# Security Engineering

Lesson 11

Public-Key Digital Signature Algorithms, Key-Exchange Algorithms

Spring 2010

Dr. Marenglen Biba

# Outline

- Public-Key Digital Signature Algorithms
  - Digital Signature Algorithm (DSA)
  - DSA Variants
  - Gost Digital Signature Algorithm
  - Discrete Logarithm Signature Schemes

# DSA

- In August 1991, The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS).
- DSA is the algorithm; the DSS is the standard. The standard employs the algorithm. The algorithm is part of the standard.
- DSA is covered by U.S. Patent 5,231,668, filed July 26, 1991, and attributed to David W. Kravitz, a former NSA employee.
  - This patent was given to "The United States of America as represented by the Secretary of Commerce, Washington, D.C." and the NIST has made this patent available worldwide royalty-free.

# Features of DSA

1. DSA cannot be used for encryption or key distribution.
   - DSS is a signature standard.
2. DSA was developed by the NSA, and there may be a trapdoor in the algorithm.
3. DSA is slower than RSA
4. RSA is a *de facto* standard.
5. The DSA selection process was not public; sufficient time for analysis has not been provided.
6. DSA may infringe on other patents.

# Description of DSA

The algorithm uses the following parameters:

$p$ = a prime number $L$ bits long, when $L$ ranges from 512 to 1024 and is a multiple of 64. (In the original standard, the size of $p$ was fixed at 512 bits).

$q$ = a 160-bit prime factor of $p - 1$.

$g = h^{(p-1)/q} \bmod p$, where $h$ is any number less than $p - 1$ such that $h^{(p-1)/q} \bmod p$ is greater than 1.

x = a number less than $q$.

$y = g^x \bmod p$.

- The algorithm also makes use of a one-way hash function: H(m). The standard specifies the Secure Hash Algorithm.

# Signing a message with DSA

- The first three parameters, p, q, and g, are public and can be common across a network of users.
- The private key is x; the public key is y.
- To sign a message, m:
  - **(1)** Alice generates a random number, k, less than q.
  - **(2)** Alice generates:

$$r = (g^k \bmod p) \bmod q$$

$$s = (k^{-1} (H(m) + xr)) \bmod q$$

- The parameters r and s are her signature; she sends these to Bob.

# Verifying a message with DSA

**3)** Bob verifies the signature by computing

- $w = s^{-1} \bmod q$
- $u_1 = (H(m) * w) \bmod q$
- $u_2 = (rw) \bmod q$
- $v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$

If v = r, then the signature is verified.

# DSA Summary

**Table 20.1**
**DSA Signatures**

***Public Key:***

$p$ 512-bit to 1024-bit prime (can be shared among a group of users)

$q$ 160-bit prime factor of $p - 1$ (can be shared among a group of users)

$g = h^{(p-1)/q} \bmod p$, where $h$ is less than $p - 1$ and $h^{(p-1)/q} \bmod p > 1$ (can be shared among a group of users)

$y = g^x \bmod p$ (a p-bit number)

***Private Key:***

$x < q$ (a 160-bit number)

***Signing:***

$k$ choose at random, less than $q$

$r$ (signature) $= (g^k \bmod p) \bmod q$

$s$ (signature) $= (k^{-1} (H(m) + xr)) \bmod q$

***Verifying:***

$w = s^{-1} \bmod q$

$u_1 = (H(m) * w) \bmod q$

$u_2 = (rw) \bmod q$

$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$

If $v = r$, then the signature is verified.

# DSA Speed

**Table 20.2**
**DSA Speeds for Different Modulus Lengths with a 160-bit Exponent (on a SPARC II)**

| | 512 bits | 768 bits | 1024 bits |
|---|---|---|---|
| Sign | 0.20 sec | 0.43 sec | 0.57 sec |
| Verify | 0.35 sec | 0.80 sec | 1.27 sec |

# Security of DSA

- At 512-bits, DSA wasn’t strong enough for long-term security. At 1024 bits, it is.

# Outline

- Public-Key Digital Signature Algorithms
  - Digital Signature Algorithm (DSA)
  - DSA Variants
  - Gost Digital Signature Algorithm
  - Discrete Logarithm Signature Schemes

# DSA Variants

- This variant makes computation easier on the signer by not forcing him to compute $k^{-1}$. All the parameters are as in DSA. To sign a message, m, Alice generates two random numbers, k and d, both less than q. The signature is

  $r = (g^k \bmod p) \bmod q$

  $s = (H(m) + xr) * d \bmod q$

  $t = kd \bmod q$

- Bob verifies the signature by computing

  $w = t/s \bmod q$

  $u_1 = (H(m) * w) \bmod q$

  $u_2 = (rw) \bmod q$

- If $r = ((g^{u1} * y^{u2}) \bmod p) \bmod q$, then the signature is verified.

# Outline

- Public-Key Digital Signature Algorithms
  - Digital Signature Algorithm (DSA)
  - DSA Variants
  - Gost Digital Signature Algorithm
  - Discrete Logarithm Signature Schemes

# GOST

- This is a Russian digital signature standard, officially called GOST R 34.10-94. The algorithm is very similar to DSA, and uses the following parameters
  - p = a prime number, either between 509 and 512 bits long, or between 1020 and 1024 bits long.
  - q = a 254- to 256-bit prime factor of p – 1.

    a = any number less than p – 1 such that $a^q \bmod p = 1$.

    x = a number less than q.

