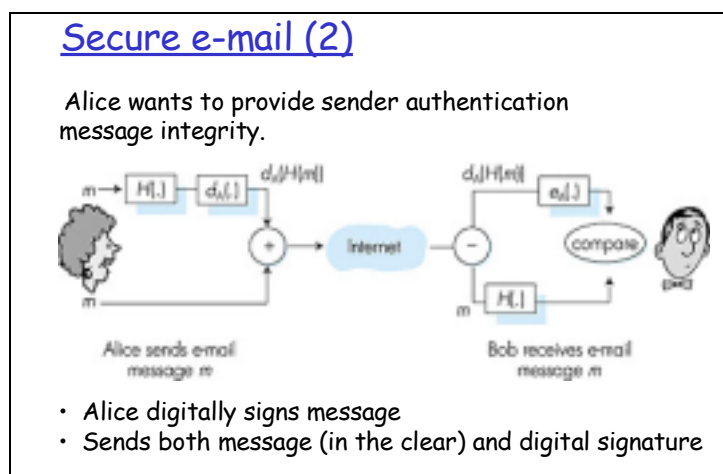
In the context of e-mail, Alice wants to send an e-mail message to Bob, and Trudy wants to intrude. Before ploughing ahead and designing a secure e-mail system for Alice and Bob, we should first consider which security features would be most desirable for them. First and foremost is secrecy. Neither Alice nor Bob wants Trudy to read Alice's e-mail message.

The second feature that Alice and Bob would most likely want to see in the secure e-mail system is sender authentication. In particular, when Bob receives the message from Alice, Bob would naturally want to be sure that the message came from Alice and not from Trudy.

Another feature that the two lovers would appreciate is message integrity, that is, assurance that the message Alice sends is not modified while enroute to Bob.

Finally, the e-mail system should provide receiver authentication, that is, Alice wants to make sure that she is indeed sending the letter to Bob and not to someone else (for example, Trudy) who is impersonating Bob.

Visual 25



So let us begin by addressing the foremost concern of Alice and Bob, namely, secrecy. The most straightforward way to provide secrecy is for Alice to encrypt the message with symmetric key technology (such as DES) and for Bob to decrypt the message upon message receipt. If the symmetric key is long enough, and if only Alice and Bob have the key, then it is extremely difficult for anyone else (including Trudy) to read the message.

It is hard to distribute a symmetric key so that only Alice and Bob have copies of it. So we naturally consider an alternative tool, namely, public key cryptography (using, for example, RSA). In the public key approach, Bob makes his public key publicly available (for example, in a public key server or on his personal Web page), Alice encrypts her message with Bob's public key, and sends the encrypted message to Bob's e-mail address. The encrypted message is encapsulated with MIME headers and sent over ordinary SMTP. When Bob receives the message, he simply decrypts it with his private key. Assuming that Alice knows for sure that the public key is Bob's public key (and that the key is long enough), then this approach is an excellent means to provide the desired secrecy. One problem, however, is that public key encryption is relatively inefficient, particularly for long messages. (Long e-mail messages are now commonplace in the Internet, due to the increasing use of attachments, images, audio, and video.)

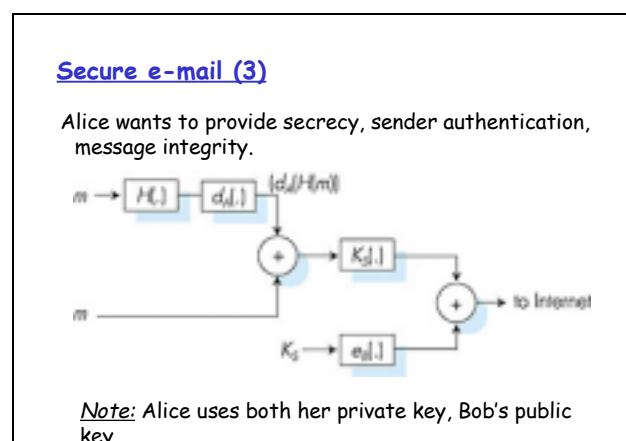
To overcome the efficiency problem, let us make use of a session key. In particular Alice:

- (1) selects a symmetric key, K_S , at random;
- (2) encrypts her message, m , with the symmetric key, K_S ;
- (3) encrypts the symmetric key with Bob's public key, e_B ;
- (4) concatenates the encrypted message and the encrypted symmetric key to form a 'package';
- (5) sends the package to Bob's e-mail address.

