CS 524 A

Introduction to Cloud Computing

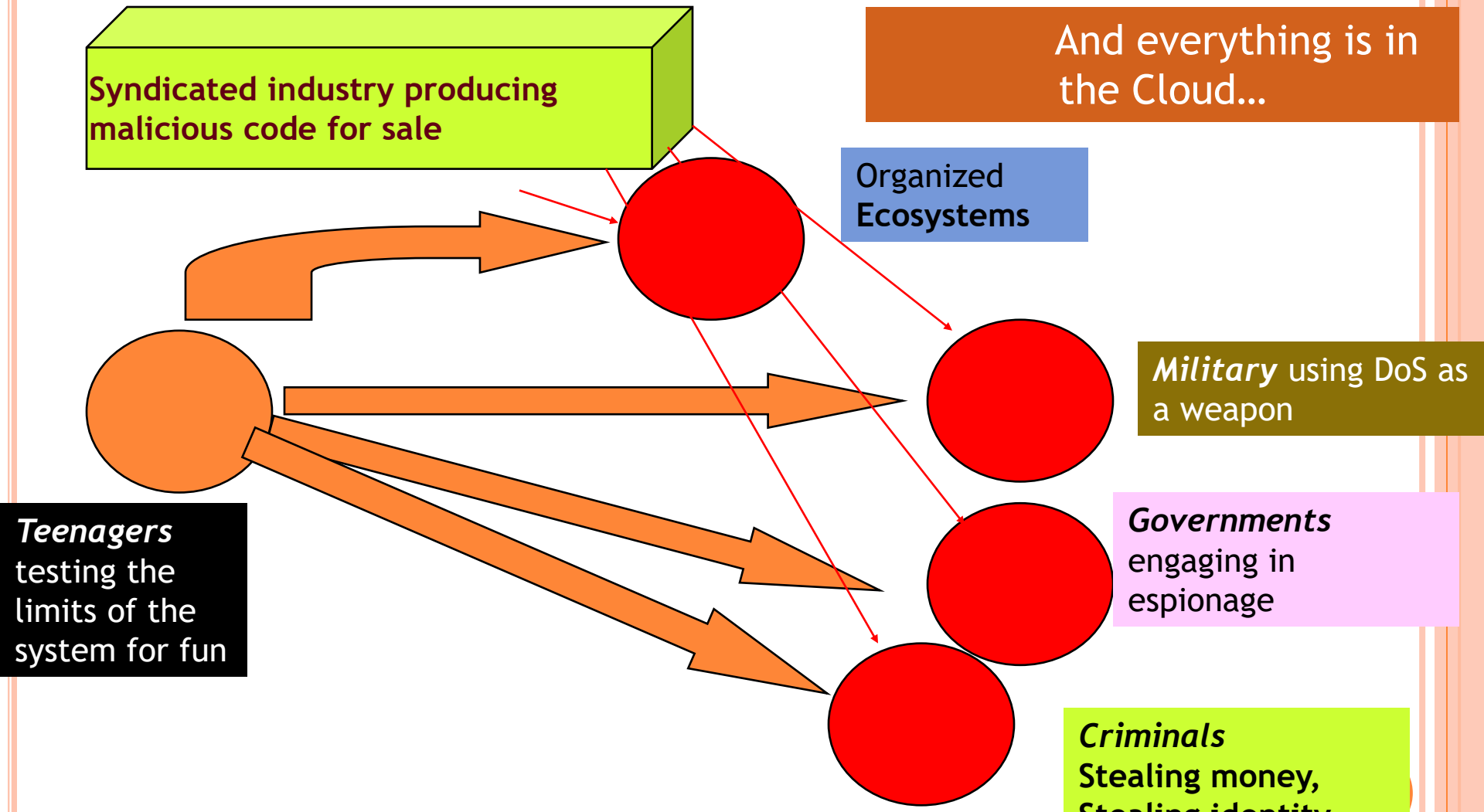
Lecture 12:
Security and Identity Management in the Cloud
Part I

