CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

MSc - Cybersecurity

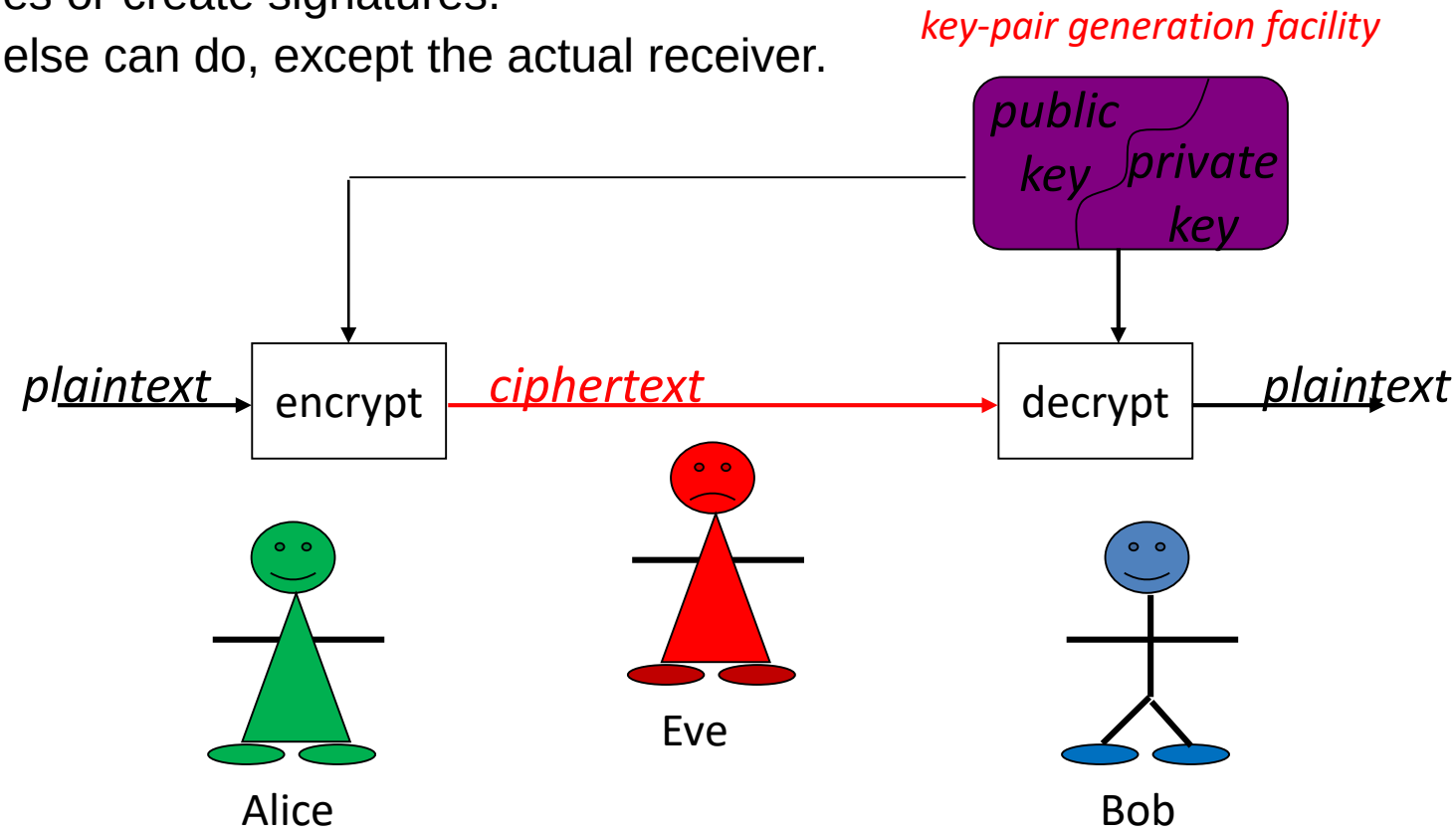
CMT310: Developing Secure Systems and Applications

Public Key Cryptography: Introduction, DHKE, RSA

Dr Neetesh Saxena
saxenan4@cardiff.ac.uk

Public Key Cryptography

- Uses two keys:
  - A **public key** - **encrypt messages** and **verify signatures**.
  - A **private key** - **decrypt messages** and **sign** (create) **signatures.**
- **Asymmetric** since parties are **not** equal.
  - Those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures.
  - No one else can do, except the actual receiver.



Why Public Key Cryptography?

- Developed to address two key issues:
  - **Key distribution** – how to have secure communications in general without having to trust a KDC with the key.
  - **Digital signatures** – how to verify a message comes intact from the claimed sender.

- Security of an asymmetric cryptosystem relates to the difficulty of solving a “difficult” problem:
  - factorise a composite number (e.g., n = p*q)
  - calculate the discrete log of a number (e.g., y = x^i)

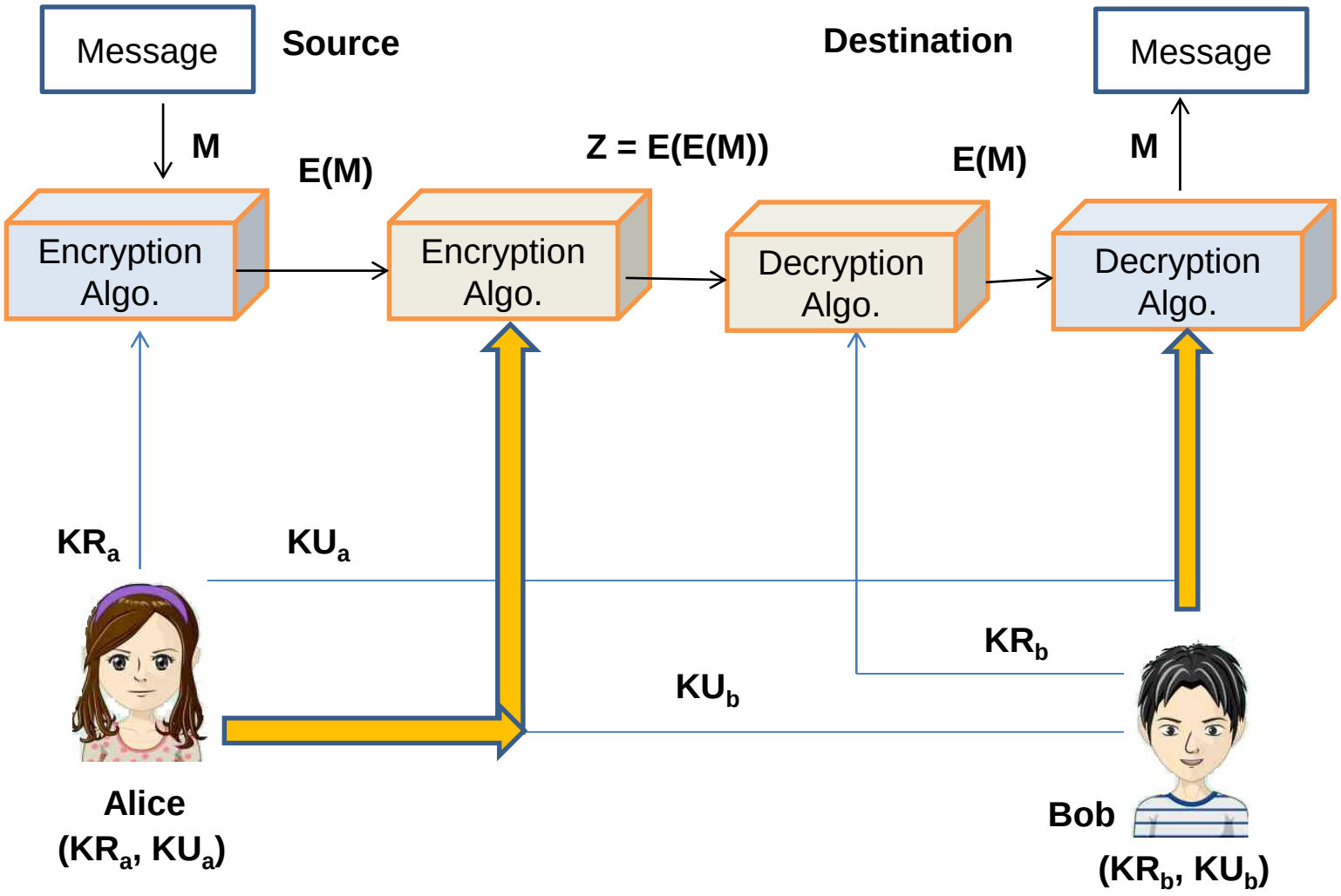
Public Key Applications

- Can classify uses into 3 categories:
  - **Encryption/decryption** (provide secrecy)
  - **Digital signatures** (provide identity authentication)
  - **Key exchange** (of session keys)

