



# CS 524 A

Introduction to Cloud Computing

Lecture 12:

Security and Identity Management in the Cloud  
Part I

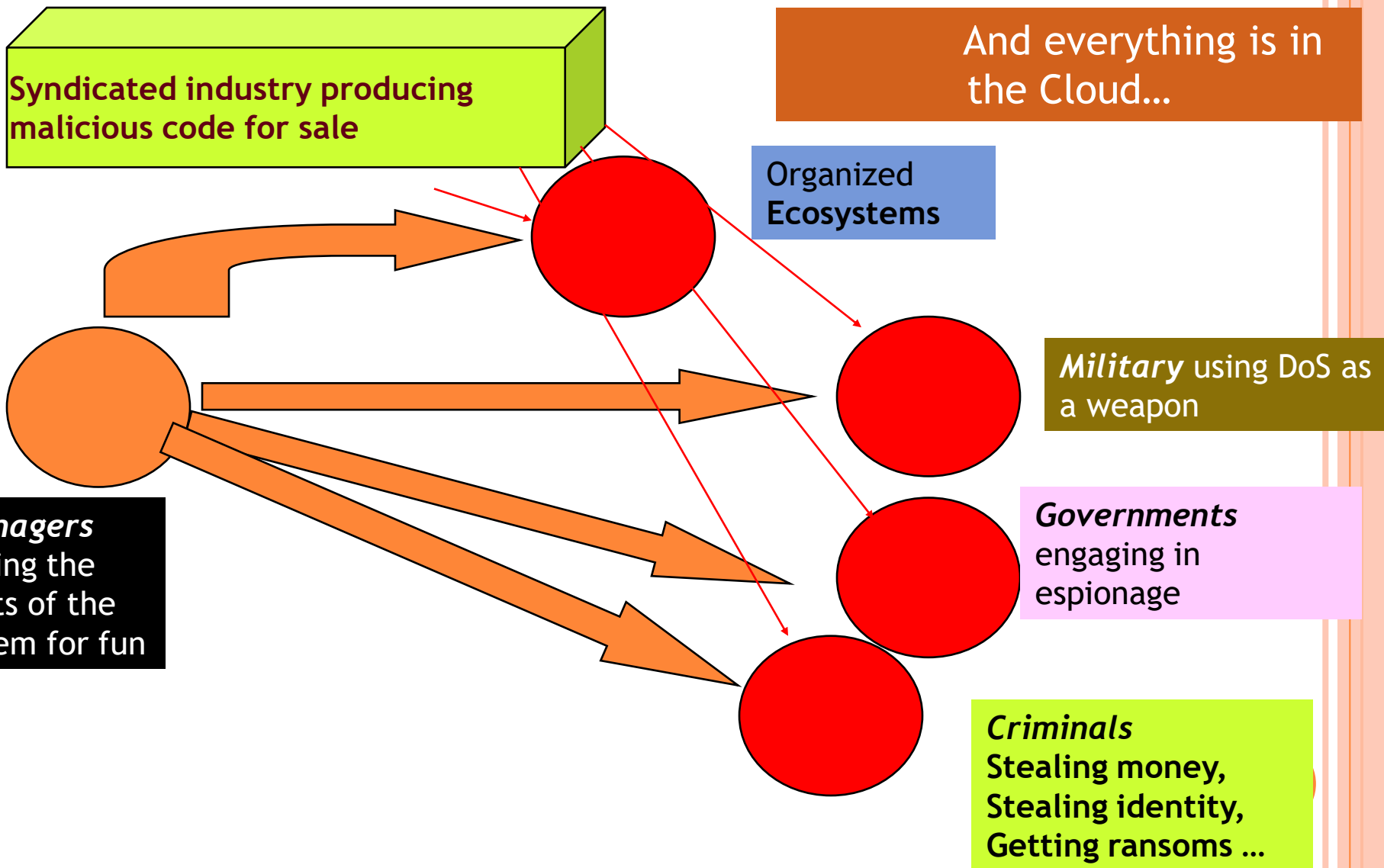
**WARNING:** THIS IS JUST THE TIP OF THE...



# OUTLINE

- The threat trends
- Basic security properties and cryptographic means to support them
  - Symmetric cryptography
  - Public-Key cryptography (with an example of RSA)
  - Cryptographic signatures
- Public Key Infrastructure (or lack thereof...)

# THE THREATS TREND



# AN EXAMPLE: E-MAIL...

- Can you send a message that is truly private?
- Do you know who really sent you a message?
  - Do you know what will happen when you open an attachment?
  - Do you know what will happen when you click on a link?
- Can you be sure that the message you know was sent to you by a friend was not *modified* in transit?
- Can you send a truly anonymous message?
- Can you really *delete* a message
- What is the (typical) way for an adversary to read your messages?

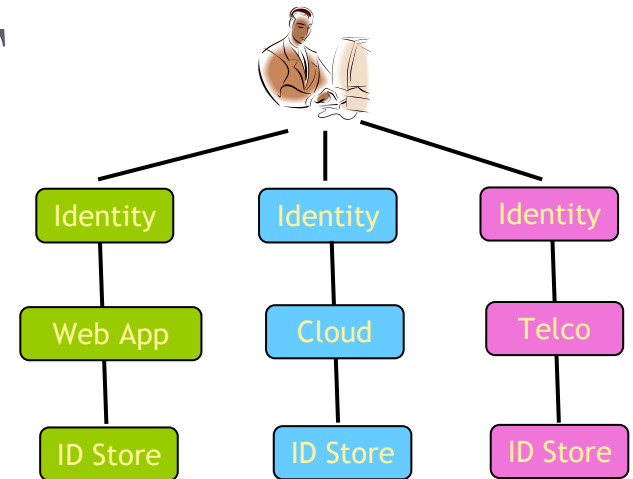
## ANOTHER EXAMPLE: BUYING ON-LINE

- Can you be sure that the information you are supplying (including your credit card number and code—which proves your possession of the card) is not reached by a thief?
- Can you be really sure that you are paying to the real merchant?
- Can you buy anonymously?
- Can you deny the payment after receiving the product (i.e., can the merchant prove that you have ordered the product)?