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…

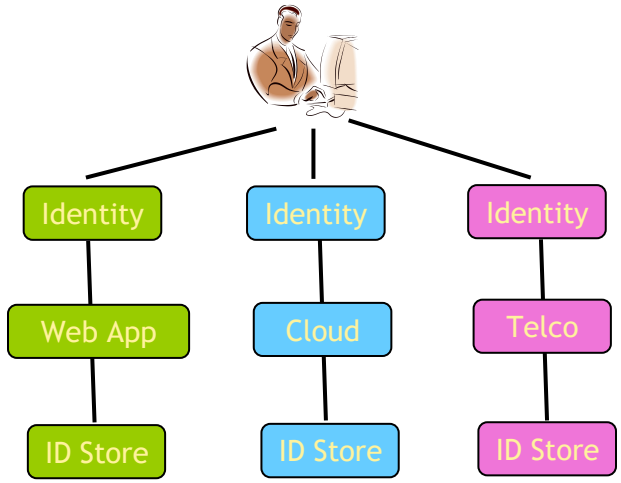
- Can you send a message that is truly private?
- Do you know who really sent you a message?
  - Do you know what will happen when you open an attachment?
  - Do you know what will happen when you click on a link?
- Can you be sure that the message you know was sent to you by a friend was not *modified* 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 7th Framework Programme conference established by the European Commission

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

NGN IdM Framework (ITU-T Rec. Y.2720, 1/2009)

Business and Security Services

Federated Services

Application Access Control (e.g., Multimedia and IPTV)

Single Sign-on and Sign-off

Role -based Access to Resources

Protection of Personally-Identifiable Information

Security Protection of Information and Network Infrastructure

IdM Capabilities

Identity Lifecycle Management

Correlation and Binding of Identity Information

Authentication, Assurance, and Assertion of Identity Information

Discovery and Exchange of Identity Information

Identity Information

Identifiers
(e.g., User ID, email address, telephone number, URI, IP address)

Credentials
(e.g., digital certificates, tokens, and biometrics)

Attributes
(e.g., roles, claims, context, privileges, location)