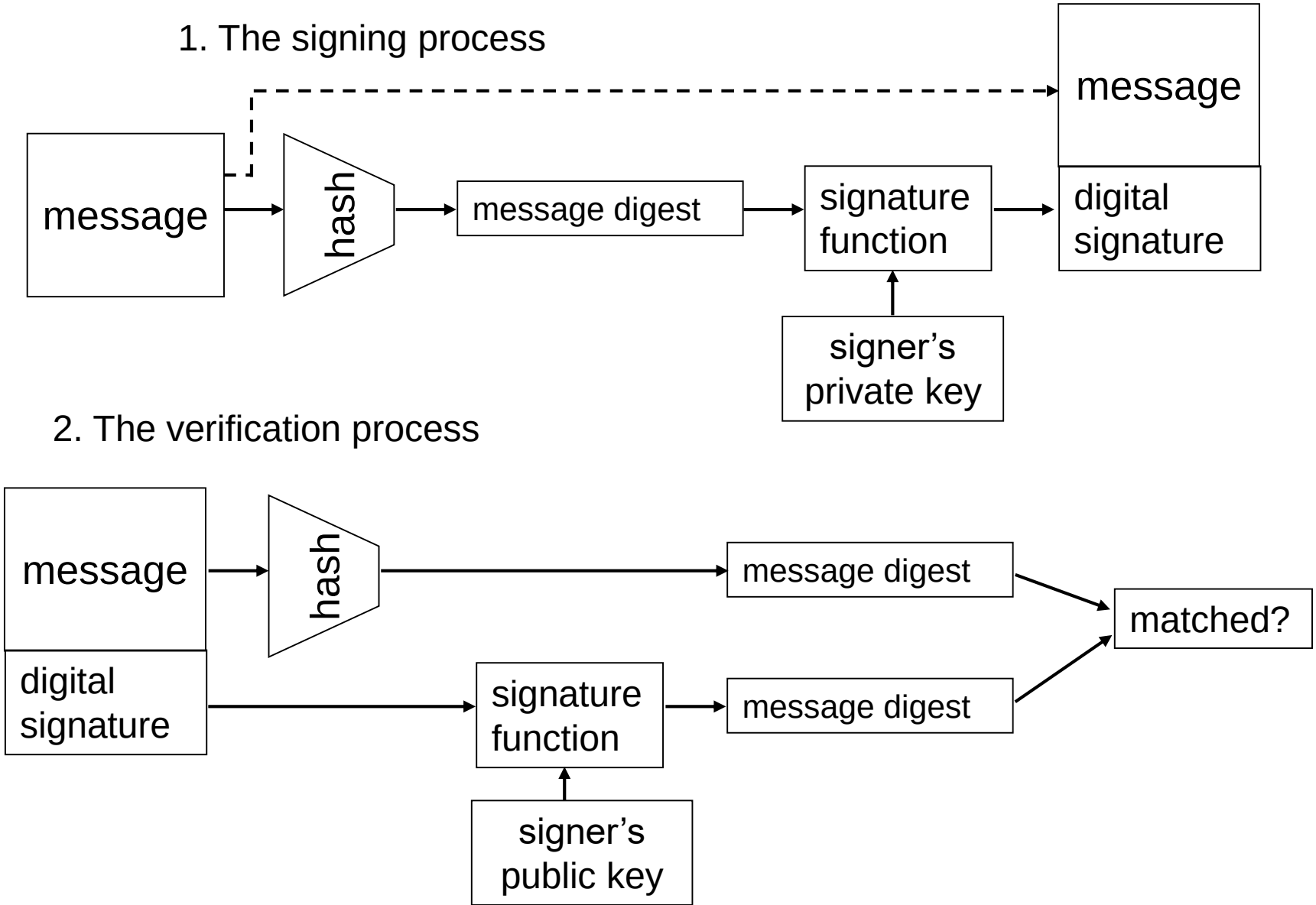
Public-Key Algorithms - Characteristics

- Public-Key algorithms rely on 2 keys with characteristics:
  - **Computationally infeasible** to find *decryption key* knowing only algorithm and encryption key.
  - **Computationally easy** to *en/decrypt* messages when the relevant (en/decrypt) key is known.
  - Either of the two related keys can be used for encryption, with the other used for decryption.
  - **Breaking cost** – much more than decrypting the message.
  - **Breaking time** – higher than meaningful information time.

Authentication and Confidentiality



Digital Signatures: Message Authentication



Facts About Numbers

- Prime number $\boldsymbol{p}$:
  - $\boldsymbol{p}$ is an integer
  - $\boldsymbol{p} \geq 2$
  - The only divisors of $\boldsymbol{p}$ are 1 and $\boldsymbol{p}$

- Examples
  - 2, 7, 19 are primes
  - **-3**, 0, 1, 6 are not primes

- Prime decomposition of a positive integer $\boldsymbol{n}$:

  $\boldsymbol{n} = \boldsymbol{p}_1^{\boldsymbol{e}}1 \times \ldots \times \boldsymbol{p}_{\boldsymbol{k}}^{\boldsymbol{e}}\boldsymbol{k}$

- Example: $200 = 2^3 \times 5^2$

Cont.

- Given a number compute the co-prime of that number.
- Two numbers are coprime if their highest common factor (or greatest common divisor if you must) is 1.
- Hint:
  - $\varphi(n) = \varphi(pxq) = (p-1)x(q-1)$

Primitive Root

- In modular arithmetic,
  - a number *m* is a **primitive root modulo *n*** if every number *a* coprime to *n* is congruent to a power of *m* modulo *n*.

- i.e.,
  - for every integer *a* coprime to *n*, there is an integer *k* such that $m^k \equiv a \pmod{n}$.

Such *k* is called the **index** or **discrete logarithm** of *a* to the base *m* modulo *n*.

Example

- For example, if $n = 14$ then the elements of $\mathbf{Z}_n^\times$ are the congruence classes {1, 3, 5, 9, 11, 13}; there are $\varphi(14) = 6$ of them.

x	$x, x^2, x^3, \ldots \pmod{14}$
1 :	1
3 :	3, 9, 13, 11, 5, 1
5 :	5, 11, 13, 9, 3, 1
9 :	9, 11, 1
11 :	11, 9, 1
13 :	13, 1

- [Fermat Theorem: for any integer m coprime to n, we have $m^{\varphi(n)} \equiv 1 \pmod{n}$, and $1 \leq m < n$]

- The order of 1 is 1, the orders of 3 and 5 are 6, the orders of 9 and 11 are 3, and the order of 13 is 2. Thus, 3 and 5 are the primitive roots modulo 14.

Greatest Common Divisor

- The greatest common divisor (GCD) of two positive integers ***a*** and ***b***, denoted gcd(***a***, ***b***), is the largest positive integer that divides both ***a*** and ***b***

- Examples:

  gcd(18, 30) = 6 gcd(0, 20) = 20
  gcd(-21, 49) = 7

- Two integers a and b are said to be relatively prime if

  gcd(***a***, ***b***) = 1

- Example:
  - Integers 15 and 28 are relatively prime

Modular Arithmetic

- Modulo operator for a positive integer n

  $$r = a \bmod n$$

  equivalent to

  $$a = r + kn$$

  and

  $$r = a - \lfloor a/n \rfloor n$$

- Example:

  29 mod 13 = 3 13 mod 13 = 0 -1 mod 13 = 12

  $29 = 3 + 2\times13$ $13 = 0 + 1\times13$ $12 = -1 + 1\times13$

  For a<0, we first add a large kn to a such that it becomes positive

- Modulo and GCD:

  $$\gcd(a, b) = \gcd(b, a \bmod b)$$

- Example:

  gcd(21, 12) = 3 gcd(12, 21 mod 12) = gcd(12, 9) = 3

Euclid's GCD Algorithm

- Euclid's algorithm for computing the GCD repeatedly applies the formula

  gcd(***a***, ***b***) = gcd(***b***, ***a*** mod ***b***)
- Example
  - gcd(412, 260) = 4

```
Algorithm EuclidGCD(a, b)
  Input integers a and b
  Output gcd(a, b)

  if b = 0
    return a
  else
    return EuclidGCD(b, a mod b)
```

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Diffie-Hellman Key Exchange

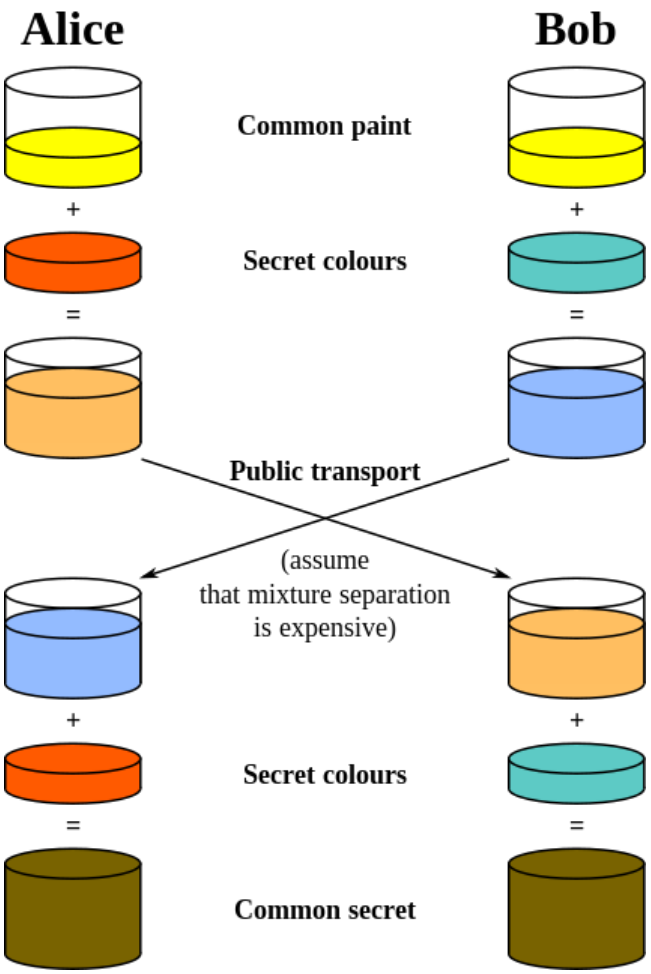
- History: Developed by Whitfield Diffie, Martin Hellman
  - Published in 1976 paper “New Directions in Cryptography”
- Allows negotiation of secret key over insecure network
- Algorithm
  - Public parameters: protocol uses the multiplicative group of integers modulo *p*, where *p* is prime, and *g* is a primitive root modulo *p*.