When Bob receives the package, he:

- (1) uses his private key d_B to obtain the symmetric key, K_S ; and
- (2) uses the symmetric key K_S to decrypt the message m .

Visual 26



Now let us consider designing an e-mail system that provides secrecy, sender authentication, and message integrity. This can be done by combining the procedures:

- Alice first creates a preliminary package, which consists of her original message along with a digitally signed hash of the message.
- She then treats this preliminary package as a message in itself, and sends this new message through the sender creating a new package that is sent to Bob.

It should be clear that this design achieves the goal of providing secrecy, sender authentication, and message integrity. Note in this scheme that Alice applies public key encryption twice: once with her own private key and once with Bob's public key. Similarly, Bob applies public key encryption twice – once with his private key and once with Alice's public key.

The secure e-mail design outlined probably provides satisfactory security for most e-mail users for most occasions. But there is still one important issue that remains to be addressed. The design requires Alice to obtain Bob's public key, and requires Bob to obtain Alice's public key. The distribution of these public keys is a nontrivial problem. For example, Trudy might masquerade as Bob and give Alice her own public key while saying that it is Bob's public key. A popular approach for securely distributing public keys is to certify the public keys.

Visual 27

Pretty Good Privacy (PGP)

- Internet e-mail encryption scheme, a de facto standard.
- Uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- Provides secrecy, sender authentication, integrity.
- Inventor, Phil Zimmerman, was target of 3-year federal investigation.

A PGP signed message:

```

---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob:My husband is out of town
tonight.Passionately yours,
Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRhhGJGhgq/l2EpJ+lo8gE4vB3mqJ
hFEvZP9t6n7G6m5Gw2
---END PGP SIGNATURE---
```

Originally written by Phil Zimmermann in 1991, pretty good privacy (PGP) is an e-mail encryption scheme that has become a de-facto standard. Versions of PGP are available in the public domain; for example, you can find the PGP software for your favourite platform as well as lots of interesting reading at the International PGP Home Page .

PGP is also commercially available [Network Associates 1999], and is also available as a plug-in for many e-mail user agents, including Microsoft's Exchange and Outlook, and Qualcomm's Eudora.

Depending on the version, the PGP software uses MD5 or SHA for calculating the message digest; CAST, Triple-DES, or IDEA for symmetric key encryption; and RSA for the public key encryption. In addition, PGP provides data compression.

When PGP is installed, the software creates a public key pair for the user. The public key can be posted on the user's Web site or placed in a public key server. The private key is protected by the use of a password. The password has to be entered every time the user accesses the private key. PGP gives the user the option of digitally signing the message, encrypting the message, or both digitally signing and encrypting.

Case History

Phil Zimmermann and PGP

Philip R. Zimmermann is the creator of Pretty Good Privacy (PGP). For that, he was the target of a three-year criminal investigation, because the government held that U.S. export restrictions for cryptographic software were violated when PGP spread all around the world following its 1991 publication as freeware. After releasing PGP as shareware, someone else put it on the Internet and foreign citizens downloaded it. Cryptography programs in the United States are classified as munitions under federal law and may not be exported.

Despite the lack of funding, the lack of any paid staff, the lack of a company to stand behind it, and despite government interventions, PGP nonetheless became the most widely used e-mail encryption software in the world. Oddly enough, the U.S. government may have inadvertently contributed to PGP's spread, by making it more popular because of the Zimmermann case.