Entities

Organizations, Business Enterprises, Government Enterprises

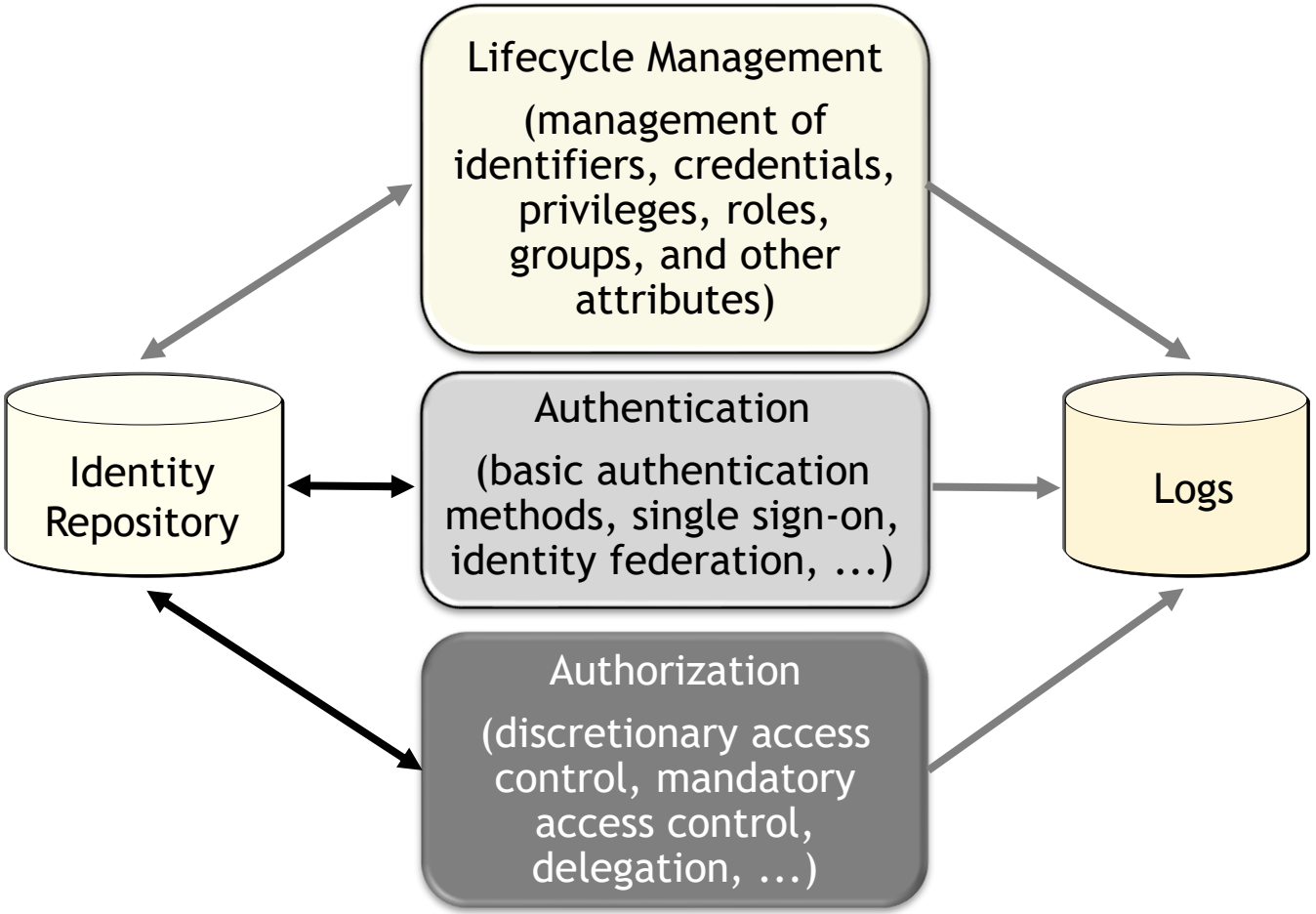
Cloud Providers

Virtual Objects

Users & Subscribers

User Devices

Network Elements and Objects



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 