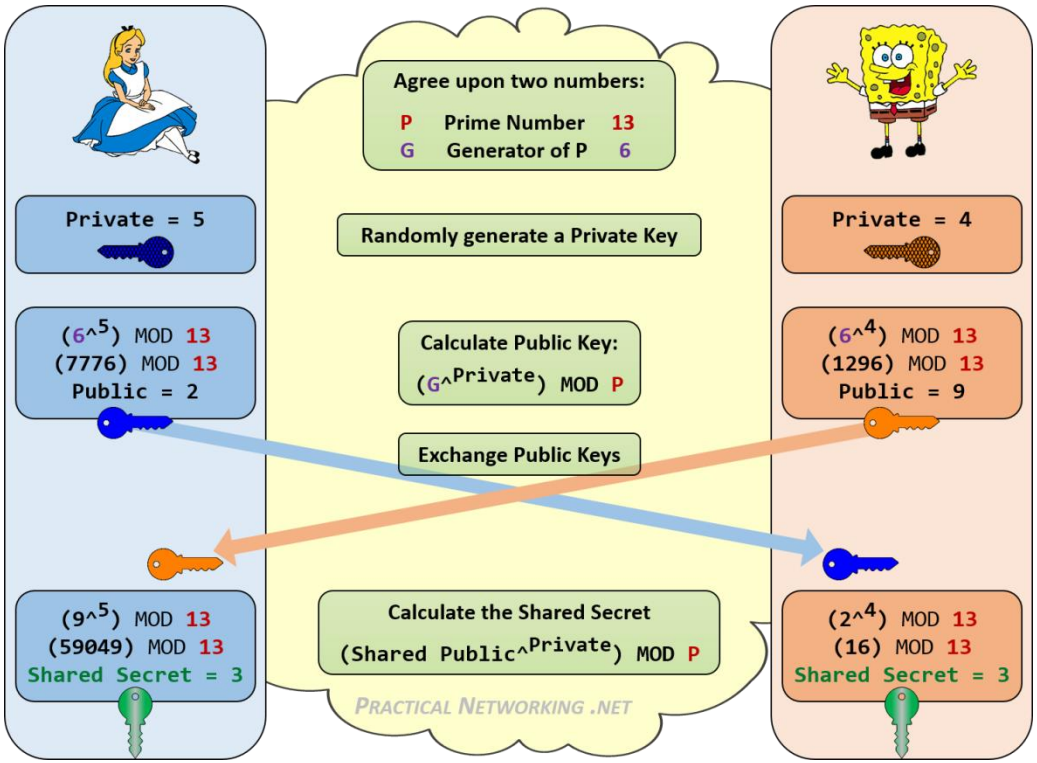
1. (publicly) base *g* and modulus *n*

Alice	Bob
2. Randomly select a large integer *x* and send $A = g^x \bmod n$.	2. Randomly select a large integer *y* and send $B = g^y \bmod n$.
3. Compute the key $K = B^x \bmod n$.	3. Compute the key $K = A^y \bmod n$.

$$K = B^x = (g^y)^x = (g^x)^y = A^y = K$$

Cont.





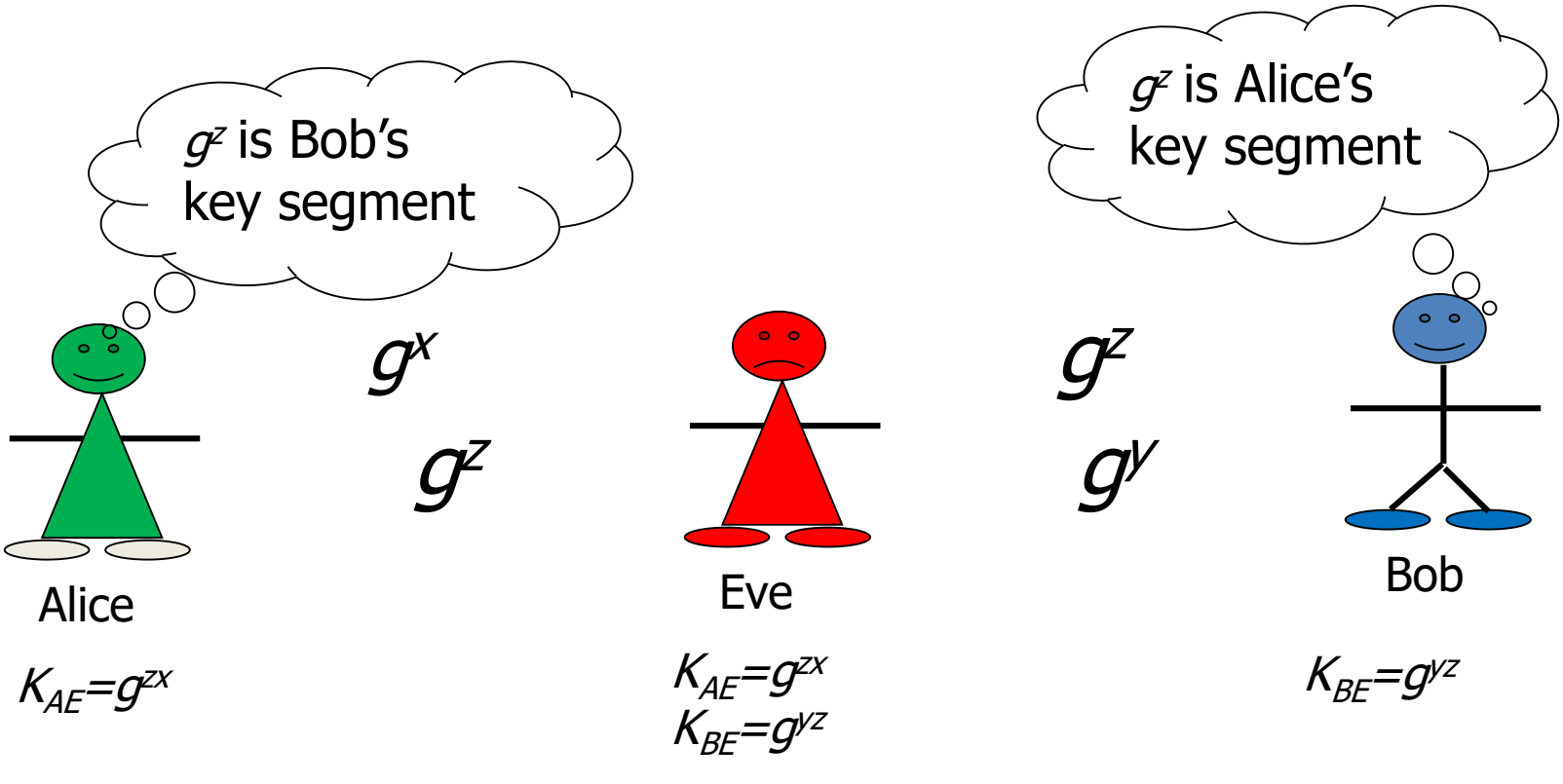
Diffie-Hellman – Security

- Which security properties do we get from Diffie-Hellman key agreement?
- It is a key agreement protocol. ✓
- Secrecy: An attacker does not learn the key.
- No authentication: The parties do not know whom they are establishing a key with.

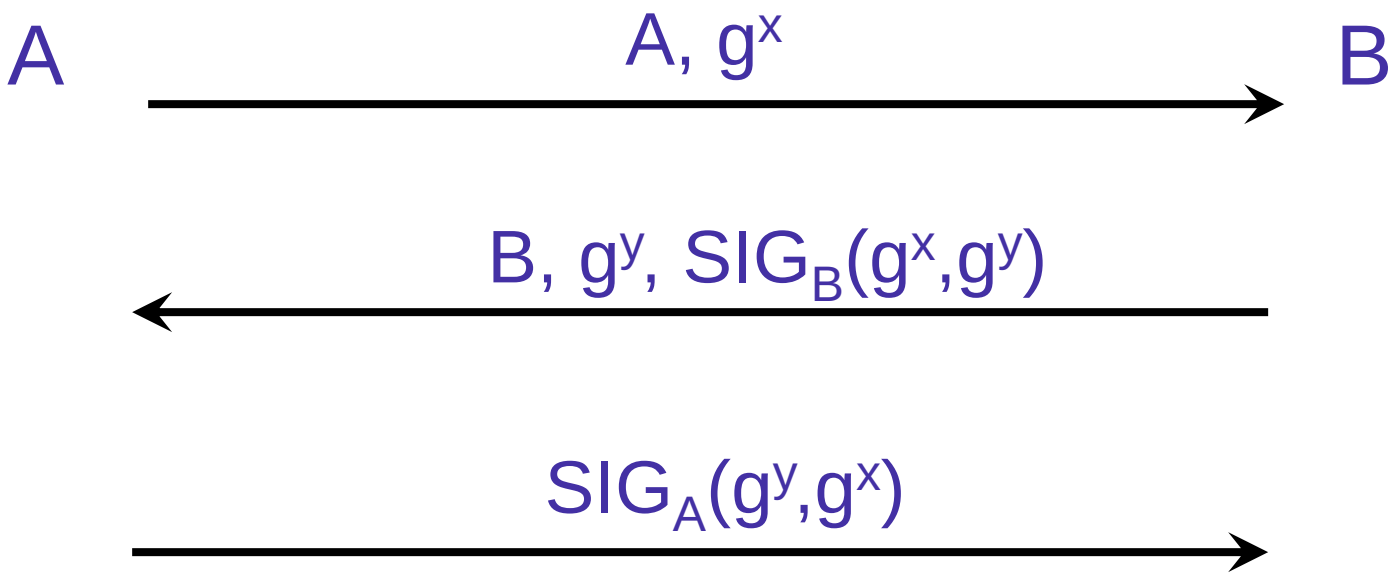
"On the Internet, nobody knows you're a dog."