The U.S. government dropped the case in early 1996. The announcement was met with celebration by Internet activists. The Zimmermann case had become the story of an innocent person fighting for his rights against the abuses of big government. The government's giving in was welcome news, in part because of the campaign for Internet censorship in Congress, and the push by the FBI to allow increased government snooping.

After the government dropped its case, Zimmermann founded PGP Inc., which was acquired by Network Associates in December 1997. Zimmermann is now a Senior Fellow at Network Associates, as well as an independent consultant in matters cryptographic.

Secure Sockets Layer

Visual 28

Secure Sockets Layer (SSL)

- PGP provides security for a specific network app.
- SSL works at transport layer. Provides security to any TCP-based app using SSL services
- SSL: used between WWW browsers, servers for I-commerce (shttp)
- SSL security services:
 - server authentication
 - data encryption
 - client authentication (optional)
- Server authentication:
 - SSL-enabled browser includes public keys for trusted CAs
 - browser requests server certificate, issued by trusted CA
 - browser uses CA's public key to extract server's public key from certificate
- Visit your browser's security menu to see its trusted CAs

Secure Sockets Layer (SSL), originally developed by Netscape, is a protocol designed to provide data encryption and authentication between a Web client and a Web server. The protocol begins with a handshake phase that negotiates an encryption algorithm (for example, DES or IDEA) and keys, and authenticates the server to the client. Optionally, the client can also be authenticated to the server. Once the handshake is complete and the transmission of application data begins, all data is encrypted using session keys negotiated during the handshake phase. SSL is widely used in Internet commerce, being implemented in almost all popular browsers and Web servers.

SSL and TLS are not limited to the Web application; for example, they can similarly be used for authentication and data encryption for IMAP mail access. SSL can be viewed as a layer that sits between the application layer and the transport layer. On the sending side, SSL receives data (such as an HTTP or IMAP message from an application), encrypts the data, and directs the encrypted data to a TCP socket. On the receiving side, SSL reads from the TCP socket, decrypts the data, and directs the data to the application. Although SSL can be used with many Internet applications, we will discuss it in the context of the Web, where it is principally being used today for Internet commerce.

Visual 29

SSL (continued)

Encrypted SSL session:

- Browser generates symmetric session key, encrypts it with server's public key, sends encrypted key to server.
- Using its private key, server decrypts session key.
- Browser, server agree that future msgs will be encrypted.
- All data sent into TCP socket (by client or server) is encrypted with session key.
- SSL: basis of IETF Transport Layer Security (TLS).
- SSL can be used for non-Web applications, e.g., IMAP.
- Client authentication can be done with client certificates.

Features Provided by SSL

- SSL server authentication, allowing a user to confirm a server's identity. An SSL-enabled browser maintains a list of trusted Certification Authorities (CAs) along with the public keys of the CAs. When the browser wants to do business with an SSL-enabled Web server, the browser obtains a certificate from the server containing the server's public key. The certificate is issued (that is, digitally signed) by a CA listed in the client's list of trusted CAs. This feature allows the browser to authenticate the server before the user submits a payment card number. In the context of the earlier example, this server authentication enables Bob to verify that he is indeed sending his payment card number to Alice Incorporated, and not to someone else who might be masquerading as Alice Incorporated.
- SSL client authentication, allowing a server to confirm a user's identity. Analogous to server authentication, client authentication makes use of client certificates, which have also been issued by CAs. This authentication is important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity. Client authentication, although supported by SSL, is optional. To keep our discussion focused, we will henceforth ignore it.
- An encrypted SSL session, in which all information sent between browser and server is encrypted by the sending software (browser or Web server) and decrypted by the receiving software (browser or Web server). This confidentiality may be important to both the customer and the merchant. Also, SSL provides a mechanism for detecting tampering of the information by an intruder.