user may have access to a resource
- 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

Alice

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

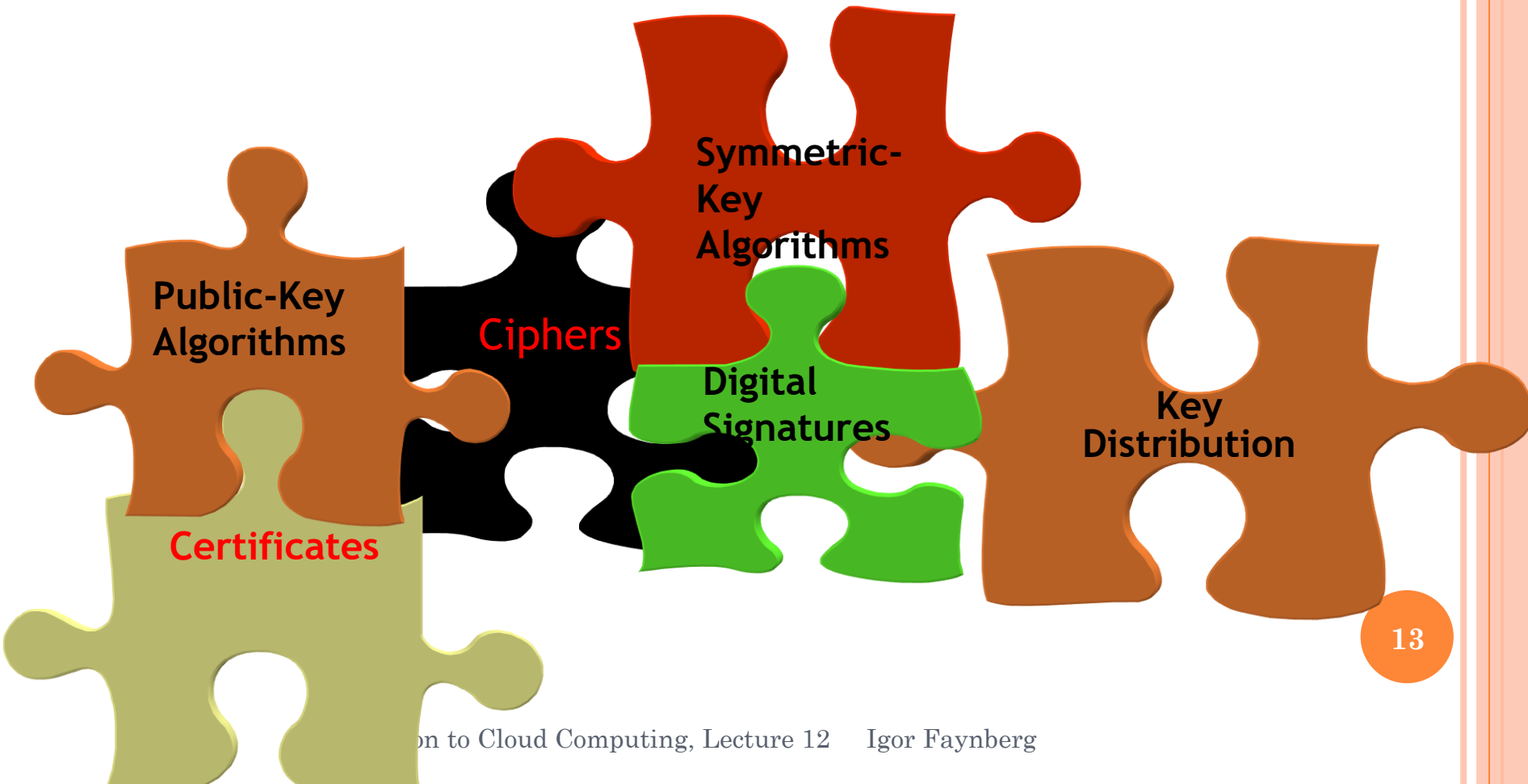
3. What Alice is
- Fingerprint
- Iris
- Voiceprint