# KEY DRIVERS OF IDENTITY AND ACCESS MANAGEMENT

- Improvement of user experience
  - Containment of proliferation of application-specific identities
  - Consistent user access from varied devices to different applications



- Control and protection of **personal information**
- Dynamic **access control**
- **Mitigation** of security risks
- **Compliance with regulatory requirements**, including those pertinent to the “B” column of cloud cases
- **Enablement of new applications** (e.g., identity-based services, social networking, targeted advertisement, and personalized services)

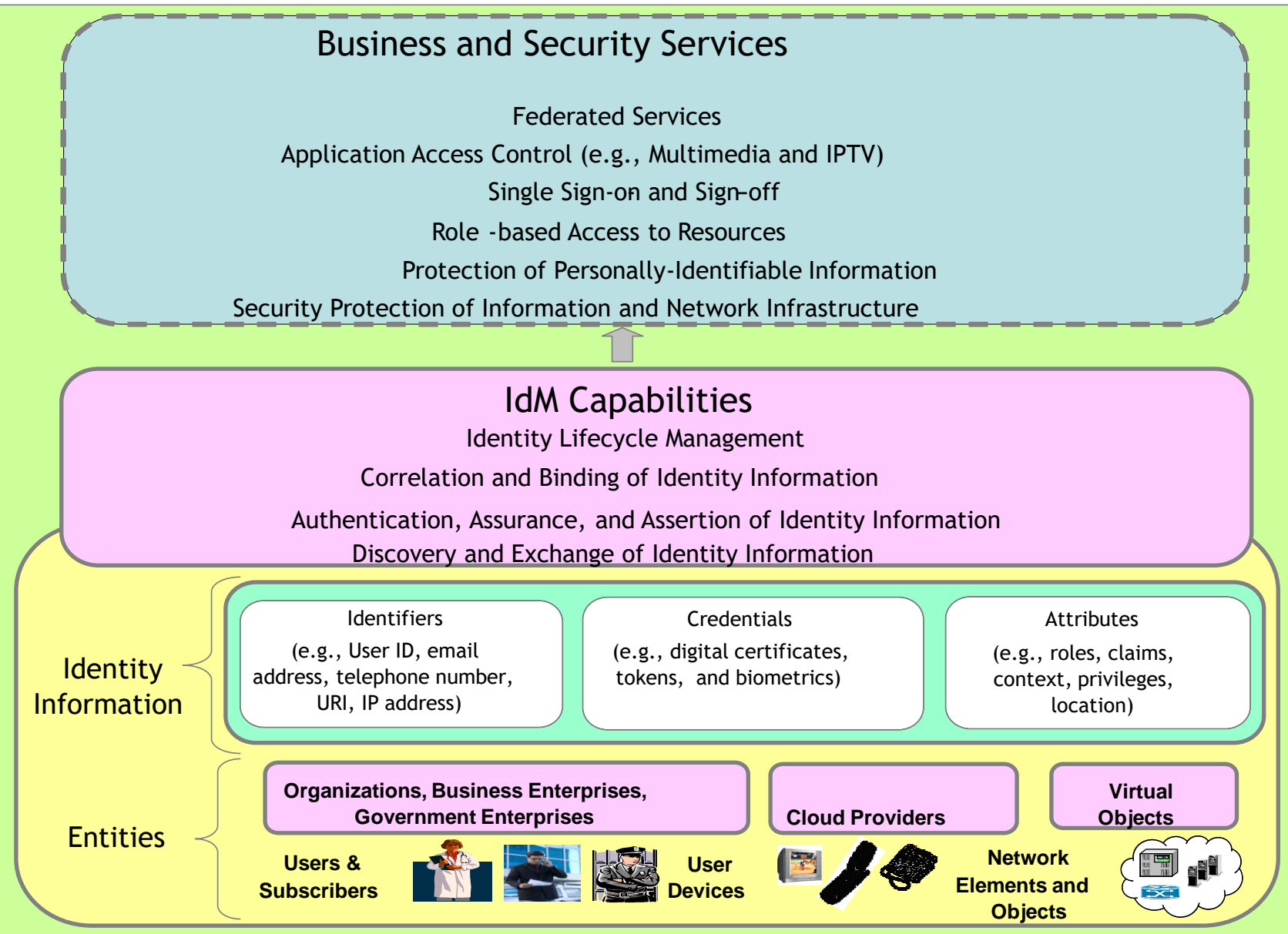
# WHY IDENTITY MANAGEMENT?

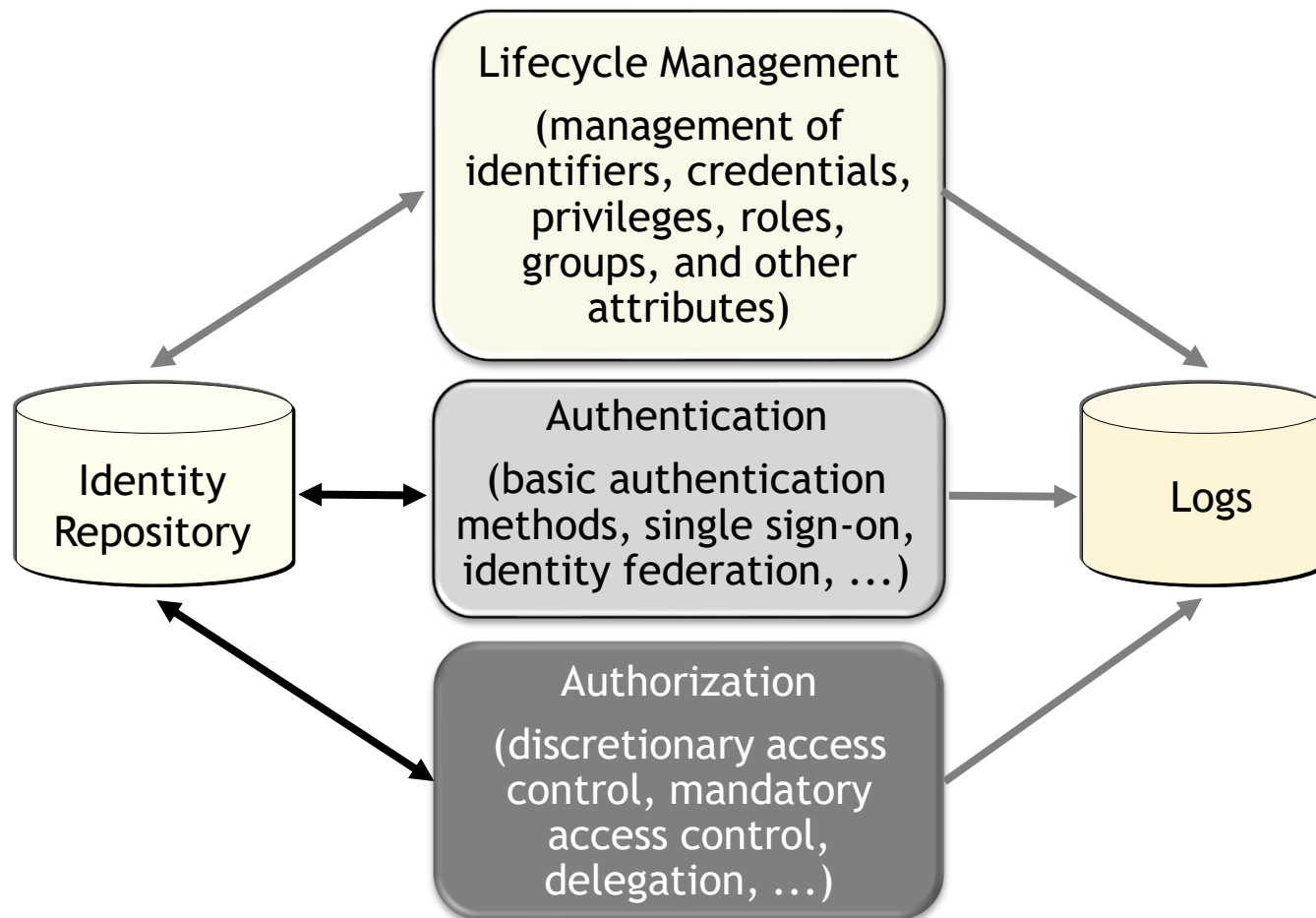
- “The Internet will be everywhere, from every mote to interstellar communication”
- “The Internet needs to have a secure identity layer”