Diffie-Hellman Weakness

- Man-in-the-Middle attack
  - Assume Eve can intercept and modify packets
- Defense requires mutual authentication
  - Back to key distribution problem



Basic Authenticated DH (BADH)

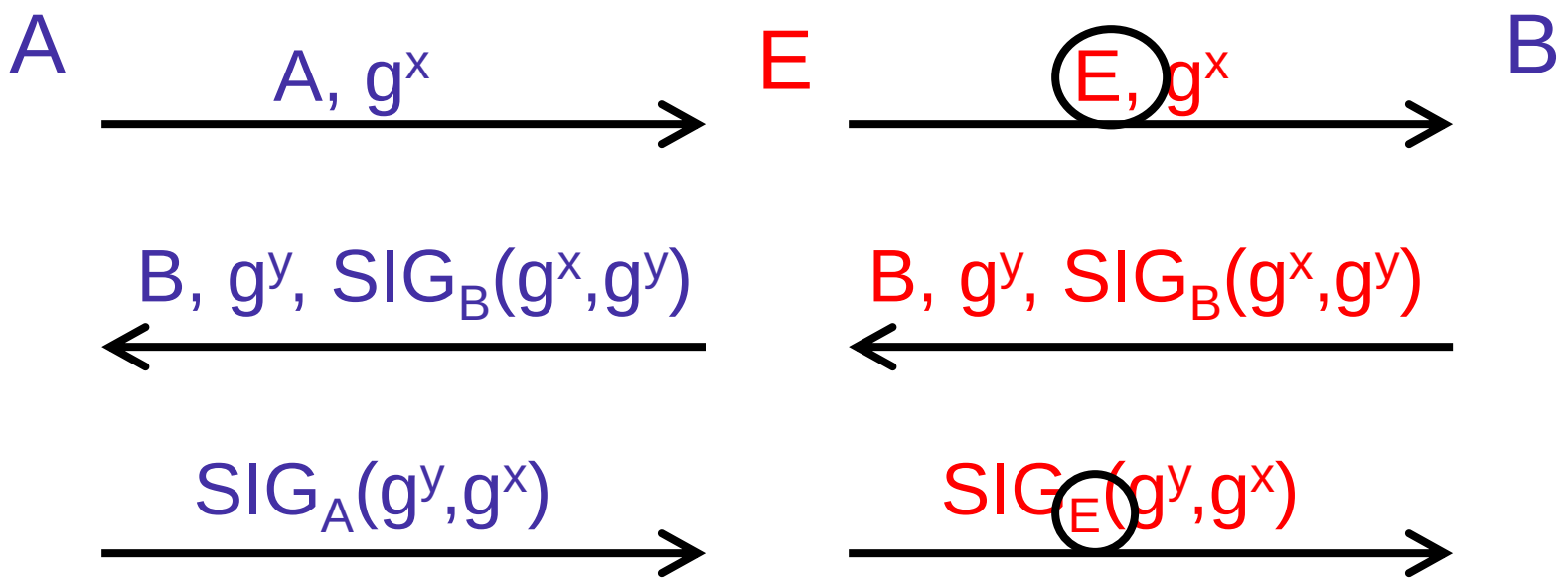


Each party signs its own DH value to prevent MitM attack (and the peer's DH value as a freshness guarantee against replay)

A: "Shared K=g^{xy} with B" (K⇔B) B: "Shared K=g^{xy} with A" (K⇔A)

Looks fine, but… **(there must be a reason we call it BADH)**

Identity-Misbinding Attack

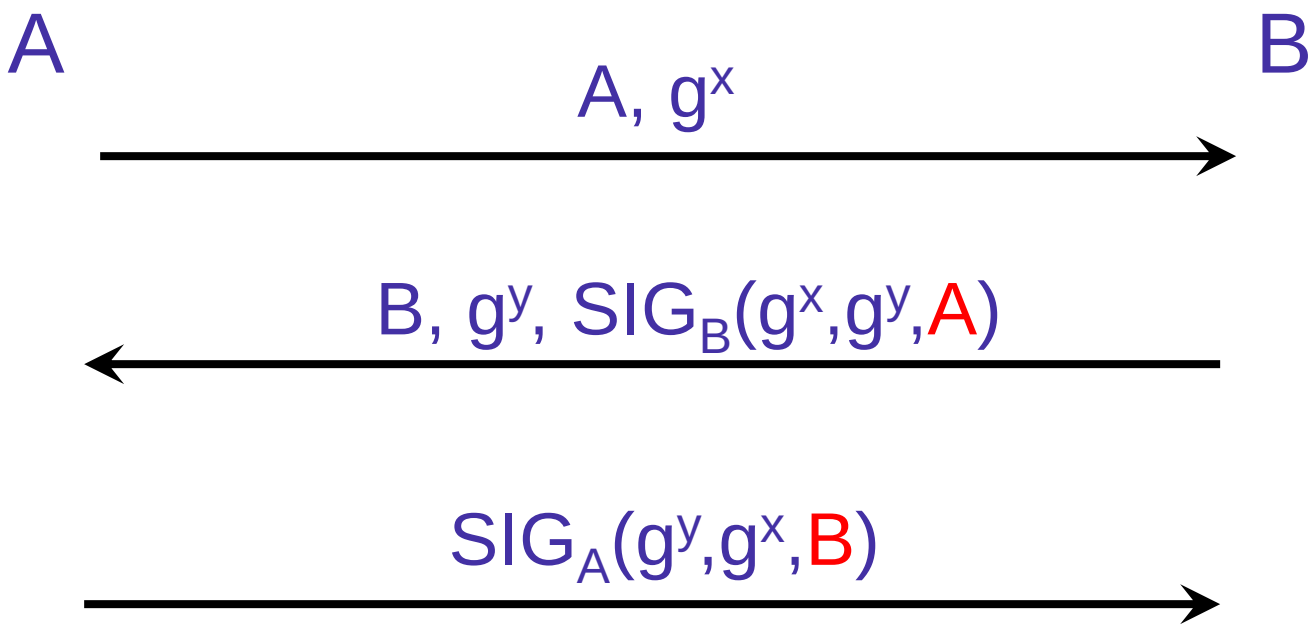


– Any damage? Wrong identity binding!

A: "Shared $K=g^{xy}$ with B" ($K \Leftrightarrow B$) B: "Shared $K=g^{xy}$ with E" ($K \Leftrightarrow E$)

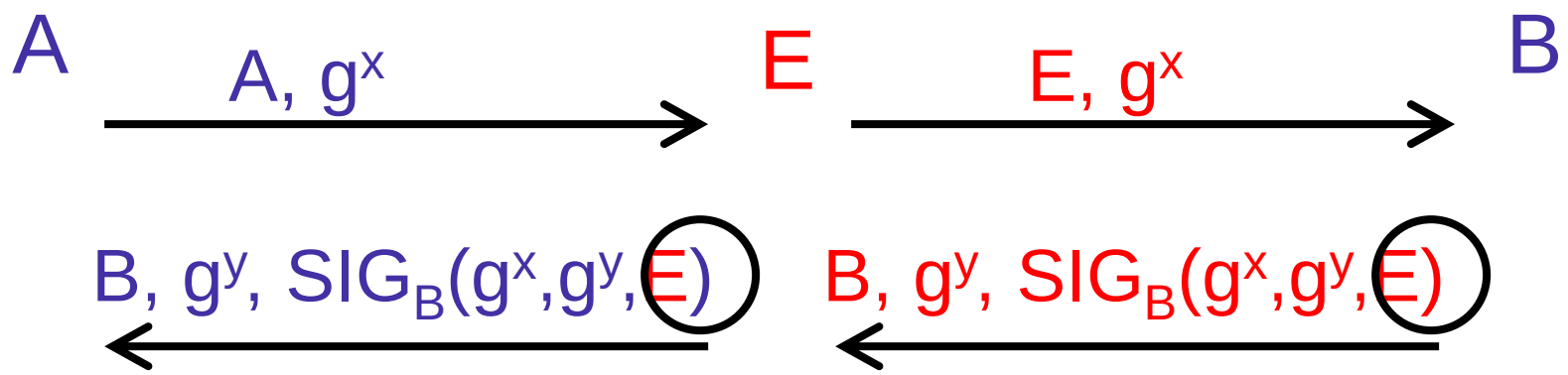
E doesn't know $K=g^{xy}$ but B considers anything sent by A as coming from E

A Possible Solution (ISO-9796)



Thwarts the identity-misbinding attack by including the identity of the peer under the signature

The ISO Defense



A: aha! B is talking to E not to me!

Note that E cannot produce $SIG_B(g^x,g^y,A)$

DHKE Key Issue with Internet Browsers

- **Google Chrome**, Opera and Vivaldi throw a "this site can't provide a secure connection" error with no override option. Other Chrome or Chromium-based browsers are likely throwing the same error message.
- **Pale Moon** throws a "secure connection failed" error.
- **Microsoft Edge** displays "hmm, we can't reach this page" error instead.
- **Internet Explorer** throws the error "this page can't be displayed.