May be stolen or cloned

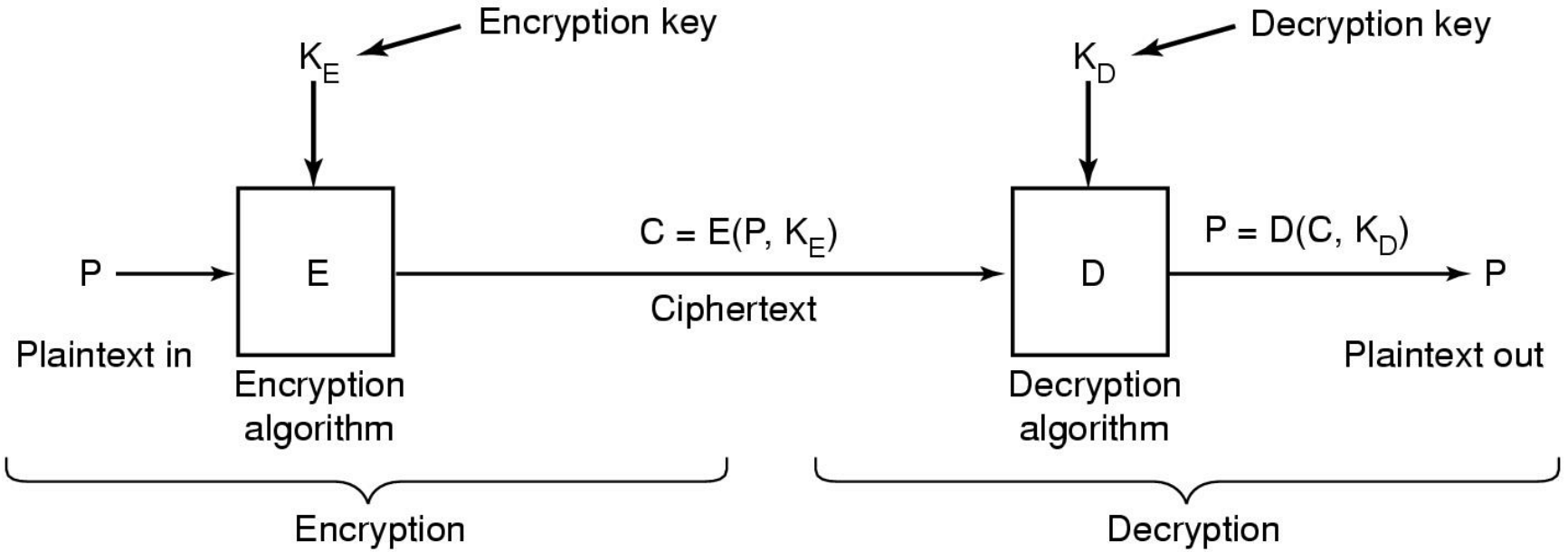
May be cloned

Ensuring Confidentiality, Integrity, and Non-Repudiation: Cryptography

κρνπτωσ (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***

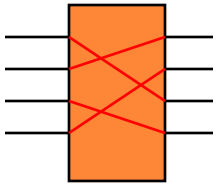
$$K_E = K_D$$

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

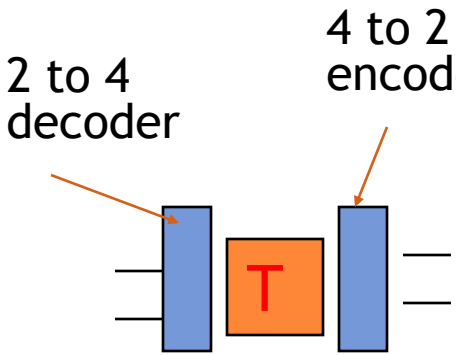
Plaintext: 0011100110100101 10
Key: 1001001001111101 10
Cyphertext: 1 0101011110110000 0

Modern Symmetric-Key ($K_D = K_E$) Algorithms

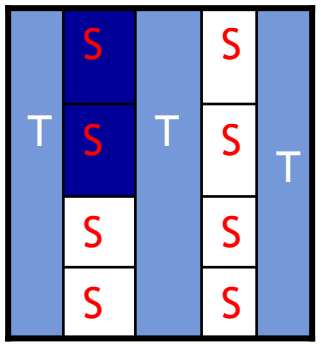
- Combine transpositions and substitutions and cascade them to make the algorithms very complex (to prevent cryptanalysis even when large amounts of cyphertext are available)
- Often use *block ciphers*



4-bit transposition (T)



2-bit substitution (S)



Cascading into a product

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: $ed \equiv 1 \pmod{z}$
- 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 $D(C) \equiv C^d \pmod{n} \equiv P^{ed} \pmod{n} \equiv P^{k\phi(n)+1} \pmod{n} \equiv P^{k\phi(n)} P \pmod{n} \equiv P \pmod{n}$

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

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 difficulty 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 shorter keys)

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 Koblits, 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-base 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* resultling 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 literally *digesting* a big plaintext file *P*

- Break input into chunks of the desired hash size *s*: $P = p_1 \mid p_2 \mid p_3 \ \ldots \mid p_n$
- 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, … 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

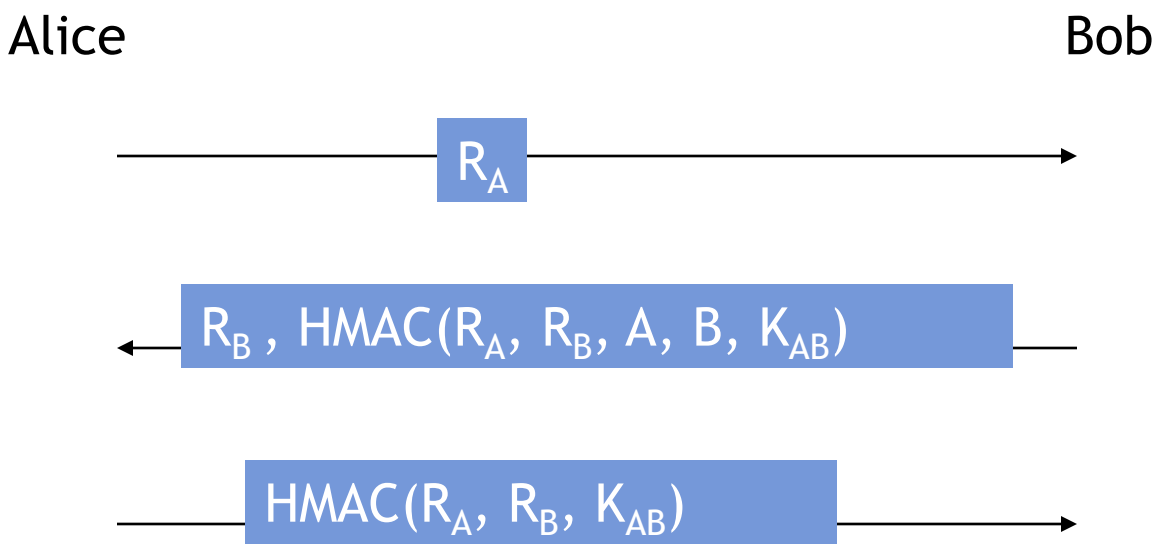
- Take the key, K, and compute $m_1 = h(K)$
- Then continue for i=2, … n=[|P|/|K|] + 1:

$$m_i = h(m_{i-1})$$

- Concatenate these chunks into a one-time pad key $M = m_1 | m_2 | m_3 \ldots | m_n$
- $E = P \oplus M$