Both statements attributed to **Vint Cerf** in Jan Camenisch’s talk at the 7<sup>th</sup> Framework Programme conference established by the European Commission

<http://www.primelife.eu/images/stories/talks/2009-04-21-zisc-primelife.pdf>







## SCOPE OF IDENTITY AND ACCESS MANAGEMENT

# BASIC SECURITY PROPERTIES

- Confidentiality

Keeping information secret from unintended users

- Authentication

Confirming the identity of the presenter of the information

- Authorization

Determining whether a give resource may be accessed

- Integrity

Ensuring that a message received was the one that was actually sent

- Non-repudiation

Ensuring that a party that has signed a contract cannot later deny having signed it



## CREDENTIALS FOR THE USER AUTHENTICATION

May be disclosed  
or guessed

### 1. What Alice knows

- Password
- (Cryptographic keys)
- Image or pattern
- Answers to security questions

### 2. What Alice has

- Security token
- Smart card
- Mobile phone

May be stolen  
or cloned

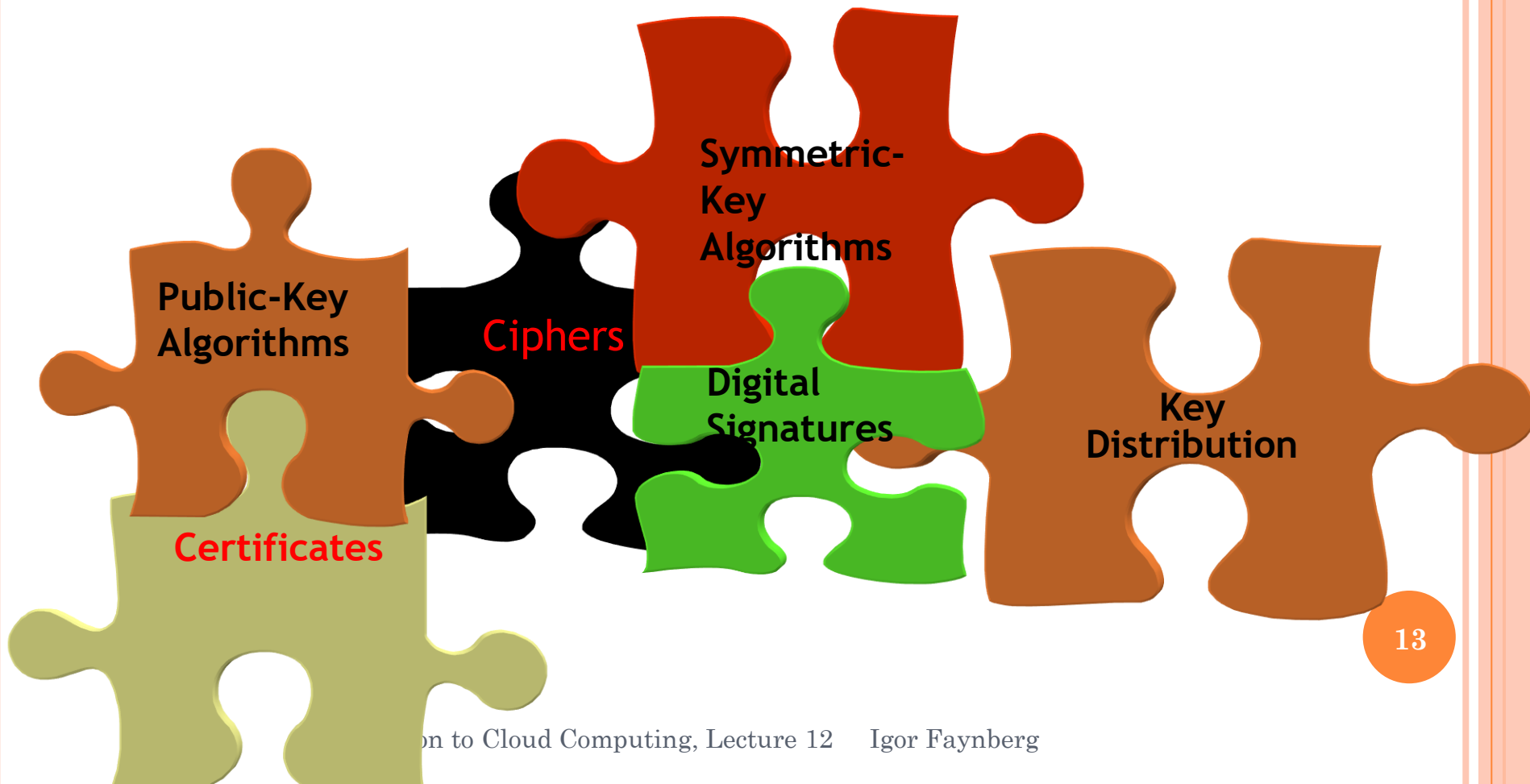
### 3. What Alice is

- Fingerprint
- Iris
- Voiceprint

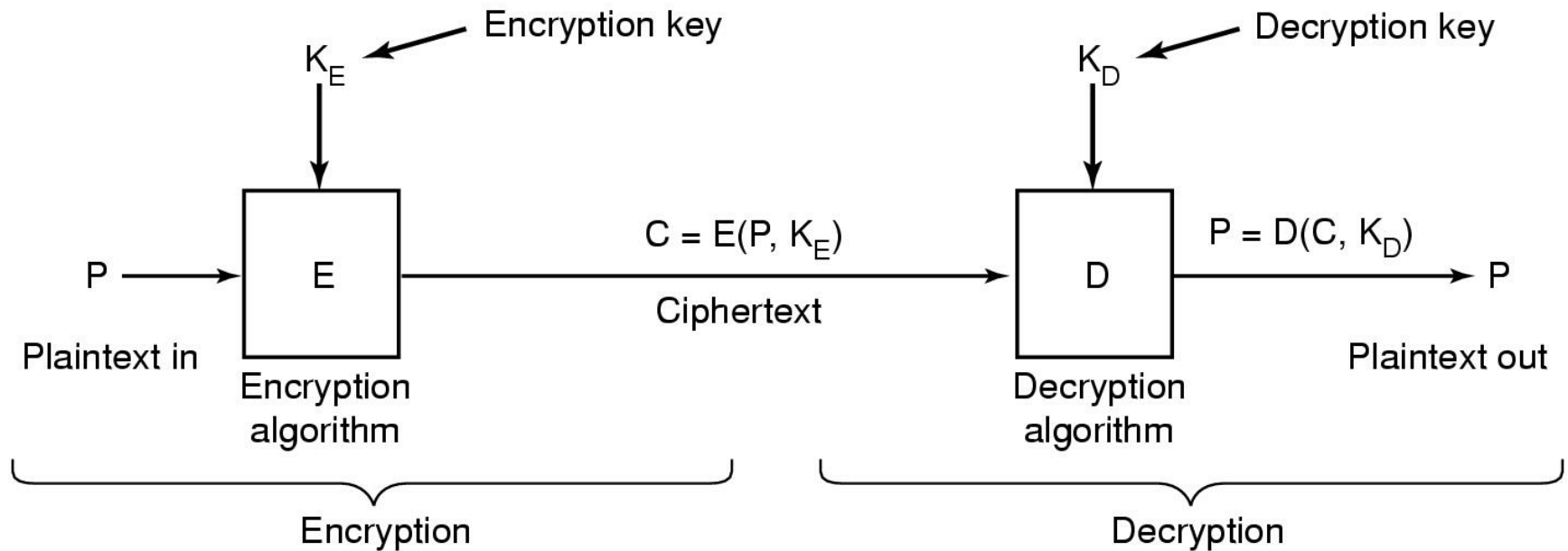
May be cloned

# ENSURING CONFIDENTIALITY, INTEGRITY, AND NON-REPUDIATION: CRYPTOGRAPHY

*κρυπτωσ* (secret) *γραφια* (writing)



# BASICS OF CRYPTOGRAPHY



*All algorithms must be public; only the keys are secret*  
August Kerckhoff, 1883

# SECRET-KEY (OR *SYMMETRIC KEY*) CRYPTOGRAPHY

- Given the encryption key,  $K_E$  it is easy to find the decryption key  $K_D$