Look for detail here: https://weakdh.org/
Full Paper: https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

Firefox blocks weak Diffie-Hellman keys

by Martin Brinkmann on October 03, 2016 in Firefox - 6 comments

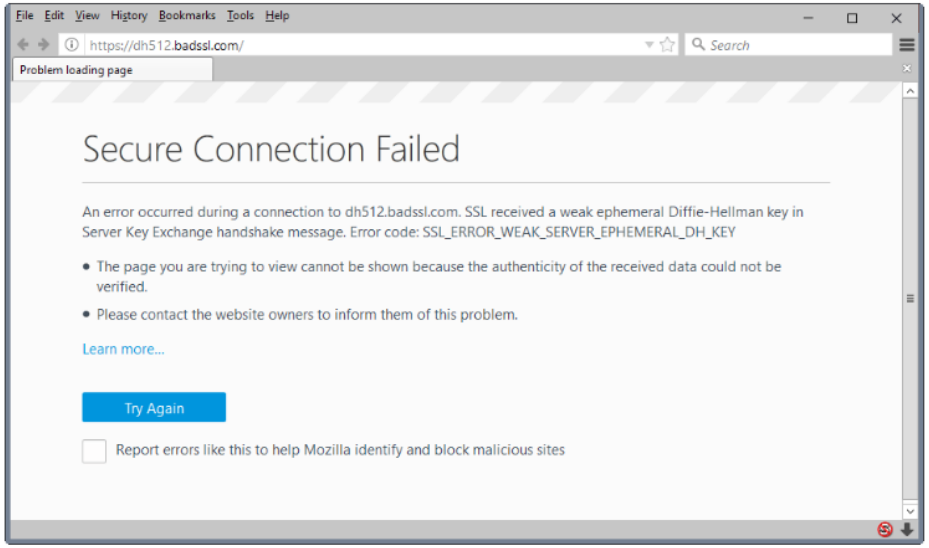
Mozilla announced on September 30, 2016 that it made the decision to enforce stronger Diffie-Hellman keys in the Firefox web browser.

Firefox users who visit websites that use weak -- now less than1023 bits -- will see a connection error message in the web browser instead of the actual site.

The message reads "secure connection failed" and the reason given is the following one:

> SSL received a weak ephemeral Diffie-Hellman key in Server Key Exchange handshake message. Error code: SSL_ERROR_WEAK_SERVER_EPHEMERAL_DH_KEY

The page lists a learn more link that leads to the Firefox "what does your connection is not secure mean" support page on Mozilla Support.



https://www.ghacks.net/2016/10/03/firefox-blocks-weak-diffie-hellman-keys/

DH Fails?

Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

David Adrian[¶] Karthikeyan Bhargavan[*] Zakir Durumeric[¶] Pierrick Gaudry[†] Matthew Green[§]
J. Alex Halderman[¶] Nadia Heninger[‡] Drew Springall[¶] Emmanuel Thomé[†] Luke Valenta[‡]
Benjamin VanderSloot[¶] Eric Wustrow[¶] Santiago Zanella-Béguelin[‖] Paul Zimmermann[†]

[*] INRIA Paris-Rocquencourt [†] INRIA Nancy-Grand Est, CNRS, and Université de Lorraine
[‖] Microsoft Research [‡] University of Pennsylvania [§] Johns Hopkins [¶] University of Michigan

For additional materials and contact information, visit WeakDH.org.

ABSTRACT

We investigate the security of Diffie-Hellman key exchange as used in popular Internet protocols and find it to be less secure than widely believed. First, we present Logjam, a novel flaw in TLS that lets a man-in-the-middle downgrade connections to "export-grade" Diffie-Hellman. To carry out this attack, we implement the number field sieve discrete log algorithm. After a week-long precomputation for a specified 512-bit group, we can compute arbitrary discrete logs in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, allowing us to compromise connections to 7% of Alexa Top Million HTTPS sites. In response, major browsers are being changed to reject short groups.

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. We estimate that even in the 1024-bit case, the computations are plausible given nation-state resources. A small number of fixed or standardized groups are used by millions of servers; performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers. A close reading of published NSA leaks shows that the agency's attacks on VPNs are consistent with having achieved such a break. We conclude that moving to stronger key exchange methods should be a priority for the Internet community.

coded, or widely shared Diffie-Hellman parameters has the effect of dramatically reducing the cost of large-scale attacks, bringing some within range of feasibility today.

The current best technique for attacking Diffie-Hellman relies on compromising one of the private exponents (a, b) by computing the discrete log of the corresponding public value $(g^a \bmod p, g^b \bmod p)$. With state-of-the-art number field sieve algorithms, computing a single discrete log is more difficult than factoring an RSA modulus of the same size. However, an adversary who performs a large precomputation for a prime p can then quickly calculate arbitrary discrete logs in that group, amortizing the cost over all targets that share this parameter. Although this fact is well known among mathematical cryptographers, it seems to have been lost among practitioners deploying cryptosystems. We exploit it to obtain the following results:

Active attacks on export ciphers in TLS. We introduce Logjam, a new attack on TLS by which a man-in-the-middle attacker can downgrade a connection to export-grade cryptography. This attack is reminiscent of the FREAK attack [7] but applies to the ephemeral Diffie-Hellman ciphersuites and is a TLS protocol flaw rather than an implementation vulnerability. We present measurements that show that this attack applies to 8.4% of Alexa Top Million HTTPS sites and 3.4% of all HTTPS servers that have browser-trusted certificates.

https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

Cont.

Source	Popularity	Prime
Apache	82%	9fdb8b8a004544f0045f1737d0ba2e0b 274cdf1a9f588218fb435316a16e3741 71fd19d8d8f37c39bf863fd60e3e3006 80a3030c6e4c3757d08f70e6aa871033
mod_ssl	10%	d4bcd52406f69b35994b88de5db89682 c8157f62d8f33633ee5772f11f05ab22 d6b5145b9f241e5acc31ff090a4bc711 48976f76795094e71e7903529f5a824b
(*others*)	8%	(463 distinct primes)

Table 1: **Top 512-bit DH primes for TLS.** 8.4% of Alexa Top 1M HTTPS domains allow DHE_EXPORT, of which 92.3% use one of the two most popular primes, shown here.

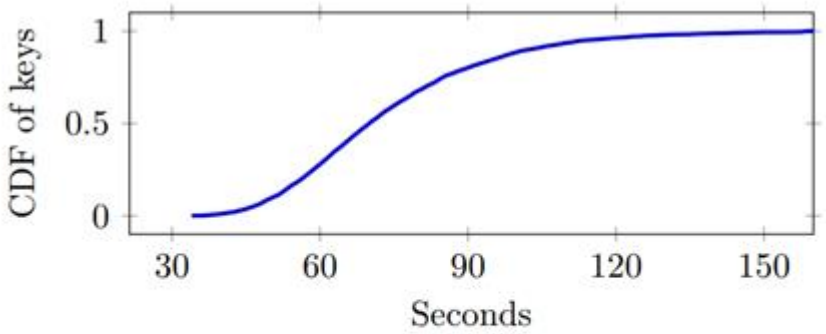


Figure 3: **Individual discrete log time for 512-bit DH.** After a week-long precomputation for each of the two top export-grade primes (see Table 1), we can quickly break any key exchange that uses them. Here we show times for computing 3,500 individual logs; the median is 70 seconds.

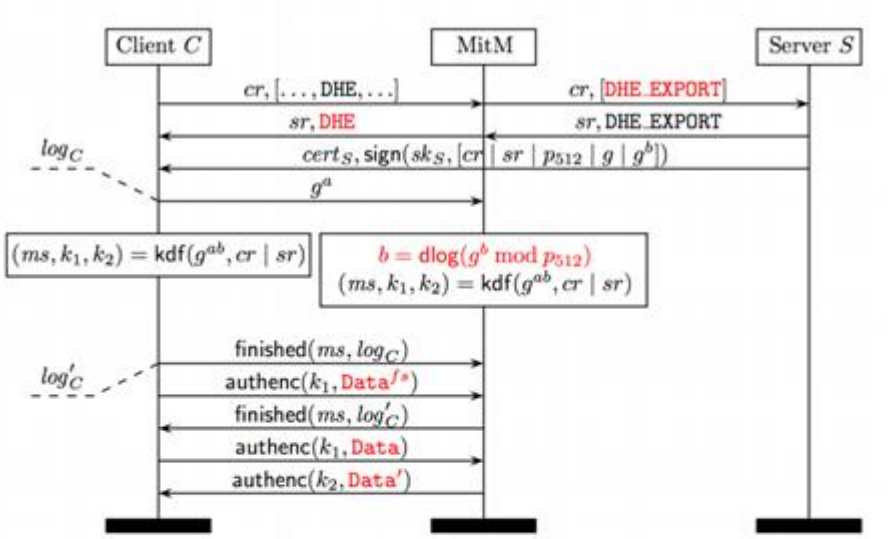


Figure 2: **The Logjam attack.** A man-in-the-middle can force TLS clients to use export-strength DH with any server that allows DHE_EXPORT. Then, by finding the 512-bit discrete log, the attacker can learn the session key and arbitrarily read or modify the contents. Data^{fs} refers to False Start [30] application data that some TLS clients send before receiving the server's Finished message.

Is NSA Breaking 1024-bit DH?

Our calculations suggest that it is plausibly within NSA's resources to have performed number field sieve precomputations for at least a small number of 1024-bit Diffie-Hellman groups. This would allow them to break any key exchanges made with those groups in close to real time. If true, this would answer one of the major cryptographic questions raised by the Edward Snowden leaks: How is NSA defeating the encryption for widely used VPN protocols?