    $y = a^x \bmod p$.

# GOST

- The first three parameters, p, q, and a, are public and can be common across a network of users. The private key is x; the public key is y.
- To sign a message, m
  - **(1)** Alice generates a random number, k, less than q
  - **(2)** Alice generates

    $r = (a^k \bmod p) \bmod q$

    $s = (xr + k(H(m))) \bmod q$

    If $H(m) \bmod q = 0$, then set it equal to 1. If r = 0, then choose another k and start again. The signature is two numbers: r mod $2^{256}$ and s mod $2^{256}$. She sends these to Bob.
  - **(3)** Bob verifies the signature by computing
    - $v = H(m)^{q-2} \bmod q$
    - $z1 = (sv) \bmod q$
    - $z2 = ((q - r) * v) \bmod q$
    - $u = ((a^{z1} * y^{z2}) \bmod p) \bmod q$

  If u = r, then the signature is verified.

# Outline

- Public-Key Digital Signature Algorithms
  - Digital Signature Algorithm (DSA)
  - DSA Variants
  - Gost Digital Signature Algorithm
  - Discrete Logarithm Signature Schemes

# Discrete Logarithm Signature Schemes

- ElGamal and DSA signature schemes are very similar.
- In fact, they are just three examples of a general digital signature scheme based on the Discrete Logarithm Problem.
- Choose p, a large prime number, and q, either p – 1 or a large prime factor of p – 1.
- Then choose g, a number between 1 and p such that $g^q \equiv 1 \pmod{p}$.
- All these numbers are public, and can be common to a group of users.
- The private key is x, less than q. The public key is $y = g^x \bmod p$.

# Discrete Logarithm Signature Schemes

- To sign a message, m, first choose a random k less than and relatively prime to q. If q is also prime, any k less than q works. First compute:

$$r = g^k \bmod p$$

The generalized **signature equation** now becomes

$$ak = b + cx \bmod q$$

- The coefficients a, b, and c can be any of a variety of things. (Table 20.4)
- To verify the signature, the receiver must confirm that

$$r^a = g^b y^c \bmod p$$

This is called the **verification equation**.

# Possible Permutations of $a$, $b$, and $c$

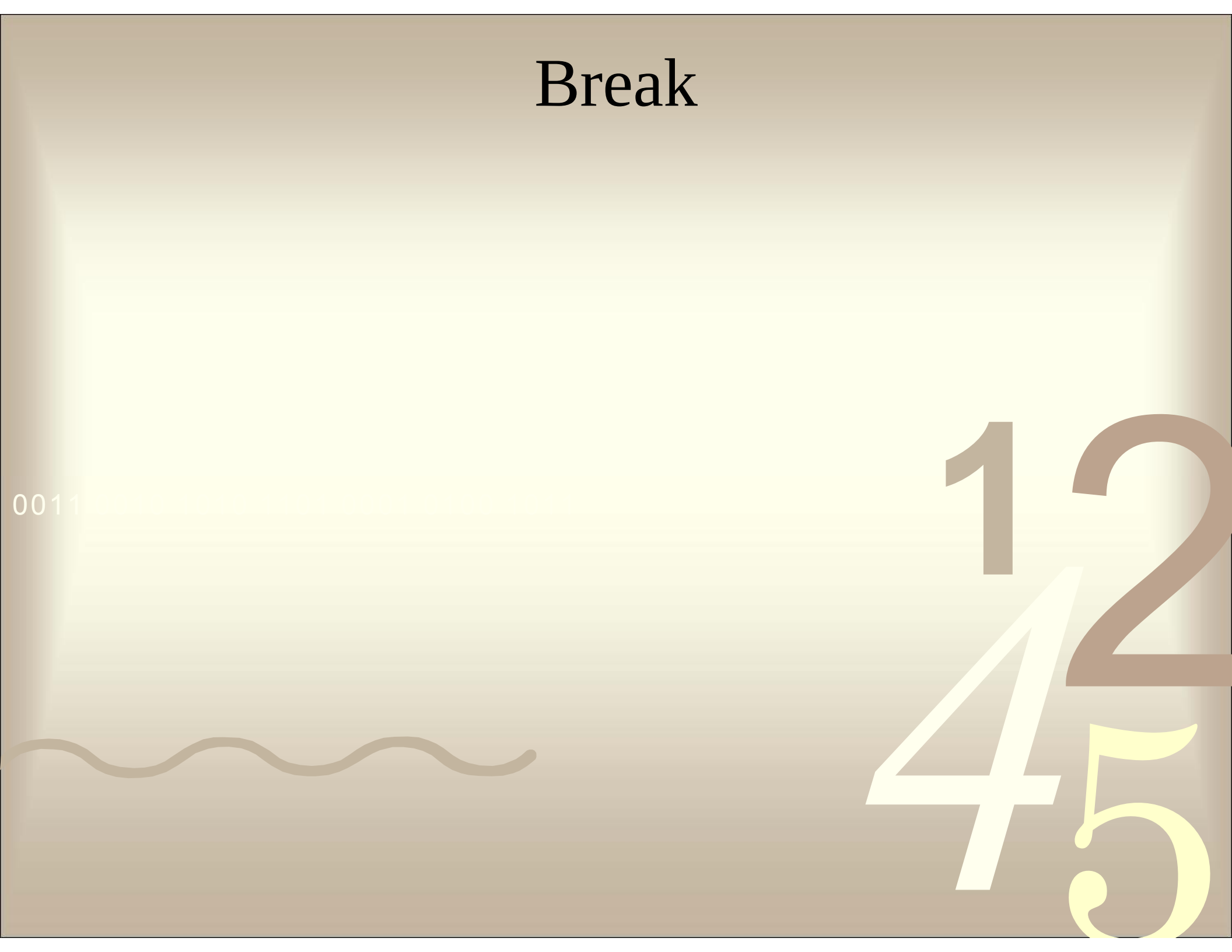
**Table 20.4**
**Possible Permutations of $a$, $b$, and $c$ ($r' = r \bmod q$)**