- Problem: Key distribution**
  - An example: *One-Time Pad*

$$K_E = K_D$$

$\oplus$ (XOR)	0	1
0	0	1
1	1	0

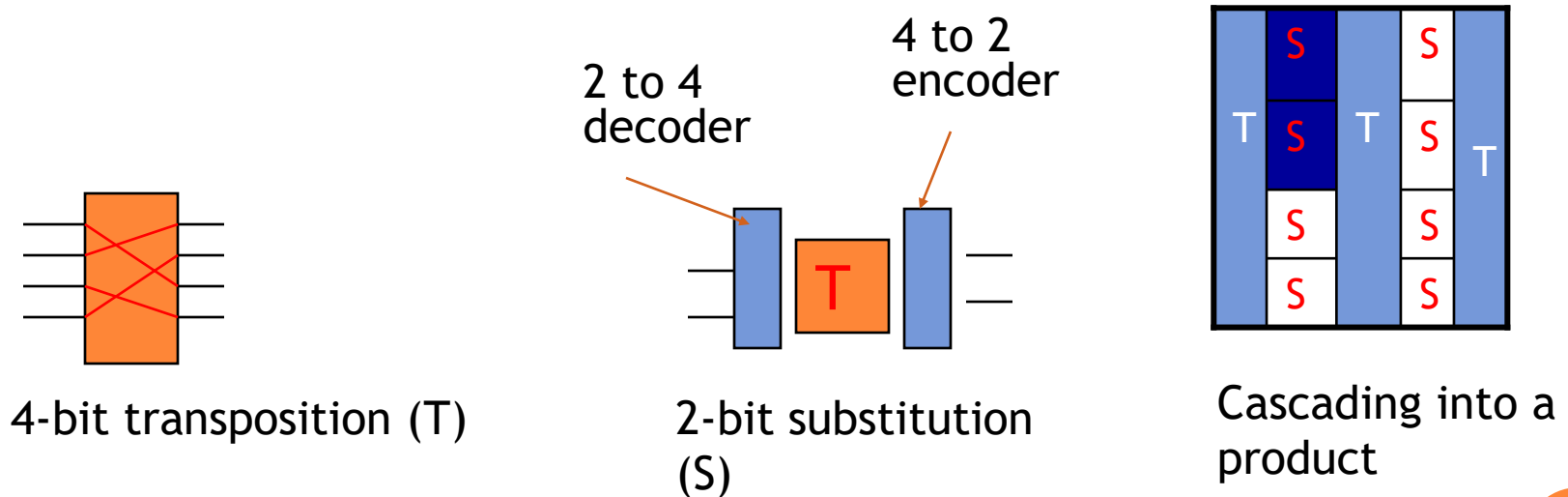
Plaintext: 001110011010010110

Key: 100100100111110110

Cyphertext: 1 01010111101100000

# MODERN SYMMETRIC-KEY ( $K_D = K_E$ ) ALGORITHMS

- Combine transpositions and substitutions and cascade them to make the algorithms very complex (to prevent cryptanalysis even when large amounts of cyphertext are available)
- Often use *block ciphers*





## SOME COMMON SYMMETRIC-KEY CRYPTOGRAPHIC ALGORITHMS (AFTER A. TANENBAUM)

Cipher	Key size (bits)	Characteristics
Rijndael (AES)	128-256	Best
Triple DES	168	Second best
Serpent, Twofish	128-256	Very strong
IDEA	128	Good (but patented)
RC5	128-256	Good (but patented)
RC4	1-2048	Some keys are weak
DES	56	Weak

# PUBLIC-KEY CRYPTOGRAPHY

A (public key, private key) pair

- Publish the public key (= encryption key)  $K_E$
- Keep the private key (= decryption key)  $K_D$  secret

Two essential requirements:

1)  $K_D * K_E = I$

2) It is very hard (i.e, computationally infeasible) to obtain  $K_D$  from  $K_E$

- To send a message  $M$  to you, I send  $K_E(M)$
- You decrypt it, obtaining:  $K_D(K_E(M)) = M$

# RSA (RIVEST, SHAMIR, ADLEMAN)

- Parameters:  $p, q, n, z, d, e$

1. Choose, large (1024 bits) primes:  $p, q$
2. Compute  $n = pq, z = \phi(n) = (p-1)(q-1)$
3. Choose the exponent  $e$  relatively prime to  $z$
4. Find  $d: ed \equiv 1 \pmod{z}$

The probability that  $P$  and  $n$  are not relatively prime is extremely low!