How SSL Works

A user, say Bob, surfs the Web and clicks on a link that takes him to a secure page housed by Alice's SSL-enabled server. The protocol part of the URL for this page is 'https' rather than the ordinary 'http'. The browser and server then run the SSL handshake protocol, which (1) authenticates the server and (2) generates a shared symmetric key. Both of these tasks make use of RSA public key technology. The main flow of events in the handshake phase.

During this phase, Alice sends Bob her certificate, from which Bob obtains Alice's public key. Bob then creates a random symmetric key, encrypts it with Alice's public key, and sends the encrypted key to Alice. Bob and Alice now share a symmetric session key. Once this handshake protocol is complete, all data sent between the browser and server (over TCP connections) is encrypted using the symmetric session key.

The Limitations of SSL in Internet Commerce

Due to its simplicity and early development, SSL is widely implemented in browsers, servers, and Internet commerce products. These SSL-enabled servers and browsers provide a popular platform for payment card transactions. Nevertheless, we should keep in mind that SSL was not specifically tailored for payment card transactions, but instead for generic secure communication between a client and server. Because of this generic design, SSL lacks many features that the payment-card industry would like to see in an Internet commerce protocol.

Consider once again what happens when Bob makes a purchase from Alice Incorporated over SSL. The signed certificate that Bob receives from Alice assures Bob that he is really dealing with Alice Incorporated, and that Alice Incorporated is a bona fide company. However, the generic certificate does not indicate whether Alice Incorporated is authorized to accept payment-card purchases nor if the company is a reliable merchant. This opens the door for merchant fraud. Also, there is a similar problem for client authorization. Even if SSL client authentication is used, the client certificate does not tie Bob to a specific authorized payment card; thus, Alice Incorporated has no assurance about whether Bob is authorized to make a payment-card purchase. This opens the door to all kinds of fraud, including purchases with stolen credit cards and customer repudiation of purchased goods.

Of course, this kind of fraud is already rampant in mail order and telephone order (MOTO) purchases. With MOTO transactions, the law dictates that the merchant accepts liability for fraudulent transactions. Thus, if a customer makes a MOTO purchase with a payment card and claims to have never made the purchase, then the merchant is liable, that is, the merchant is legally bound to return the money to the customer (unless the merchant can prove that the customer actually ordered and received the goods). Similarly, if a MOTO purchase is made with a stolen payment card, the merchant is again liable. On the other hand, with physically present transactions, the merchant's bank accepts the liability. As you might expect, it is more difficult for a customer to repudiate a physically present purchase that involves a hand-written signature or a PIN.

SSL purchases are similar to MOTO purchases, and naturally the merchant is liable for a fraudulent SSL purchase. It would be preferable, of course, to use a protocol that provides superior authentication of the customer and of the merchant, something that is as good or better than a physically present transaction. Authentication involving payment-card authorization would reduce fraud and merchant liability.

Secure Electronic Transactions (SET)

Visual 30

Secure Electronic Transactions (SET)

- ❑ Designed for payment-card transactions over Internet.
- ❑ Provides security services among 3 players:
 - customer
 - merchant
 - merchant's bankAll must have certificates.
- ❑ SET specifies legal meanings of certificates.
 - apportionment of liabilities for transactions
- ❑ Customer's card number passed to merchant's bank without merchant ever seeing number in plain text.
 - Prevents merchants from stealing, leaking payment card numbers.
- ❑ Three software components:
 - Browser wallet
 - Merchant server
 - Acquirer gateway
- ❑ See text for description of SET transaction.

Secure Electronic Transactions (SET) is a protocol specifically designed to secure payment-card transactions over the Internet. It was originally developed by Visa

International and MasterCard International in February 1996 with participation from leading technology companies around the world.