| | | |
|---|---|---|
| $\pm r'$ | $\pm s$ | $m$ |
| $\pm r'm$ | $\pm s$ | 1 |
| $\pm r'm$ | $\pm ms$ | 1 |
| $\pm mr'$ | $\pm r's$ | 1 |
| $\pm ms$ | $\pm r's$ | 1 |

# Discrete Logarithm Signature Schemes

- This is a remarkable piece of research.
- All of the various discrete-logarithm-based digital signature schemes have been put in one coherent framework.
- This puts to rest any patent dispute between Schnorr and DSA: DSA is not a derivative of Schnorr, nor even of ElGamal.
- All three are examples of this general construction, and this general construction is unpatented.

# Break



# PART II: Outline

– Key-Exchange Algorithms

Diffie-Hellman

Station-to-Station Protocol

Encrypted Key Exchange

Fortified Key Negotiation

# Diffie-Hellman

- Diffie-Hellman was the first public-key algorithm ever invented, way back in 1976.
- It gets its security from the difficulty of calculating discrete logarithms in a finite field.
- Diffie-Hellman can be used for key distribution — Alice and Bob can use this algorithm to generate a secret key — but it cannot be used to encrypt and decrypt messages.

# Diffie-Hellman

- The math is simple.
- First, Alice and Bob agree on a large prime, *n* and *g*, such that *g* is primitive mod *n*.
- These two integers don’t have to be secret.
- Alice and Bob can agree to them over some insecure channel.
- They can even be common among a group of users. It doesn’t matter.

# Diffie-Hellman

Then, the protocol goes as follows:

**(1)** Alice chooses a random large integer $x$ and sends Bob

$X = g^x \bmod n$

**(2)** Bob chooses a random large integer $y$ and sends Alice

$Y = g^y \bmod n$

**(3)** Alice computes

$k = Y^x \bmod n$

**(4)** Bob computes

$k' = X^y \bmod n$

- Both $k$ and $k'$ are equal to $g^{xy} \bmod n$.
- No one, listening on the channel, can compute that value; they only know $n$, $g$, $X$, and $Y$.
- Unless they can compute the discrete logarithm and recover $x$ or $y$, they do not solve the problem.
- So, $k$ is the secret key that both Alice and Bob computed independently.

# Diffie-Hellman with Three or More Parties

- The Diffie-Hellman key-exchange protocol can easily be extended to work with three or more people. In this example, Alice, Bob, and Carol together generate a secret key.

  **(1)** Alice chooses a random large integer $x$ and sends Bob

  $X = g^x \bmod n$

  **(2)** Bob chooses a random large integer $y$ and sends Carol

  $Y = g^y \bmod n$

  **(3)** Carol chooses a random large integer $z$ and sends Alice

  $Z = g^z \bmod n$

  **(4)** Alice sends Bob

  $Z' = Z^x \bmod n$

  **(5)** Bob sends Carol

  $X' = X^y \bmod n$

# Diffie-Hellman with Three or More Parties

**(6)** Carol sends Alice

$Y' = Y^z \bmod n$

**(7)** Alice computes

$k = Y'^x \bmod n$

**(8)** Bob computes

$k = Z'^y \bmod n$

**(9)** Carol computes

$k = X'^z \bmod n$

- The secret key, $k$, is equal to $g^{xyz} \bmod n$, and no one else listening in on the communications can compute that value.
- The protocol can be easily extended to four or more people; just add more people and more rounds of computation.

# Hughes

- This variant of Diffie-Hellman allows Alice to generate a key and send it to Bob.
  - **(1)** Alice chooses a random large integer x and generates

    $k = g^x \bmod n$
  - **(2)** Bob chooses a random large integer y and sends Alice

    $Y = g^y \bmod n$
  - **(3)** Alice sends Bob

    $X = Y^x \bmod n$
  - **(4)** Bob computes

    $z = y\text{-}1$

    $k' = X^z \bmod n$

    If everything goes correctly, $k = k'$.
- The advantage of this protocol over Diffie-Hellman is that k can be computed before any interaction, and Alice can encrypt a message using k prior to contacting Bob.

# Outline

- Key-Exchange Algorithms
  - Diffie-Hellman
  - Station-to-Station Protocol
  - Encrypted Key Exchange
  - Fortified Key Negotiation
  - Conference Key Distribution and Secret Broadcasting

# Station-to-Station Protocol

- Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack.
- One way to prevent this problem is to have Alice and Bob sign their messages to each other.
- This protocol assumes that Alice has a certificate with Bob's public key and that Bob has a certificate with Alice's public key.
- These certificates have been signed by some trusted authority outside this protocol.
- Here's how Alice and Bob generate a secret key, *k*.