- Keys: **public**,  $(e, n)$ ; **private**,  $(d, n)$ ;

- Encryption and decryption:

- Break the plaintext into largest equal even-digit blocks ( $P$ ) shorter than  $n$  bits
- Encrypt each block  $P$  by computing  $C = E(P) \equiv P^e \pmod{n}$
- Decrypt  $C$  by computing  $D(C) \equiv C^d \pmod{n} \equiv P^{ed} \pmod{n} \equiv P^{k\phi(n)+1} \pmod{n} \equiv P^{k\phi(n)} P \pmod{n} \equiv P \pmod{n}$

## Euler's Theorem:

If  $n > 0$  and  $e$  and  $d$  are integers, such that  $(a, m) = 1$ , then  $a^{\phi(m)} \equiv 1 \pmod{m}$ .

Plaintext: 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

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Public key: (13, 2537)


Private key: (937, 2537)

## RSA: AN EXAMPLE

- $p = 43, q = 59; n = 43 * 59 = 2537; \phi(n) = 42 * 58 = 2436$

- Exponent  $e = 13; (e, \phi(n)) = (13, 42 * 58) = 1;$

- Block length is 4

  $d = 937$

$937 * 13 \equiv 1 \pmod{2436}$

## PUBLIC KEY CRYPTOGRAPHY

$1520^{13} \equiv 95 \pmod{2537}$

1520 0111 0802 1004 2402 1724 1519 1406 1700 1507 2423

$E(P) \equiv P^e \pmod{n}$

0095 1648 1410 1299 0811 2333 2132 0370 1185 1457 1084

$0095^{937} \equiv 1520 \pmod{2537}$

$P \equiv C^d \pmod{n}$

# ANALYSIS OF RSA

- 100-digit primes  $p$  and  $q$ , the encryption exponent  $e$ , and its inverse,  $d$ , can be found in a few minutes of computer time. Now, both keys are ready!
- Modular exponentiation for encryption can be performed in a few seconds when  $e$  is chosen to be small
- Decryption (private key operations) takes longer, in general, because  $d$  depends on  $e$  (already chosen)
- Any known method of finding  $d$  from  $e$  and  $n$  is based on factoring  $n$

RSA security is based on the difficulty of factoring large integers

# PROPERTIES OF RSA

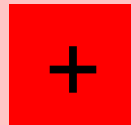
- **The algorithm is secure** because of the *perceived* complexity (nothing has been proven!) of factoring  $N$ . **Factoring a 500-digit number should take 1025 years using a CPU with 1 microsecond instruction time.**
- **Encryption and decryption are inverse and commutative** (an important property for digital signatures).
- **The algorithm is slow** (compared to DES and other symmetric algorithms with much

RSA might be prohibitively slow when dealing with large blocks of data. It is typically used for one-time session key distribution for a symmetric-key algorithm (such as triple-DES)

# TYPICAL USE OF RSA FOR KEY DISTRIBUTION IN SYMMETRIC CRYPTOGRAPHY (HYBRID ENCRYPTION)

Sender randomly generates a symmetric key  $K$  and sends:

$K$ , encrypted using RSA with the public key of the receiver



Plaintext encrypted with the symmetric-key algorithm  $E_K$

Receiver

1. Decrypts  $K$  using the private key
2. Decrypts the message using  $D_K$

# PUBLIC-KEY ALGORITHMS

- Knapsack (Merkle and Hellman, 1978)—based on NP-completeness of the Knapsack problem
  - Was the first public-key algorithm, but is considered unsecure and not used
- El Gamal (1985) is based on difficulties computing discrete logarithm
  - Is more computationally-intensive than RSA
  - Is totally unencumbered by copyright and patents
- RSA
  - Users might have problems with proper generation of primes (some primes or pseudo-primes may aid factoring)
  - It is not appropriate for use in situations where key generation occurs regularly
  - Patents expired
- Elliptic-Curve Cryptography (ECC) (Miller and Koblitz, 1985) hinges on the intractability of the discrete logarithm problem in the algebraic system defined on elliptic curve points
  - Uses smaller keys than RSA or El Gamal
  - Is significantly faster than RSA (for the same security)
  - Is patented



# BACK TO BASIC SECURITY PROPERTIES

- Confidentiality
  - Keeping information secret from unintended users
  - Achieved through encryption
- Authentication
  - Confirming the identity of the presenter of the information
  - Achieved by *authentication protocols*
- Authorization
  - Determining whether a user may have access to a resource
  - Achieved through access control lists (ACLs), implementing policies, and also by *token-based authorization protocols* (OAuth, OpenStack Keystone tokens, etc.)
- Integrity
  - Ensuring that a message received was the one that was actually sent
  - Achieved by cryptographic means called *signatures*
- Non-repudiation
  - Ensuring that a party that has signed a contract cannot later deny having signed it
  - Achieved by either third-party attestation or asymmetric cryptographic means

# HASHING AND MESSAGE DIGESTS

- A *hash* is a one-way function  $h$  (i.e., it is fairly “easy” to compute the *message digest*  $h(P)$ , but is “very hard” to solve the equation  $h(x) = M$  for  $x$ ).
- Typically, the size of  $h(P)$  is much smaller than the size of  $P$ ; however
  - $h(P)$  “preserves”  $P$  in that if a bit changes in  $P$ , it must produce a “drastically” different hash (i.e., output changes are uncorrelated with input change);
  - If  $n$  ( $n > 1000$ ) inputs are selected at random, then any particular bit in the  $n$  resulting outputs must be on about half the time
  - Each output should have about half the bits on

# TWO POPULAR MESSAGE DIGEST ALGORITHMS

- Message Digest (MD5) (Rivest, 1992)
  - Produces a 64-bit result
  - Supersedes the previous four MDs in a series, but they are all “broken”
- Secure Hash Algorithm (SHA-1)
  - produces a 160-bit result
  - Is standardized by NIST in FIPS 180-1
  - Is replacing MD5
  - Is being replaced by SHA-2
- NIST announced **Keccak** as the winner of the SHA-3 Cryptographic Hash Algorithm Competition on October 2, 2012  
(<http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf>)