Cont.

Classified documents published by Der Spiegel [46] indicate that NSA is passively decrypting IPsec connections at significant scale. The documents do not describe the cryptanalytic techniques used, but they do provide an overview of the attack system architecture. After reviewing how IPsec key establishment works, we will use the published information to evaluate the hypothesis that the NSA is leveraging precomputation to calculate discrete logs at scale.

IKE Internet Key Exchange (IKE) is the main key establishment protocol used for IPsec VPNs. There are two versions, IKEv1 [22] and IKEv2 [25], which differ in message structure but are conceptually similar.

Each IKE session begins with a Phase 1 handshake, in which the client and server select a Diffie-Hellman group from a small set of standardized parameters and perform a key exchange to establish a shared secret. The shared secret is combined with other cleartext values transmitted by each side, such as nonces and cookies, to derive a value called SKEYID. IKE provides several authentication mechanisms, including symmetric pre-shared keys (PSK); when IKEv1 is authenticated with a PSK, this value is incorporated into the derivation of SKEYID.

The resulting SKEYID is used to encrypt and authenticate a Phase 2 handshake. Phase 2 establishes the parameters and key material, KEYMAT, for a cryptographic transport protocol used to protect subsequent traffic, such as Encapsulating Security Payload (ESP) [27] or Authenticated Header (AH) [26]. In some circumstances, this phase includes an additional round of Diffie-Hellman. Ultimately, KEYMAT is derived from SKEYID, additional nonces, and the result of the optional Phase 2 Diffie-Hellman exchange.

The initial phases of the attack involve collecting IKE and ESP payloads and determining whether the traffic matches any tasked selector [65]. If so, TURMOIL transmits the complete IKE handshake and may transmit a small amount of ESP ciphertext to NSA's Cryptanalysis and Exploitation Services (CES) [56, 65] via a secure tunnel. Within CES, a specialized VPN Attack Orchestrator (VAO) system manages a collection of high-performance grid computing resources located at NSA Headquarters and in a data center at Oak Ridge National Laboratory, which perform the computation required to generate the ESP session key [61, 62, 67]. VAO also maintains a database, CORALREEF, that stores cryptographic values, including a set of known PSKs and the resulting "recovered" ESP session keys [60, 61, 67].

Evidence for a discrete log attack While the ability to decrypt VPN traffic does not by itself indicate a defeat of Diffie-Hellman, there are several features of IKE and the VAO's operation that support this hypothesis.

The IKE protocol has been extensively analyzed [9, 36], and is not believed to be exploitable in standard configurations under passive eavesdropping attacks. In order to recover the session keys for the ESP or AH protocols, the attacker must at minimum recover the SKEYID generated by the Phase 1 exchange. Absent a vulnerability in the key derivation function or transport encryption, this requires the attacker to recover a Diffie-Hellman shared secret after passively observing an IKE handshake.

Cont.

	Vulnerable servers, if the attacker can precompute for . . .			
	all 512-bit groups	all 768-bit groups	one 1024-bit group	ten 1024-bit groups
HTTPS Top 1M w/ active downgrade	45,100 (8.4%)	45,100 (8.4%)	205,000 (37.1%)	309,000 (56.1%)
HTTPS Top 1M	118 (0.0%)	407 (0.1%)	98,500 (17.9%)	132,000 (24.0%)
HTTPS Trusted w/ active downgrade	489,000 (3.4%)	556,000 (3.9%)	1,840,000 (12.8%)	3,410,000 (23.8%)
HTTPS Trusted	1,000 (0.0%)	46,700 (0.3%)	939,000 (6.56%)	1,430,000 (10.0%)
IKEv1 IPv4	–	64,700 (2.6%)	1,690,000 (66.1%)	1,690,000 (66.1%)
IKEv2 IPv4	–	66,000 (5.8%)	726,000 (63.9%)	726,000 (63.9%)
SSH IPv4	–	–	3,600,000 (25.7%)	3,600,000 (25.7%)

Table 3: **Estimated impact of Diffie-Hellman attacks.** We use Internet-wide scanning to estimate the number of real-world servers for which typical connections could be compromised by attackers with various levels of computational resources. For HTTPS, we provide figures with and without downgrade attacks on the chosen ciphersuite. All others are passive attacks.

RECOMMENDATIONS

Transition to elliptic curves. Transitioning to elliptic curve Diffie-Hellman (ECDH) key exchange with appropriate parameters avoids all known feasible cryptanalytic attacks. Current elliptic curve discrete log algorithms for strong curves do not gain as much of an advantage from precomputation. In addition, ECDH keys are shorter than in "mod p" Diffie-Hellman, and shared-secret computations are faster.

Increase minimum key strengths. Server operators should disable DHE_EXPORT and configure DHE ciphersuites to use primes of 2048 bits or larger. Browsers and clients should raise the minimum accepted size for Diffie-Hellman groups to at least 1024 bits in order to avoid downgrade attacks when communicating with servers that still use smaller groups. Primes of less than 1024 bits should not be considered secure, even against an attacker

Avoid fixed-prime 1024-bit groups. For implementations that must continue to use or support 1024-bit groups for compatibility reasons, generating fresh groups may help mitigate some of the damage caused by NFS-style precomputation for very common fixed groups. However, we note that it is possible to create trapdoored primes [20, 44] that are computationally difficult to detect. At minimum, clients should check that servers' parameters use safe primes or a verifiable generation process, such as that proposed in FIPS 186 [38].

Don't deliberately weaken crypto.

RSA - Key Setup

- By Rivest, Shamir and Adleman of MIT in 1977.
- Widely used public-key scheme.

- Select two large primes at random: p, q where $p \neq q$.
- Compute system modulus $n = p*q$.

  Compute Euler's totient function (or Euler's phi function)

  $\phi(n) = (p-1)*(q-1)$.
- Select at random the encryption key e
  - where $1 < e < \phi(n)$, gcd(e, $\phi(n)$) = 1 (relatively prime or coprime).
- Calculate decryption key d

  $e*d \equiv 1 \bmod \phi(n)$

  $d = (1 + k*\phi(n))/e$
- Publish the public key: KU = {e, n}
- Keep secret the private key: KR = {d, n}

RSA Use

- To encrypt a message m the sender:
  - Retrieves the public key of the recipient KU = {e, n}.
  - Computes: $c = m^e \bmod n$, where $1 \leq m < n$.

- To decrypt the ciphertext c the recipient:
  - Uses its private key KR = {d, n}.
  - Computes: $m = c^d \bmod n$.
  - $= (m^e)^d \bmod n = m^{e*d} \bmod n$
  - $= m^{k*\varphi(n)+1} \bmod n$
  - $= (m^{\varphi(n)*k} \bmod n) * (m \bmod n)$
  - $= 1*m = m$.
  - [Fermat Theorem: for any integer m coprime to n, we have $m^{\varphi(n)} \equiv 1 \pmod n$, and $1 \leq m < n$]

- Note: message m must be smaller than the modulus n (block if needed).

Exponentiation Computations

- Can use the Square and Multiply Algorithm.
- Concept is based on repeatedly squaring base.

e.g., $7^5 \bmod 11 = 7^4.7^1 \bmod 11$

$= 7^2.7^2.7 \bmod 11$

$= 5.5.7 \bmod 11$

$= 3.7 \bmod 11$

$= 21 \bmod 11$

$= 10.$

RSA Example

1. Select primes: $p = 17$ & $q = 11$
2. Compute $n = p*q = 17*11 = 187$
3. Compute $ø(n) = (p-1)*(q-1) = 16*10 = 160$
4. Select e*:* gcd(e, 160) = 1; choose $e = 7$
5. Determine d*:* $e*d \equiv 1 \bmod 160$ and $d < 160$, d = 23, {since 23*7 = 161 = 10*16+1}
6. Publish public key KU = {7, 187}
7. Keep secret private key KR = {23, 187}

Given message m = 88 (88 < 187)

- Encryption: $c = 88^7 \bmod 187 = 11$.
- Decryption: $m = 11^{23} \bmod 187 = 88$.

RSA Security and Applications

- Attacking RSA:
  - Brute force key search (infeasible given size of numbers).
  - Mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N).