# Station-to-Station Protocol

**(1)** Alice generates a random number, $x$, and sends it to Bob.

**(2)** Bob generates a random number, $y$. Using the Diffie-Hellman protocol he computes their shared key based on $x$ and $y$: $k$. He signs $x$ and $y$, and encrypts the signature using $k$. He then sends that, along with $y$, to Alice.

$$y, E_k(S_B(x,y))$$

**(3)** Alice also computes $k$. She decrypts the rest of Bob's message and verifies his signature. Then she sends Bob a signed message consisting of $x$ and $y$, encrypted in their shared key.

$$E_k(S_A(x,y))$$

**(4)** Bob decrypts the message and verifies Alice's signature.

# Outline

- Key-Exchange Algorithms
  - Diffie-Hellman
  - Station-to-Station Protocol
  - Encrypted Key Exchange
  - Fortified Key Negotiation
  - Conference Key Distribution and Secret Broadcasting

# Encrypted Key Exchange

- The Encrypted Key Exchange (EKE) protocol was designed by Steve Bellovin and Michael Merritt.
- It provides security and authentication on computer networks, using both symmetric and public-key cryptography in a novel way:
  - A shared secret key is used to encrypt a randomly generated public key.

# Encrypted Key Exchange

**(1)** Alice generates a random public-key/private-key key pair. She encrypts the public key, $K'$, using a symmetric algorithm and $P$ as the key: $E_p(K')$. She sends Bob

$A, E_P(K')$

**(2)** Bob knows $P$. He decrypts the message to obtain $K'$. Then, he generates a random session key, $K$, and encrypts it with the public key he received from Alice and $P$ as the key. He sends Alice

$E_P(E_{K'}(K))$

**(3)** Alice decrypts the message to obtain $K$. She generates a random string, $R_A$, encrypts it with $K$, and sends Bob

– $E_K(R_A)$

# Encrypted Key Exchange

**(4)** Bob decrypts the message to obtain $R_A$. He generates another random string, $R_B$, encrypts both strings with $K$, and sends Alice the result.

$E_K(R_A, R_B)$

**(5)** Alice decrypts the message to obtain $R_A$ and $R_B$. Assuming the $R_A$ she received from Bob is the same as the one she sent to Bob in step (3), she encrypts $R_B$ with $K$ and sends it to Bob.

$E_K(R_B)$

**(6)** Bob decrypts the message to obtain $R_B$. Assuming the $R_B$ he received from Alice is the same one he sent to Alice in step (4), the protocol is complete.

Both parties now communicate using $K$ as the session key.

# Outline

- Key-Exchange Algorithms
  - Diffie-Hellman
  - Station-to-Station Protocol
  - Encrypted Key Exchange
  - Fortified Key Negotiation

# Fortified Key Negotiation

- This scheme also protects key-negotiation schemes from poorly chosen passwords and man-in-the-middle attacks.
- It uses a hash function of two variables that has a very special property:
  - It has many collisions on the first variable while having effectively no collisions on the second variable.
    - $H'(x, y) = H(H(k, x) \bmod 2^m, x)$,

    where $H(k, x)$ is an ordinary hash function on $k$ and $x$

# Fortified Key Negotiation

- Here's the protocol. Alice and Bob share a secret password, $P$, and have just exchanged a secret key, $K$, using Diffie-Hellman key exchange.
- They use $P$ to check that their two session keys are the same (and that Eve is not attempting a man-in-the-middle attack), without giving $P$ away to Eve.

**(1)** Alice sends Bob

$H'(P, K)$

**(2)** Bob computes $H'(P, K)$ and compares his result with what he received from Alice. If they match he sends Alice

$H'(H(P, K))$

**(3)** Alice computes $H'(H(P, \mathrm{K}))$ and compares her result with what she received from Bob.

# Fortified Key Negotiation

- If Eve is trying a man-in-the-middle attack, she shares one key, $K_1$, with Alice, and another key, $K_2$, with Bob.
- To fool Bob in step (2), she has to figure out the shared password and then send Bob $H'(P, K_2)$.
- With a normal hash function she can try common passwords until she guesses the correct one, and then successfully infiltrate the protocol.
- But with this hash function, many passwords are likely to produce the same value when hashed with $K_1$.
- So when she finds a match, she will probably have the wrong password, and hence Bob in step 2 will not be fooled when he computes his own $H'(P, K)$.

# PART III: Outline

Special Algorithms for Protocols

- Multiple-Key Public-Key Cryptography
- Undeniable Digital Signatures
- Computing with Encrypted Data

# Multiple-Key Public-Key Cryptography

- This is a generalization of RSA. The modulus, $n$, is the product of two primes, $p$ and $q$.
- However, instead of choosing $e$ and $d$ such that $ed \equiv 1 \bmod ((p - 1)(q - 1))$, choose $t$ keys, $K_i$, such that
  - $K_1 * K_2 *...* K_t \equiv 1 \bmod ((p - 1)(q - 1))$
- Since
  - $M^{K1*K2*...*Kt} = M$

  this is a multiple-key scheme.
- If, for example, there are five keys, a message encrypted with $K3$ and $K5$ can be decrypted with $K1$ , $K2$ , and $K4$:
  - $C = M^{K3*K5} \bmod n$
  - $M = C^{K1*K2*K4} \bmod n$