# WITH A GOOD SYMMETRIC KEY ALGORITHM $E$ WE CAN BUILD A HASH FUNCTION BY *DIGESTING* A BIG PLAINTEXT FILE $P$

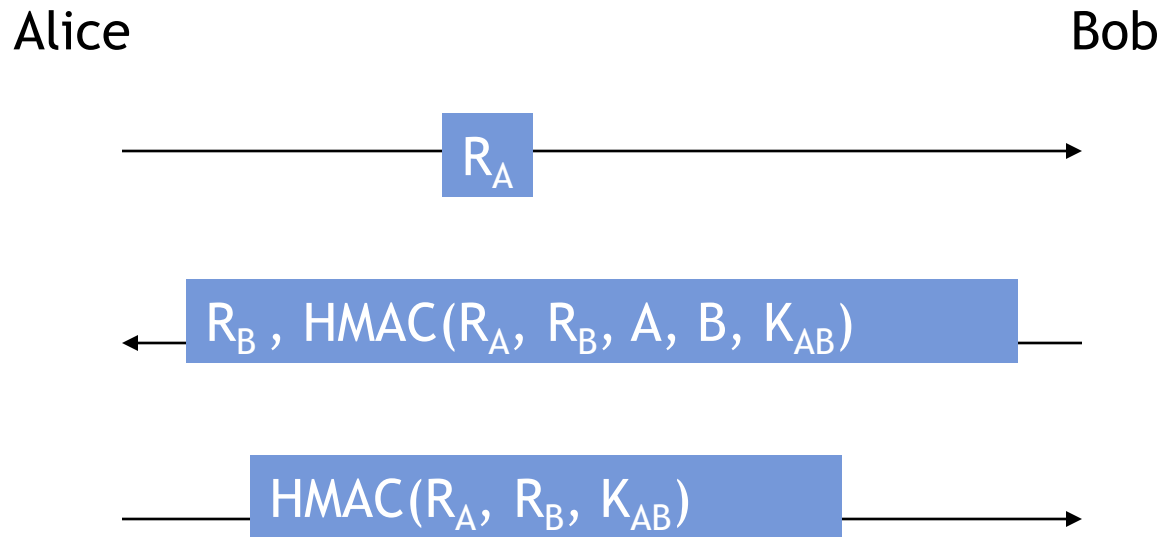
- Break input into chunks of the desired hash size  $s$ :  $P = p_1 \mid p_2 \mid p_3 \dots \mid p_n$
- Take a constant (initial vector)  $m_0$  of size  $s$  and a  $p_1$  as a key and compute  $m_1 = E_{p_1}(m_0)$
- Then continue for  $i=2, \dots, n$ :  $m_i = E_{p_i}(m_{i-1})$
- $m_n = E_{p_n}(m_{n-1})$  is the desired hash!

CONVERSELY, GIVEN A HASH FUNCTION  $h$ , WE CAN BUILD AN ENCRYPTION ALGORITHM  $E$  TO COMPUTE  $E(P)$  FOR THE PLAINTEXT  $P$  OF ARBITRARY SIZE

- Take the key,  $K$ , and compute  $m_1 = h(K)$
- Then continue for  $i=2, \dots, n = \lceil |P| / |K| \rceil + 1$ :
$$m_i = h(m_{i-1})$$
- Concatenate these chunks into a one-time pad key  $M = m_1 \mid m_2 \mid m_3 \dots \mid m_n$
- $E = P \oplus M$

# A CLASS OF PROTOCOLS THAT WORK (HMAC)

*Hashed Message Authentication Code (HMAC)*, in general,  
is the *hash* of (some data + shared secret)



Knowing  $K_{AB}$  is necessary in order to compute HMAC!

# KEY DISTRIBUTION CENTER (KDC)

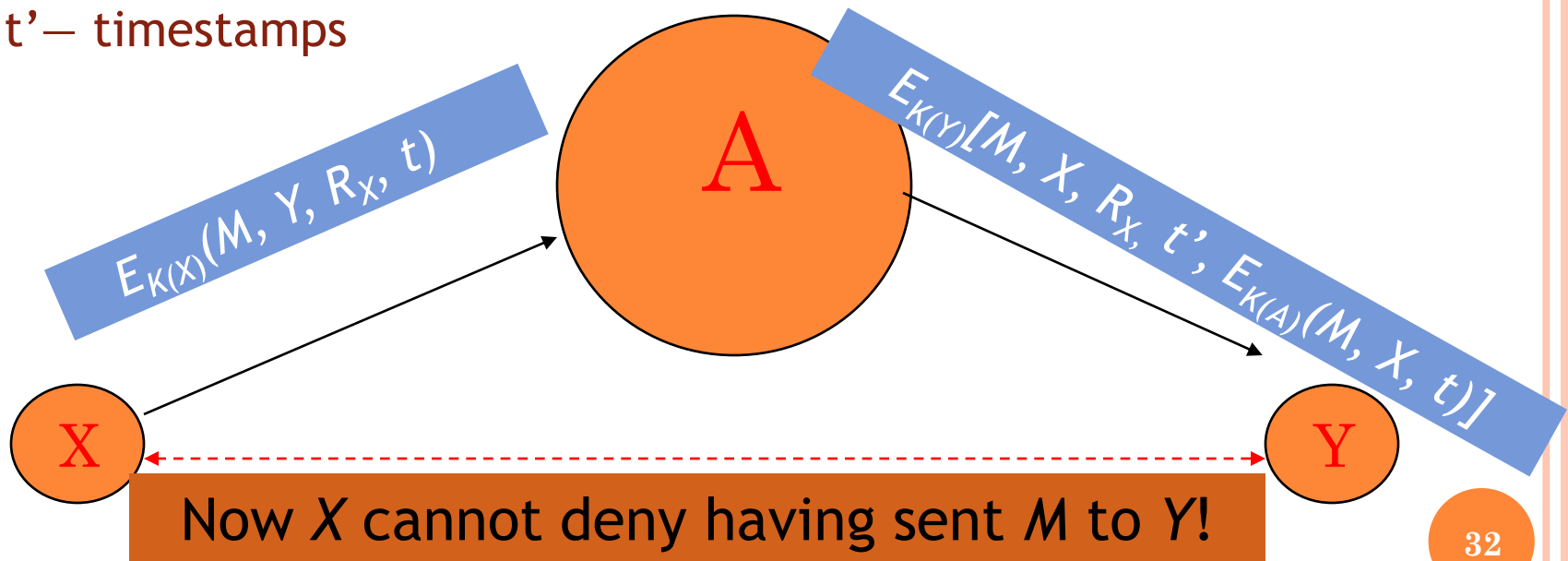
- If a process needs to talk to  $n$  other processes, it will need to share  $n$  keys. As  $n$  grows, key management becomes a burden...
- Another approach: Each user has a key shared with KDC, and all authentication and session key management go through KDC

# NON-REPUDIATION WITH SYMMETRIC-KEY DIGITAL SIGNATURES

A single third party (Central Authority,  $A$ ) keeps everyone's keys

- $K(X)$ — $X$ 's key with  $A$
- $M$ —the message from  $X$  to  $Y$
- $Y$ —the receiver's identifier
- $R_X$  — a random number
- $t, t'$ — timestamps