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)

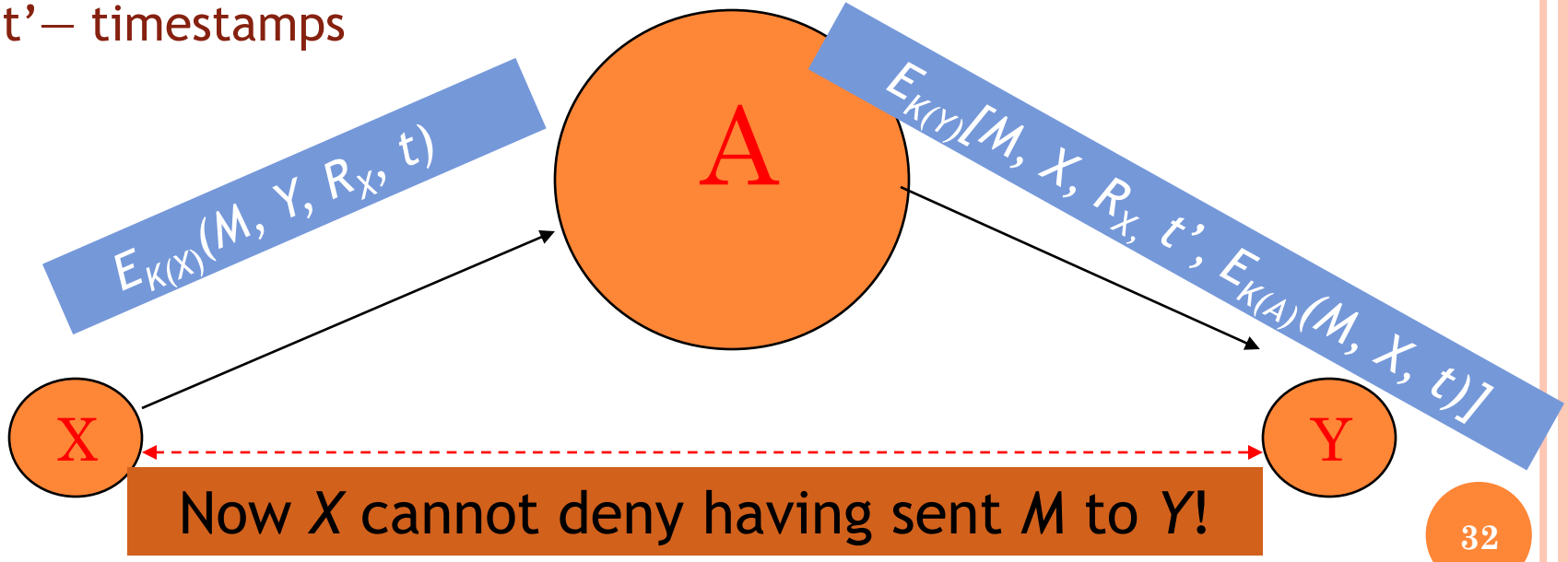
- If a process needs to talk to n other processes, it will need to share n keys. As n grows, key management becomes a burden…
- Another approach: Each user has a key shared with KDC, and all authentication and session key management go through KDC