# Part III: Outline

## Special Algorithms for Protocols

- Multiple-Key Public-Key Cryptography
- Undeniable Digital Signatures
- Computing with Encrypted Data

# Undeniable Signature Algorithm

- This undeniable signature algorithm is by David Chaum.
- First, a large prime, $p$, and a primitive element, $g$, are made public, and used by a group of signers.
- Alice has a private key, $x$, and a public key, $gx \bmod p$.
- To sign a message, Alice computes $z = m^x \bmod p$.
- Verification is a little more complicated.
  - **(1)** Bob chooses two random numbers, $a$ and $b$, both less than $p$, and sends Alice:

    $c = z^a(g^x)\text{b} \bmod p$
  - **(2)** Alice computes $t = x^{-1} \bmod (p - 1)$, and sends Bob:

    $d = c^t \bmod p$
  - **(3)** Bob confirms that

    $d \equiv m^a g^b \pmod p$

  If it is, he accepts the signature as genuine.

# Part III: Outline

## Special Algorithms for Protocols

- Multiple-Key Public-Key Cryptography
- Undeniable Digital Signatures
- Computing with Encrypted Data

# The Discrete Logarithm Problem

- There is a large prime, $p$, and a generator, $g$. Alice has a particular value for $x$, and wants to know $e$, such that
  - $g^e \equiv x \pmod{p}$
- This is a hard problem, and Alice lacks the computational power to compute the result.
- Bob has the power to solve the problem — he represents the government, or a large computing organization, or whatever.
- Here's how Bob can do it without Alice revealing $x$ => *next slide*

# Computing without revealing

**1)** Alice chooses a random number, $r$, less than $p$.

**(2)** Alice computes

$x' = xg^r \bmod p$

**(3)** Alice asks Bob to solve

$g^{e'} \equiv x' \pmod{p}$

**(5)** Bob computes $e'$ and sends it to Alice.

**(6)** Alice recovers $e$ by computing

$e = (e' - r) \bmod (p - 1)$

Alice computed e but did not reveal x.

# Readings Part I to III

- Applied Crypto: Sections of chapters 20, 22, 23.

# PART IV: Outline

- Multilevel security
  - Introduction
  - What Is a Security Policy Model?
  - The Bell-LaPadula Security Policy Model
  - Examples of Multilevel Secure Systems

# Multilevel philosophy

- Military database systems, which can hold information at a number of different levels of classification (confidential, secret, top secret, . . .) have to ensure that data can be read only by a principal whose level is at least as high as the data's classification.
- Sir Isaiah Berlin famously described thinkers as either foxes or hedgehogs: a fox knows many little things, while a hedgehog knows one big thing.
- The multilevel philosophy is the hedgehog approach to security engineering.

# Multilevel concepts

- Although multilevel concepts were originally developed to support confidentiality in military systems, there are now many commercial systems that use multilevel integrity policies.
  - For example, phone companies want their billing system to be able to see what's happening in their switching system, but not affect it.

# PART IV: Outline

- Multilevel security
  - Introduction
  - What Is a Security Policy Model?
  - The Bell-LaPadula Security Policy Model
  - Examples of Multilevel Secure Systems

# What Is a Security Policy Model?

- Where a top-down approach to security engineering is possible, it will typically take the form of:

  *threat model — security policy — security mechanisms.*

- The critical, and often neglected, part of this process is the security policy.

# Security Policy

- By a security policy, we mean a document that expresses clearly and concisely what the protection mechanisms are to achieve.
- It is driven by our understanding of threats, and in turn drives our system design.
- It will often take the form of statements about which users may access which data.
- It plays the same role in specifying the system's protection requirements, and evaluating whether they have been met, as the system specification does for general functionality.
- Indeed, a security policy may be part of a system specification.

# A typical corporate security policy

- Many organizations use the phrase ‘security policy’ to mean a collection of vapid statements.

**Megacorp Inc security policy**

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a “need-to-know”.
4. All breaches of this policy shall be reported at once to Security.

- What are the problems with this statements? Next slide =>

# Problems with the statemenst

- The first failing regards the central issue, namely ‘Who determines “need-to-know” and how?’
- Second, it mixes statements at a number of different levels (organizational approval of a policy logically should not be part of the policy itself).
- Third, there is a mechanism, but it’s implied rather than explicit: “staff shall obey” — but what does this mean they actually have to do?
  - Must the obedience be enforced by the system, or are users “on their honor?”
- Fourth, how are breaches to be detected and who has a specific duty to report them?

# Security policy model

- A *security policy model* is a clear and concise statement of the protection properties that a system, or generic type of system, must have.
- Its key points can typically be written down in a page or less.
- It is the document in which the protection goals of the system are agreed to by an entire community, or with the top management of a customer.

# Security target

- A *security target* is a more detailed description of the protection mechanisms that a specific implementation provides, and how they relate to a list of control objectives (some but not all of which are typically derived from the policy model).
- The security target forms the basis for testing and evaluation of a product.

# Protection profile

- A *protection profile* is like a security target but expressed in an implementation independent way to enable comparable evaluations across products and versions.
- This can involve the use of a semi-formal language or at least of suitable security jargon.
- A protection profile is a requirement for products that are to be evaluated under the *Common Criteria*.