Part of many standards: PKCS, ITU X.509,
ANSI X9.31, IEEE P1363 (some of these have changed)

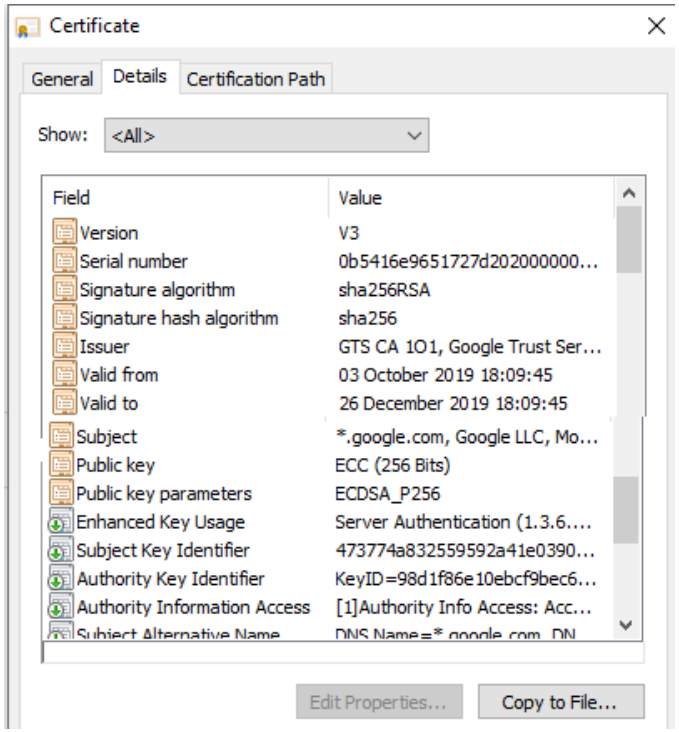
Used by: SSL, PEM, PGP, …

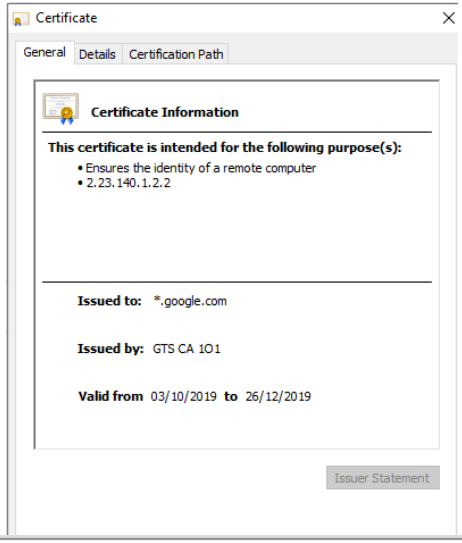
Public Key Cryptography Standards (PKCS)

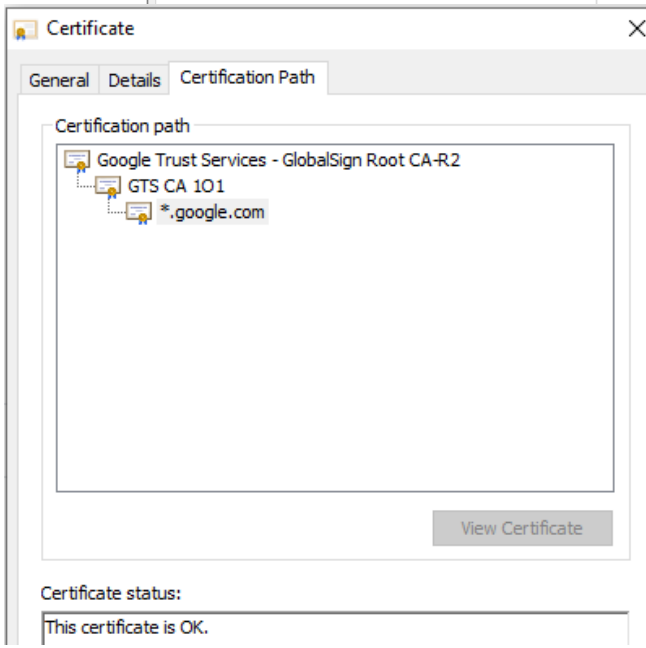
Standard Number	Standard Title	Description
PKCS#1	RSA Cryptography	Algorithms and encoding/padding schemes for performing RSA encryption, decryption, and producing and verifying signatures.
PKCS#3	Diffie–Hellman Key Agreement	Algorithms, encoding, and protocol for public secret sharing based on Diffie-Hellman
PKCS#5	Password-based Encryption	Standard and secure way to derive a secret key from a text password
PKCS#7	Cryptographic Message Syntax	Formats for signing or encrypting messages and for certificate distribution
PKCS#8	Private-Key Information Syntax	Formats for carrying private certificate key pairs (encrypted or unencrypted)
PKCS#11	Cryptographic Token Interface	Generic interface to cryptographic tokens for single sign-on, public key encryption, disk
PKCS#12	Personal Information Exchange Syntax	File format to store private keys and public key certificates, protected by a symmetric key

Digital Certificates: X.509

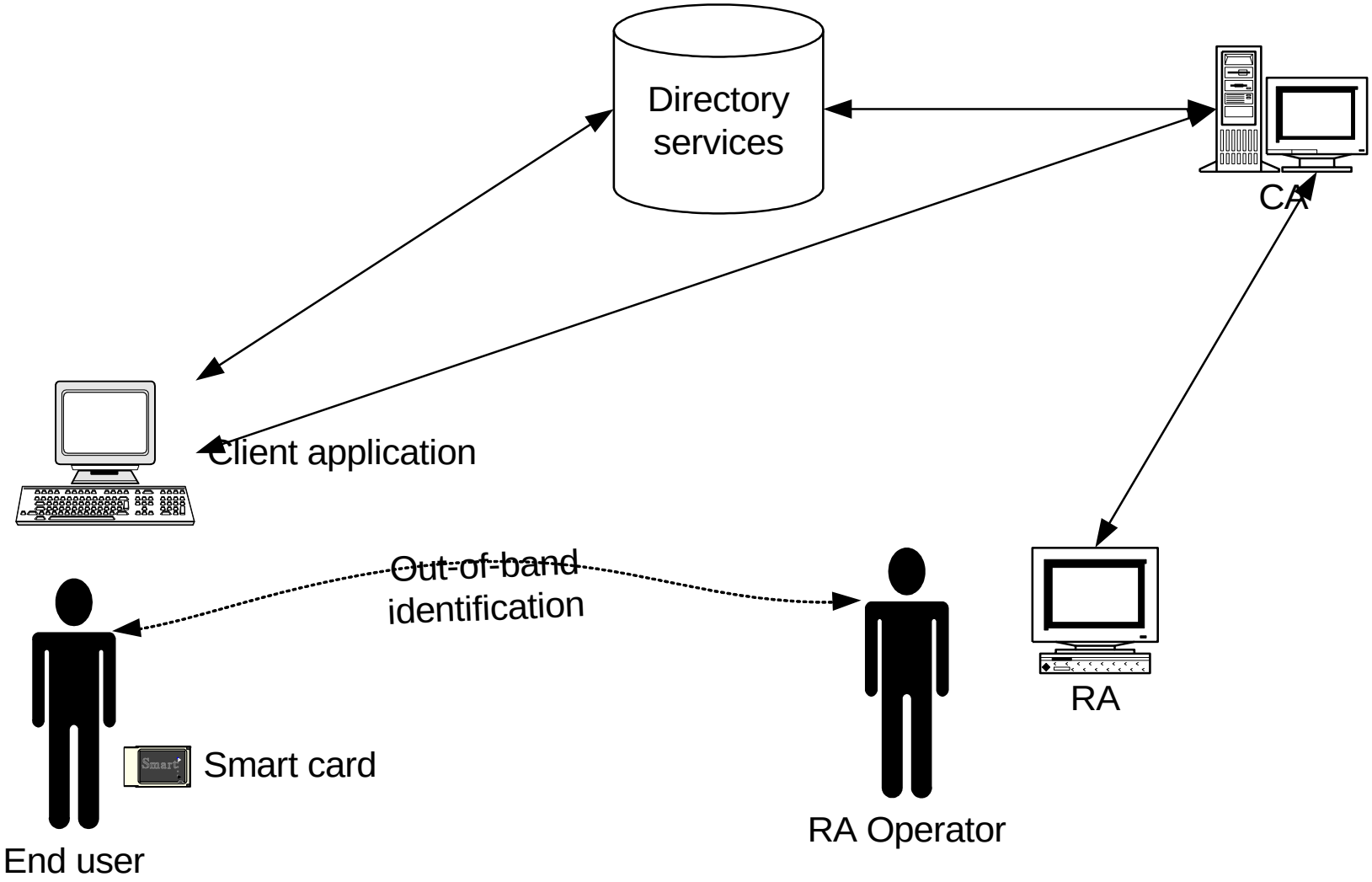
- Certificate structure:
  - Version
  - Serial number
  - Signature
  - Issuer
  - Validity
  - Subject
  - Subject public key info
  - Extension