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

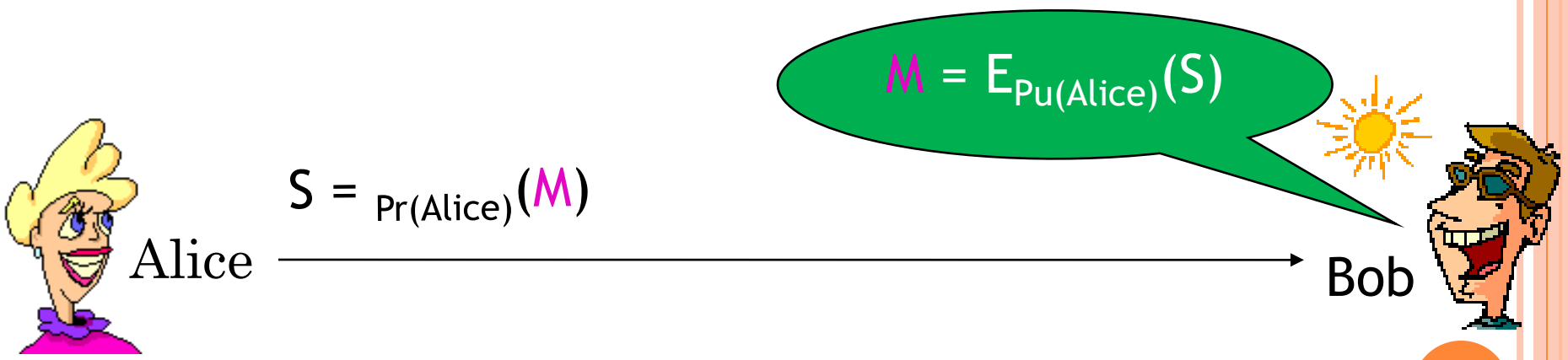
- K(Y)—Y's key with A
- K(A)—the key only A knows
- X—the sender's identifier



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(Alice)—Alice's public key
- Pr(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 $E[D(P)] = P$

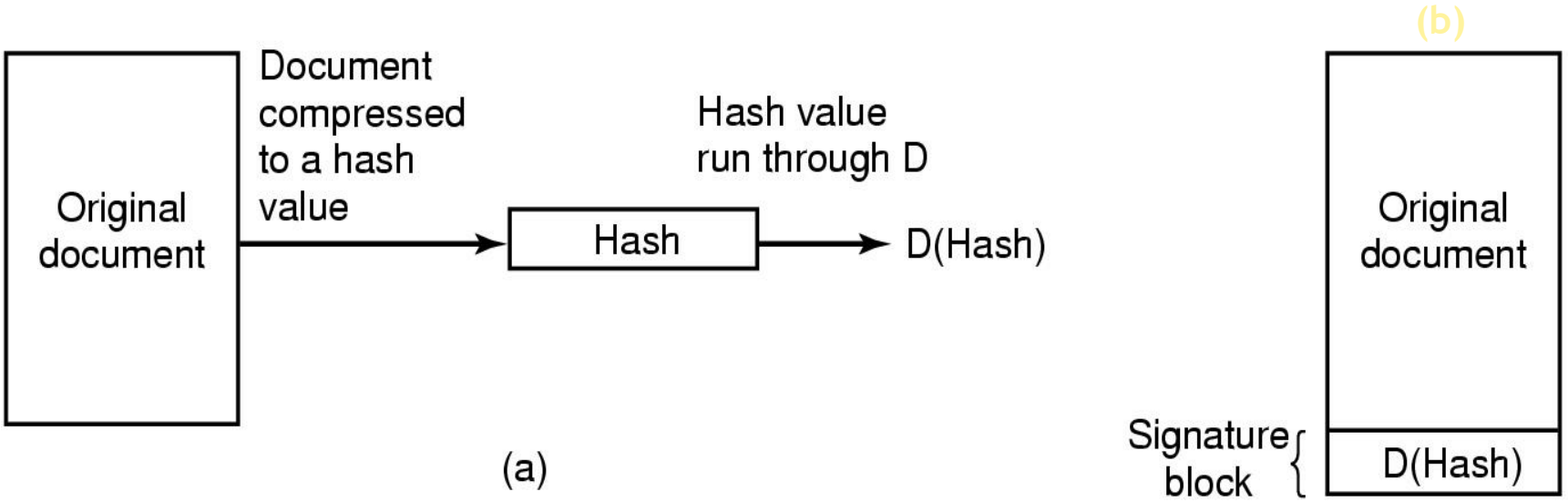
- Pu(Alice)—Alice's public key
- Pr(Alice)—Alice's private key
- Pu(Bob)—Bob's public key
- Pr(Alice)—Alice's public key