# PART IV: Outline

- Multilevel security
  - Introduction
  - What Is a Security Policy Model?
  - The Bell-LaPadula Security Policy Model
  - Examples of Multilevel Secure Systems

# The Bell-LaPadula Security Policy Model

- The best-known example of a security policy model was proposed by David Bell and Len LaPadula in 1973, in response to U.S. Air Force concerns over the security of time-sharing mainframe systems.
- By the early 1970s, people had realized that the protection offered by many commercial operating systems was poor, and was not getting any better.
  - As soon as one operating system bug was fixed, some other vulnerability would be discovered.
- There was a serious scare when it was discovered that the Pentagon's World Wide Military Command and Control System was vulnerable to Trojan Horse attacks; this had the effect of restricting its use to people with a 'Top Secret' clearance.

# Reference monitor

- A study by James Anderson led the U.S. government to conclude that a secure system should do one or two things well; and that these protection properties should be enforced by mechanisms that were simple enough to verify and that would change only rarely.
- It introduced the concept of a *reference monitor*, a component of the operating system that would mediate access control decisions and be small enough to be subject to analysis and tests, the completeness of which could be assured.
- In modern parlance, such components—together with their associated operating procedures — make up the *Trusted Computing Base* (TCB).
- More formally, the TCB is defined as the set of components (hardware, software, human, etc.) whose correct functioning is sufficient to ensure that the security policy is enforced, or, more vividly, whose failure could cause a breach of the security policy.

# Which security properties?

- But what are these core security properties that should be enforced above all others?



# Classifications and Clearances

- World War II, and the Cold War that followed, led NATO governments to move to a common protective marking scheme for labelling the sensitivity of documents.
- *Classifications* are labels, which run upward from *Unclassified* through *Confidential*, *Secret*, and *Top Secret*. The details change from time to time.
- The original idea was that information whose compromise could cost lives was marked ‘Secret’ while information whose compromise could cost many lives was ‘Top Secret’.
- Government employees have *clearances* depending on the care with which they’ve been evaluated for a position;
  - in the United States, for example, a ‘Secret’ clearance involves checking FBI fingerprint files, while ‘Top Secret’ also involves background checks for the previous 5 to 15 years’ employment,

# Access Control and Levels

- The access control policy was simple: an official could read a document only if his clearance was at least as high as the document's classification.
- So an official cleared to 'Top Secret' could read a 'Secret' document, but not vice versa.
- The effect is that information may only flow upward, from Confidential to Secret to Top Secret, but it may never flow downward unless an authorized person takes a deliberate decision to declassify it.

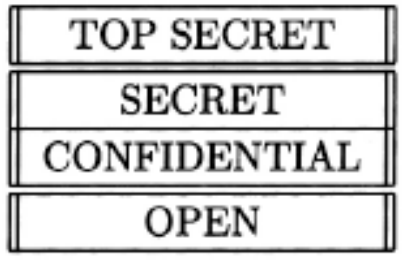



**Figure 7.2** Multilevel security.

# Document-handling rules

- There are also document-handling rules.
- Thus, a ‘Confidential’ document might be kept in a locked filing cabinet in an ordinary government office, while higher levels may require safes of an approved type, guarded rooms with control over photocopiers, and so on.
  - (The NSA security manual gives a summary of the procedures used with ‘Top Secret’ intelligence data.)

# Extra Levels

- The system rapidly became more complicated. The damage criteria for classifying documents were expanded from possible military consequences to economic harm and even political embarrassment.
- Britain has an extra level, ‘Restricted’, between ‘Unclassified’ and ‘Confidential’;
- The United States had this, too, but abolished it after the Freedom of Information Act was passed. America now has two more specific markings: ‘For Official Use only’ (FOUO) refers to unclassified data that can’t be released under the Freedom of Information Act, while ‘Unclassified but Sensitive’ includes FOUO plus material that might be released in response to a FOIA request.

# Extra Levels

- In Britain, restricted information is in practice shared freely, but marking everything ‘Restricted’ allows journalists and others involved in leaks to be prosecuted under Official Secrets law.
- Its other main practical effect is that an unclassified U.S. document sent across the Atlantic automatically becomes ‘Restricted’ in Britain, and then ‘Confidential’ when shipped back to the United States.

# Information Flow Control

- It was in the context of the classification of military and intelligence data that the *Bell-LaPadula* (*BLP*) model of computer security was formulated in 1973.
- It is also known as *multilevel security;* systems that implement it are often called *multilevel secure*, or *MLS* systems.
- Their basic property is that information cannot flow downward.
- More formally, the Bell-LaPadula model enforces two properties:
  - The *simple security property*: no process may read data at a higher level. This is also known as *no read up (NRU);*
  - The **-property*: no process may write data to a lower level. This is also known as *no write down (NWD).*

# Bell-LaPadula: Example of an attack scenario 1

- The *-property was Bell and LaPadula's critical innovation.
- It was driven by the fear of attacks using malicious code.
- An uncleared user might write a Trojan and leave it around where a system administrator cleared to 'Secret' might execute it;
  - It could then copy itself into the 'Secret' part of the system, read the data there and try to signal/copy it down somehow.

# Bell-LaPadula: Example of an attack scenario 2

- It's also quite possible that an enemy agent could get a job at a commercial software house and embed some code in a product that would look for secret documents to copy.
- If it could then copy them down to where its creator could read, the security policy would have been violated.
- Information might also be leaked as a result of a bug, if applications could write down.