- $K(Y)$ — $Y$ 's key with  $A$
- $K(A)$ —the key only  $A$  knows
- $X$ —the sender's identifier

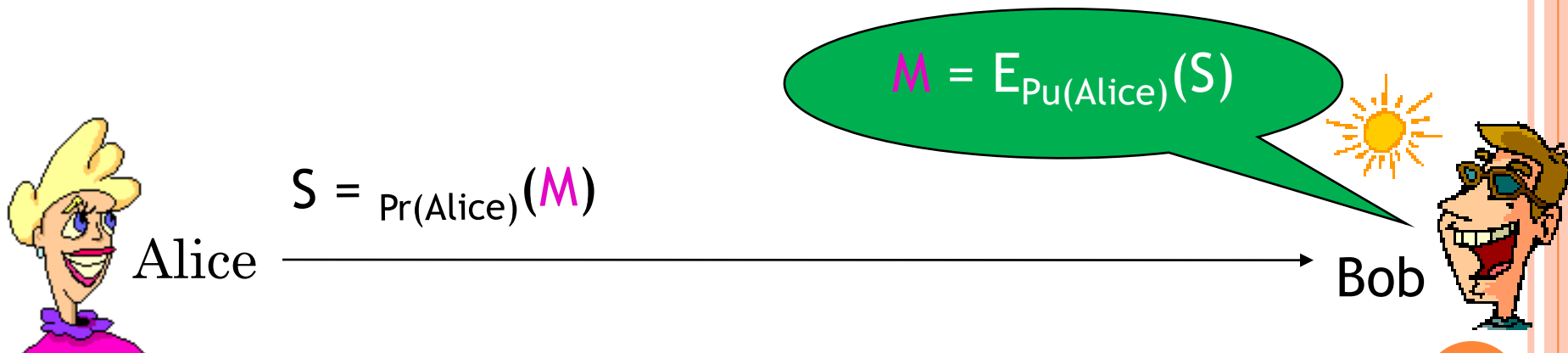




# Non-Repudiation with Public-Key Digital Signatures

Works with any public key algorithm with the property  $E[D(P)] = P$   
(RSA is one of them, but there are others)

- $Pu(\text{Alice})$ —Alice's public key
- $Pr(\text{Alice})$ —Alice's private key



**No third party needed!**

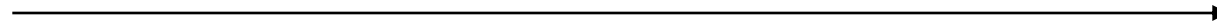
# Non-Repudiation and Confidentiality with Public-Key Digital Signatures

Again, use any public key algorithm with the property  $E[D(P)] = P$

- $Pu(Alice)$ —Alice's public key
- $Pr(Alice)$ —Alice's private key
- $Pu(Bob)$ —Bob's public key
- $Pr(Bob)$ —Bob's private key



$$S = E_{Pu(Bob)}[D_{Pr(Alice)}(M)]$$

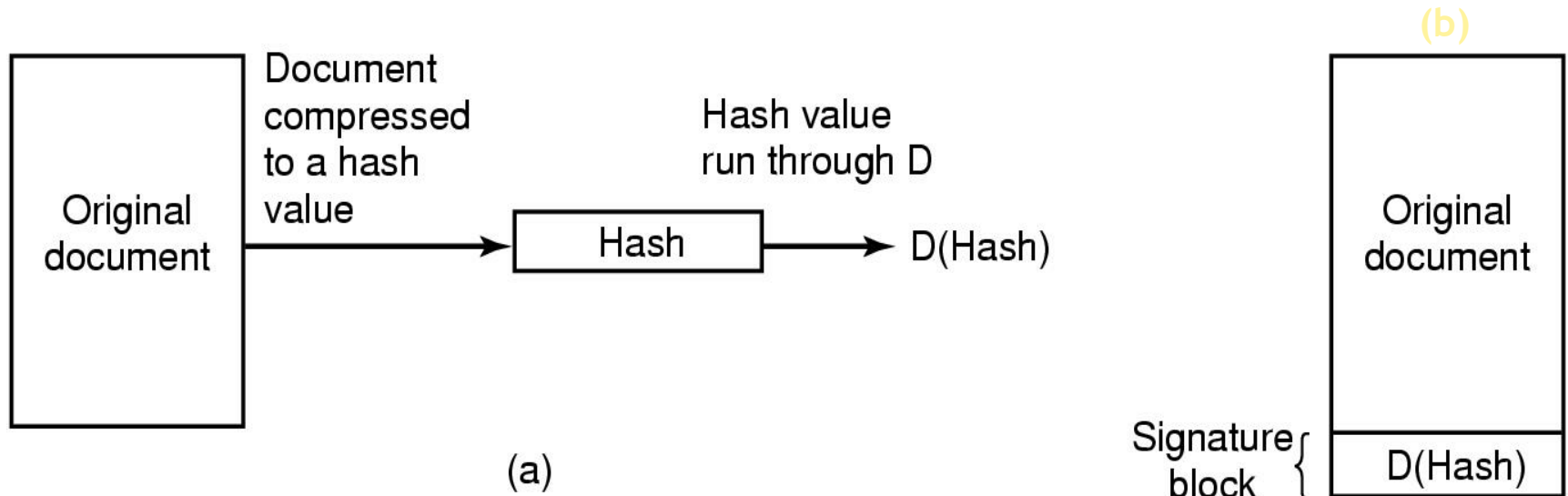


$$M = E_{Pu(Alice)}[D_{Pr(Bob)}(S)]$$



No third party needed!

# DIGITAL SIGNATURES WITH MESSAGE DIGEST (NON-REPUDIATION)



(a)  $D$  is the private key of the sender

(b) The receiver uses the public key of the sender to check the signature

**The trick: Sign only the digest, not the whole message!**

# SOME PROBLEMS WITH PUBLIC-KEY DIGITAL SIGNATURES

- Unless you take Alices's public key from her in person (hard to do for *processes*), finding it is non-trivial
- If Alice discloses her private key (or claims that it was stolen), it can no longer be proven that X had sent the message

For this reason, the **escrow** of private keys used for signatures is forbidden by law in many jurisdictions

- Ditto if Alice decides to change her key
- The scheme is an overkill (it is slow) because it combines *authentication* and *confidentiality*

# CERTIFICATES (PUBLIC KEY DISTRIBUTION)

- To use the public key signature scheme, the sender's public key must be known
- It could be published (on a web site, for example), but then there is a danger of the key being altered
- A common solution is to use certificates:
  - A sender attaches his or her (name, public key) pair, digitally signed by the trusted third party —the Certificate Authority (CA)
  - Once the receiver obtains the public key of CA, the receiver can accept certificates from all senders who use this CA

# A CERTIFICATE

*I, the SuperCert Certification Authority, am delighted to confirm that the public key  $A789FHAFDEG8600FFA$  belongs to Igor Faynberg*