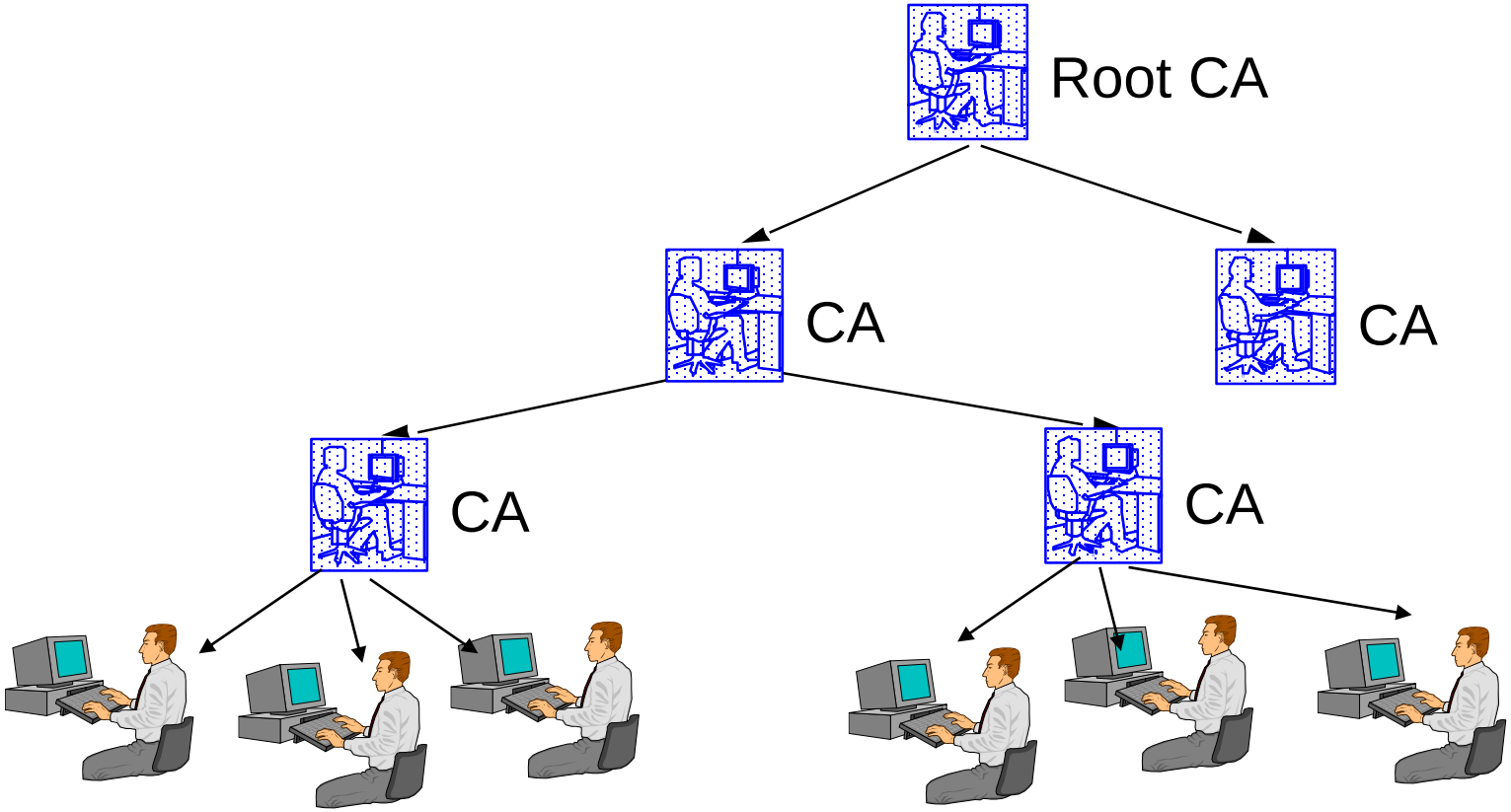




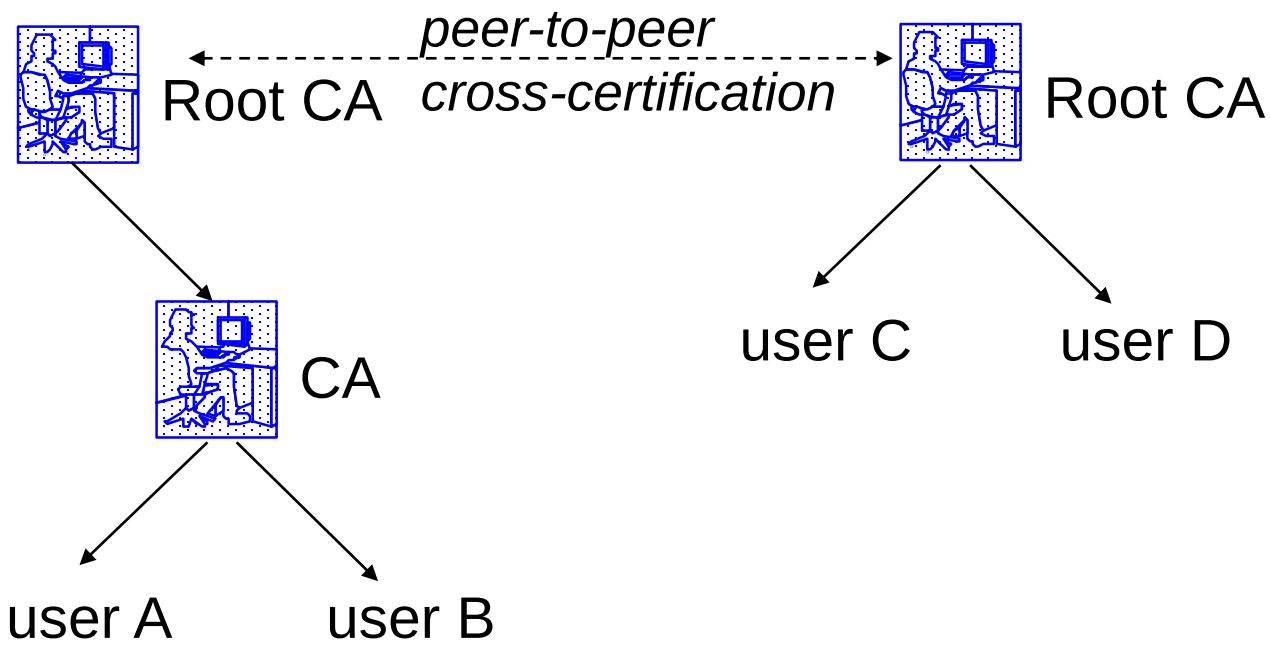
Components of a PKI



Trust Model: Hierarchical CAs

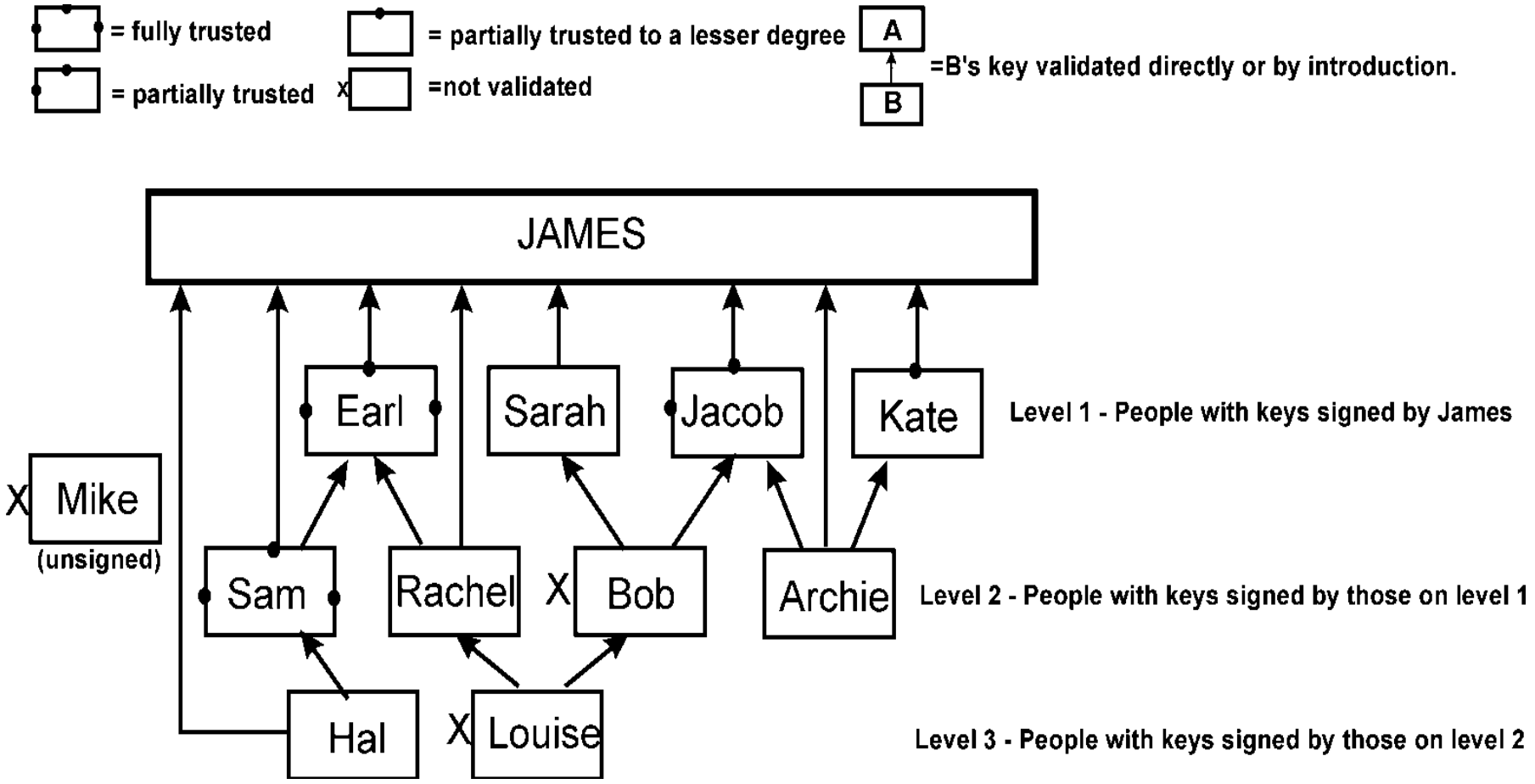


Cross-Certification



- performed between CAs
- extend trust and interoperability

PGP Trust Model



http://www.cs.bris.ac.uk/~cooper/Cryptography/crypto.html

Personal Security Environment

- Microsoft Personal Information Exchange (PFX)
- RSA Personal Security Device (PSD)
- Cybersafe Virtual Smart Card (VSC)

Key Distribution and Exchange

- Internet Engineering Task Force (IETF)
  - ISAKMP-Oakley (IPSEC)
  - SSL key exchange
  - S/MIME key exchange
  - PKIX Cert. Management protocols
  - SPKI - Simple Public Key Infrastructure

- American National Standards Institute (ANSI) Standards
  - X9.17 - Symmetric Key Management
  - X9.24 – Retail Financial Services Key Management
  - X9.42 - Diffie-Hellman key exchange

Summary

- Public key cryptography – basics
- DHKE
- RSA
- X.509
- CA and Key distribution