# Mandatory Access Control

- So we must prevent programs running at ‘Secret’ from writing to files at ‘Unclassified’; or, more generally, prevent any process at High from signalling to any object (or subject) at Low.
- In general, when systems are built to enforce a security policy independently of user actions, they are described as having *mandatory access control*,
- This is opposed to the *discretionary access control* in systems such as Unix where users can take their own access decisions about their files.

# System Z

- The introduction of BLP caused some excitement: here was a straightforward security policy that was clear to the intuitive understanding.
- But John McLean showed that the BLP rules were not in themselves enough.
- He introduced *System Z*, defined as a BLP system with the added feature that a user can ask the system administrator to temporarily declassify any file from High to Low.
- In this way, Low users can read any High file without breaking the BLP assumptions.

# Tranquility Property

- Bell's argument was that System Z cheats by doing something the model doesn't allow (changing labels isn't a valid operation on the state), and McLean's argument was that it didn't explicitly tell him so.
- The issue is dealt with by introducing a *tranquility property*.
- The strong tranquility property says that security labels never change during system operation, while the weak tranquility property says that labels never change in such a way as to violate a defined security policy.

# Weak Property

- The motivation for the weak property is that in a real system we often want to observe the principle of least privilege, and start a process at the uncleared level, even if the owner of the process were cleared to 'Top Secret'.
- If she then accesses a confidential email, that session is automatically upgraded to 'Confidential';
  - And in general, her process is upgraded each time it accesses data at a higher level (this is known as the *high water mark* principle).

# Accumulating Security Labels

- The practical implication is that a process accumulates the security label or labels of every file that it reads, and these become the default label set of every file that it writes.
- So a process that has read files at ‘Secret’ and ‘Crypto’ will thereafter create files marked (at least) ‘Secret Crypto’.
- This will include temporary copies made of other files.
- If it then reads a file at ‘Top Secret’, all files it creates after that will be labelled ‘Top Secret Crypto’, and it will not be able to write to any temporary files at ‘Secret Crypto.’

# BLP insufficiency

- Finally it's worth noting that even with this refinement, BLP still doesn't deal with the creation or destruction of subjects or objects (which is one of the hard problems of building a real MLS system).

# Harrison-Ruzzo-Ullman

- The *Harrison-Ruzzo-Ullman* model tackles the problem of how to deal with the creation and deletion of files, an issue on which BLP is silent.
- It operates on access matrices and verifies whether there is a sequence of instructions that causes an access violation.
- This is more expressive than BLP, but more complex and thus less tractable.

# Role-based access control

- Finally, the policy model getting the most attention at present from researchers is *role-based access control* (RBAC).
- This sets out to provide a more general framework for mandatory access control than BLP in which access decisions don't depend on users' names but on the functions they are currently performing within the organization.
- Transactions that may be performed by holders of a given role are specified, then mechanisms for granting membership of a role (including delegation) are given.

# Role-based access control

- Roles, or groups, had for years been the mechanism used in practice in organizations such as banks to manage access control; the RBAC model starts to formalize this.
- It can deal with integrity issues as well as confidentiality, by allowing role membership (and thus access rights) to be revised when certain programs are invoked.
  - Thus, for example, a process calling untrusted software that had been downloaded from the Net might lose the role membership required to write to sensitive system files.

# The Biba Model

- This model is due to Ken Biba, which is often referred to as “Bell-LaPadula upside down.”
- It deals with integrity alone and ignores confidentiality entirely.
- The key observation is that confidentiality and integrity are in some sense dual concepts: confidentiality is a constraint on who can read a message, while integrity is a constraint on who may have written or altered it.

# Example: an ECG system

- As a concrete application, an electronic medical device such as an ECG may have two separate modes: calibration and use.
- The calibration data must be protected from being corrupted by normal users, who will therefore be able to read it but not write to it;
  - When a normal user resets the device, it will lose its current user state (i.e., any patient data in memory) but the calibration will remain unchanged.

# Example: an ECG system

- To model such a system, we can build a multilevel integrity policy with the rules that we must only read up (i.e., a user process can read the calibration data) and write down (i.e., a calibration process can write to a buffer in a user process);
  - but we must never read down or write up.

# PART IV: Outline

- Multilevel security
  - Introduction
  - What Is a Security Policy Model?
  - The Bell-LaPadula Security Policy Model
  - Examples of Multilevel Secure Systems

# SCOMP

- One of the most important products was the *Secure Communications Processor* (SCOMP), a Honeywell derivative of Multics, launched in 1983.
- This was an implementation of what the U.S. Department of Defense believed it wanted for handling messaging at multiple levels of classification.
- SCOMP had formally verified hardware and software, with a minimal kernel and four rings of protection (rather than Multics' seven) to keep things simple.
- Its operating system, STOP, used these rings to maintain up to 32 separate compartments, and to allow appropriate one-way information flows between them.

# SCOMP

- SCOMP was used in applications such as military *mail guards*, specialized firewalls that typically allow mail to pass from Low to High, but not vice versa.
- (In general, a device that makes information flow one way only is known as a *data diode*.)
- SCOMP's successor, XTS-300, supports C2G, the Command and Control Guard.
- This is used in the *time-phased force deployment data* (TPFDD) system whose function is to plan U.S. troop movements and associated logistics.
- Overall, military plans are developed as TPFDDs, at a high classification level, then distributed at the appropriate times as commands to lower levels for implementation.