*The SHA-1 digest of the above, signed with the SuperCert private key*

Presumably, your computer has been pre-loaded with the SuperCert public key,  $P$ , so you can always

- Compute the SHA-\* digest  $D$  of the declaration part of the certificate
- Verify that  $D = P(\text{signature})$

There is nothing secret about certificates; they can be sent in the open

# A DIFFERENT USE OF A CERTIFICATE: BINDING AN ATTRIBUTE TO A KEY

*I, the SuperCert Certification Authority, am delighted to confirm that person who owns the public key A789FHAFFDEG8600FFA is older than 21, and so you can legally sell him alcohol in New Jersey.*

The SHA-1 digest of the above, signed with the SuperCert private key

**An important feature: It preserves privacy!**

## QUESTIONS (NOTE SIMILARITY WITH DNS)

- What are all the possible formats (of attributes and all), and who could possibly manage them?
- How can one CA possibly manage all certificates, and which organization is it anyway?
- And suppose everyone trusts this organization, but how could it preserve its single public key from being modified?



# X.509 v3 IN DETAIL (IDENTICAL TO ISO/IEC 9594-8)

- Defines a framework for both public-key and attribute certificates
  - Specification of data objects for certificates and their revocation notices
  - Critical components of the Privilege Management Infrastructure (PMI)
- Defines a framework for the provision of authentication services by Directory
  - Weak password-based authentication
  - Cryptographic-technique credentials authentication
- Specifies three versions of PKI certificates and two versions of attribute certificates
- Provides examples of privilege policies' syntax and attributes

# X.509: A STANDARD FOR CERTIFICATES

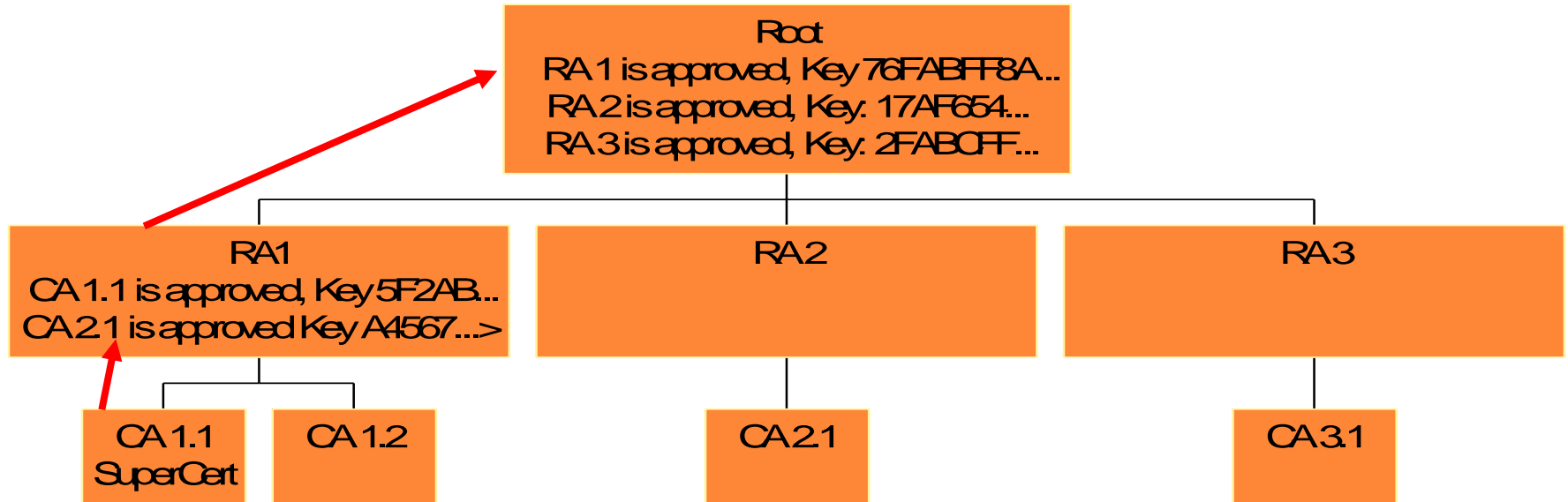
Version	v1 (default), v2, or v3
Serial Number	Integer (unique for the given CA)
Signature	Algorithm Id (deferred to a standardized profile), example: sha-1WithRSAEncryption
Issuer	ID (string) of the issuer
Validity	{not before $t_1$ , not after $t_2$ }
Subject	Name (string) of the issuant
Subjects PK information	{Algorithm Id, Public Key (bit string)}
Issuer Unique Identifier	(for name re-use, <i>only</i> in v2 or v3)
Subject Unique Identifier	(for name re-use, <i>only</i> in v2 or v3)
Extensions	<i>Optional</i> list, <i>only</i> in v2 or v3

Encrypted Hash

After ITU-T Recommendation X.509 (v3: August, 2005)

# PUBLIC KEY INFRASTRUCTURE (PKI)

## *SCHEMATIC DESCRIPTION*



I, the SuperCert Certification Authority, am delighted to confirm the public key A789FHAFDEG8600FFA belongs to Igor Faynberg

The SHA-1 digest of the above, signed with the SuperCert private key

RA: Registration Authority

CA: Certificate Authority

→ : Chain of trust

# WHERE WE ARE WITH THE PKI

- There is no single world-wide authority
- There are many roots with their own trees.  
Modern machines come pre-loaded with over 100 roots known as **trust anchors**
- Certificates can be stored at the user's sites, but it would be more convenient (easier to look them up) to use the Domain Name System and store them at DNS sites
- Certificates are timed, and they can also be revoked (CAs issue Certificate Revocation Lists [CRLs])
- RFC 5280 contains the Internet PKI profile for X.509 v3 certificate and X.509 v2 CRL

## AND THERE ARE ALSO NON-PKI SOLUTIONS (SEE THE TEXTBOOK AND REFERENCES)

- Secure Shell (SSH) uses public-key infrastructure, but does not mandate the use of certificates: the *fingerprint* (the hash of the public key) is to be obtained through another channel

*public key* A789FHAF~~F~~DEG8600FFA

In-band channel

The SHA-2 digest of the above

Out-of-band  
channel

# BUT WITH PKI THERE ARE MORE PROBLEMS

- An assignment: Please read this article:  
<http://arstechnica.com/security/2015/04/google-chrome-will-banish-chinese-certificate-authority-for-breach-of-trust/>



## HERE IS WHAT CAN HAPPEN

- An enterprise IT can insert its trust anchor certificate into Alice's certificate directory
- It can then insert a *Man-In-The-Middle* (MITM) server (a reverse proxy) to monitor Alice's actions at a site *Bob* and sign the MITM server's public key with a certificate to Bob