Some of the principle characteristics of SET are as follows:

- SET is designed to encrypt specific kinds of payment-related messages; it cannot be used to encrypt arbitrary data (such as text and images) as can SSL.
- The SET protocol involves all three players mentioned at the beginning of this section, namely, the customer, the merchant, and the merchant's bank. All sensitive information sent between the three parties is encrypted.
- SET requires all three players to have certificates. The customer's and merchant's certificates are issued by their banks, thereby assuring that these players are permitted to make and receive payment-card purchases. The customer certificate provides merchants with assurance that transactions will not be fraudulently charged back. It is an electronic representation of the customer's payment card. It contains information about the account, the issuing financial institution, and other cryptographic information. The merchant certificate assures the consumer that that merchant is authorized to accept payment-card purchases. It contains information about the merchant, the merchant's bank, and the financial institution issuing the certificate.
- SET specifies the legal meaning of the certificates held by each party and the apportionment of liabilities connected with a transaction. In a SET transaction, the customer's payment-card number is passed to the merchant's bank without the merchant ever seeing the number in plain text. This feature prevents fraudulent or careless merchants from stealing or accidentally leaking the payment-card number.

A SET transaction uses three software components:

- **Browser wallet** – this application is integrated with the browser and provides the customer with storage and management of payment cards and certificates while shopping. It responds to SET messages from the merchant, prompting the customer to select a payment card for payment.
- **Merchant server** – is the merchandising and fulfilment engine for merchants selling on the Web. For payments, it processes cardholder transactions and communicates with the merchant's bank for approval and subsequent payment capture.
- **Acquirer gateway** – is the software component at the merchant's bank. It processes the merchant's payment card transaction for authorization and payment.

Visual 31

Ipsec: Network layer security

- **Network-layer secrecy:**
 - sending host encrypts the data in IP datagram
 - TCP and UDP segments; ICMP and SNMP messages.
- **Network-layer authentication**
 - destination host can authenticate source IP address
- **Two principle protocols:**
 - Authentication Header (AH) protocol
 - Encapsulation Security Payload (ESP) protocol
- **For both AH and ESP, source, destination handshake:**
 - create network-layer logical channel called a service agreement (SA)
- **Each SA unidirectional.**
- **Uniquely determined by:**
 - security protocol (AH or ESP)
 - source IP address
 - 32-bit connection ID

The IP security protocol, more commonly known as IPsec, is a suite of protocols that provides security at the network layer. IPsec is a rather complex animal, and different parts of it are described in more than a dozen RFCs. In this section, we will discuss IPsec in a specific context, namely, in the context that all hosts in the Internet support IPsec. Although this context is many years away, the context will simplify the discussion and help us understand the key features of IPsec. Two key RFCs are RFC 2401, which describes the overall IP security architecture and RFC 2411, which provides an overview of the IPsec protocol suite and the documents describing it.

Summary**Visual 32**

Summary

Basic techniques:

- Cryptography (symmetric and public)
- Authentication
- Message integrity

.... used in many different security scenarios

- Secure e-mail
- Secure transport (SSL)
- IP sec

See also: firewalls, in network management

We saw that security is needed at various layers in a network architecture to protect against ‘bad guys’ who may sniff packets, remove packets from the network, or inject their packets into the network.

The first part of this chapter presented various principles underlying secure communication. We covered cryptographic techniques for coding and decoding data, including both symmetric key cryptography and public key cryptography. DES and RSA

were examined as specific case studies of these two major classes of cryptographic techniques in use in today's networks.

In the second part of this chapter we thus turned our attention to the use of various security techniques in networks. We also examined the use of PGP as a public-key e-mail encryption scheme. Our case studies continued as we headed down the protocol stack and examined the secure sockets layer (SSL) and secure electronic transactions, the two primary protocols in use today for secure electronic commerce. Both are based on public key techniques. Finally, we examined a suite of security protocols for the IP layer of the Internet--the so-called IPsec protocols. These can be used to provide secrecy, authentication, and message integrity between two communicating IP devices.