# Orange Book

- SCOMP's most significant contribution was to serve as a model for the *Orange Book*, also known as the *Trusted Computer Systems Evaluation Criteria* (TCSEC).
- This was the first systematic set of standards for secure computer systems, being introduced in 1985 and finally retired in December 2000.
- Although it has since been replaced by the Common Criteria, the Orange Book was enormously influential, not just in the United States but among allied powers; countries such as Britain, Germany, and Canada based their own national standards on it, and these national standards were finally subsumed into the Common Criteria.

# Blacker

- Blacker was a series of encryption devices designed to incorporate MLS technology.
- Previously, encryption devices were built with separate processors for the ciphertext, or *Black*, end, and the cleartext, or *Red*, end.
- Various possible failures can be prevented if one can coordinate the Red and Black processing.

# MLS Unix, CMWs, and Trusted Windowing

- Most of the available MLS systems are modified versions of Unix, and an example is AT&T's System V/MLS.
- *Compartmented mode workstations* (CMWs) allow data at different levels to be viewed and modified at the same time by a human operator, and ensure that labels attached to the information are updated appropriately.
- CMWs allow an analyst to view the 'Top Secret' data in one window, compose a report in another, and have mechanisms to prevent the accidental copying of the former into the latter.

# The NRL Pump

- It was soon realized that simple mail guards and crypto boxes were too restrictive, as many more networked services were developed besides mail.
- Traditional MLS mechanisms (such as periodic read-downs) are inefficient for real-time services.
- The US Naval Research Laboratory (NRL) therefore developed the *Pump*, a one-way data transfer device (a data diode) using buffering to allow one-way information flow.
- The attraction of this approach is that it is possible to build MLS systems by using pumps to connect separate systems at different security levels.

# The NRL Pump

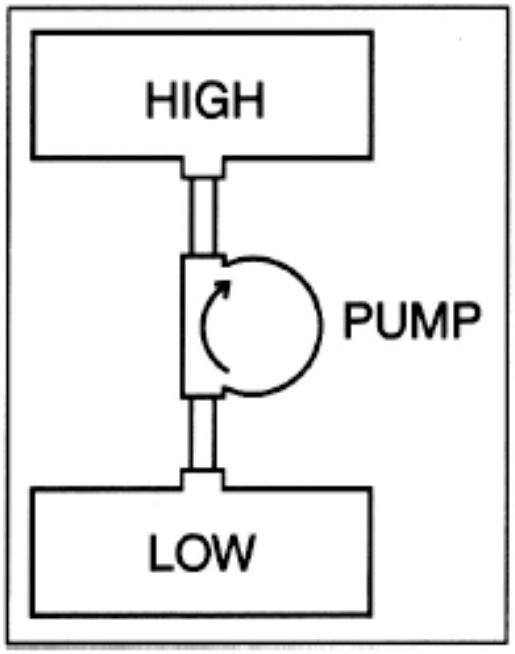



**Figure 7.3** The NRL pump.

# Logistics Systems

- Military stores, like government documents, can have different classification levels.
- The Royal Air Force's Logistics Information Technology System (LITS) was a 10-year (1989–1999), $500 million project to provide a single stores management system for the RAF's 80 bases.
- It was designed to operate on two levels: 'Restricted' for the jet fuel and boot polish, and 'Secret' for special stores such as nuclear bombs.
- It was initially implemented as two separate database systems connected by a pump to enforce the MLS property.

# LITS Project

- The project became a classic tale of escalating costs driven by creeping requirements changes.
- One of these changes was the easing of classification rules at the end of the Cold War.
- As a result, it was found that almost all the ‘Secret’ information was now static (e.g., operating manuals for air-drop nuclear bombs, which are now kept in strategic stockpiles rather than at airbases).
- To save money, the ‘Secret’ information is now kept on a CD and locked up in a safe.

# Purple Penelope

- In recent years, most governments' information security agencies have been unable to resist user demands to run standard applications (such as MS Office), which are not available for multilevel secure platforms.
- One response is 'Purple Penelope'. This software, from Britain's Defence Evaluation and Research Agency, puts an MLS wrapper round a Windows NT workstation.

# Future MLS Systems

- The MLS industry sees an opportunity in using its products as platforms for firewalls, Web servers, and other systems that are likely to come under attack.
- Thanks to the considerable effort that has often gone into finding and removing security vulnerabilities, MLS platforms can give more assurance than commodity operating systems that, even if the firewall or Web server software is hacked, the underlying operating system is unlikely to be.

# Future MLS Systems

- The usual idea is to use the MLS platform to separate trusted from untrusted networks, then introduce simple code to bypass the separation in a controlled way.
- In fact, one of the leading firewall vendors (TIS) was until recently focused on developing MLS operating systems, while Secure Computing Corporation, Cyberguard, and Hewlett-Packard have all offered MLS-based firewall products.

# Future MLS Systems

- Perhaps the real future of multilevel systems is not in confidentiality, but integrity.
- Many fielded systems implement some variant of the Biba model (even though their designers may never have heard the word “Biba”).
- For example:
  - In an electricity utility, the critical operational systems such as power dispatching should not be affected by any others.
  - They can be observed by, but not influenced by, the billing system.
  - Similarly, the billing system and the power dispatching system both feed information into the fraud detection system.

# Part IV Readings

- Ross Anderson, Security Engineering
  - Chapter 7.