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

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

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 A789FHAFFDEG8600FFA 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(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

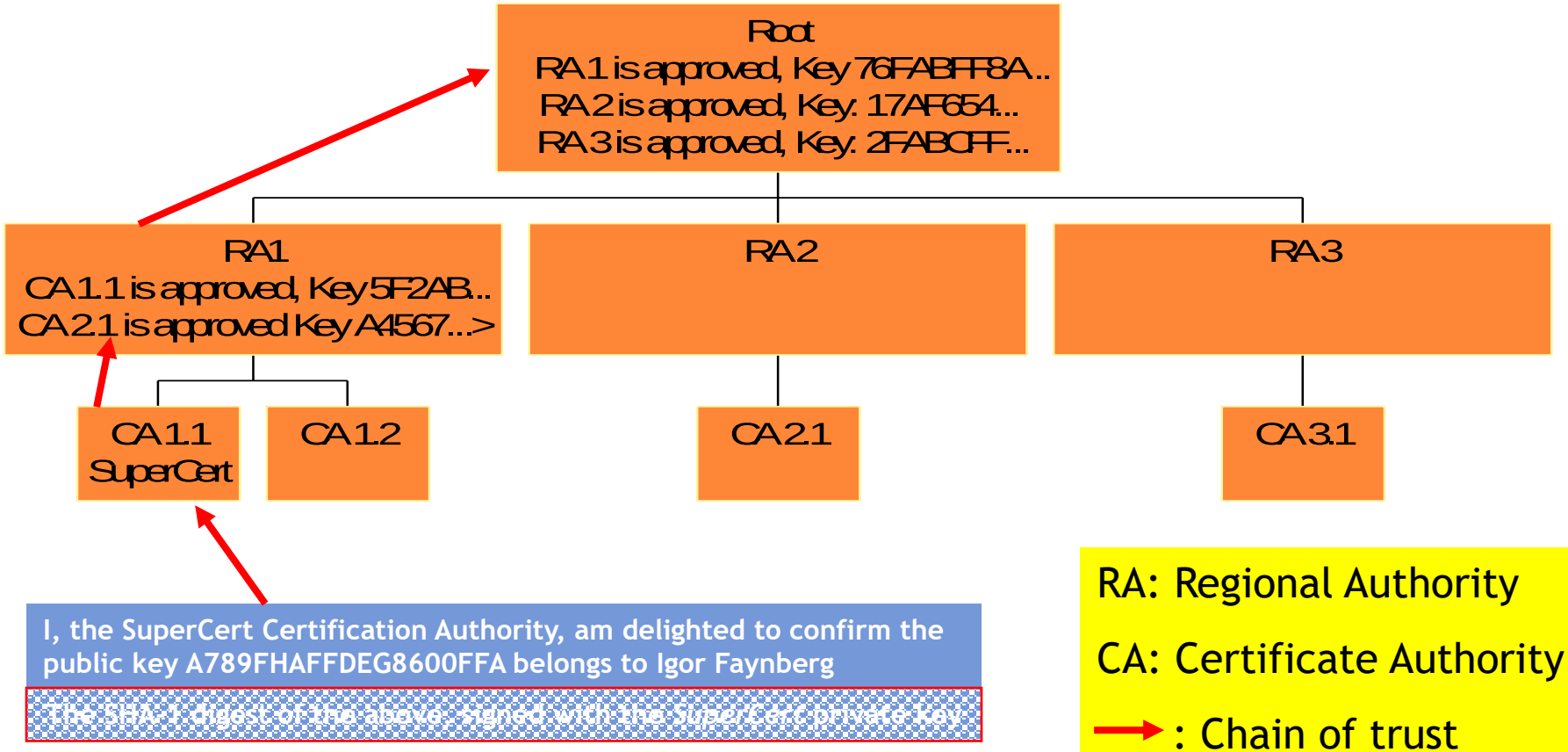
X.509: A STANDARD FOR CERTIFICATES

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, ***only*** in v2 or v3)
Subject Unique Identifier	(for name re-use, ***only*** in v2 or v3)
Extensions	*Optional* list, ***only*** in v2 or v3

Encrypted Hash

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

Public Key Infrastructure (PKI)
Schematic description



RA: Regional 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 A789FHAFFDEG8600FFA — 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 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