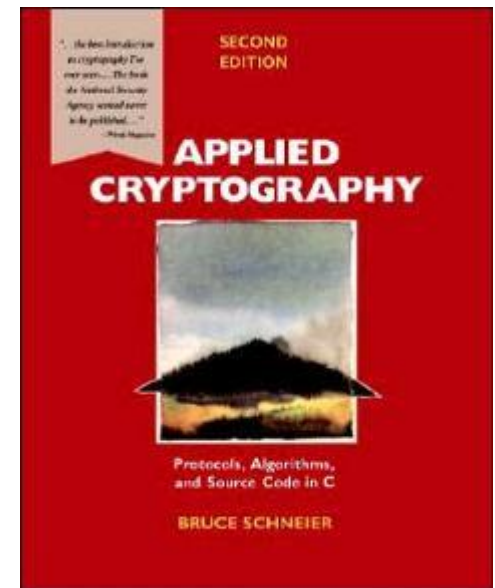
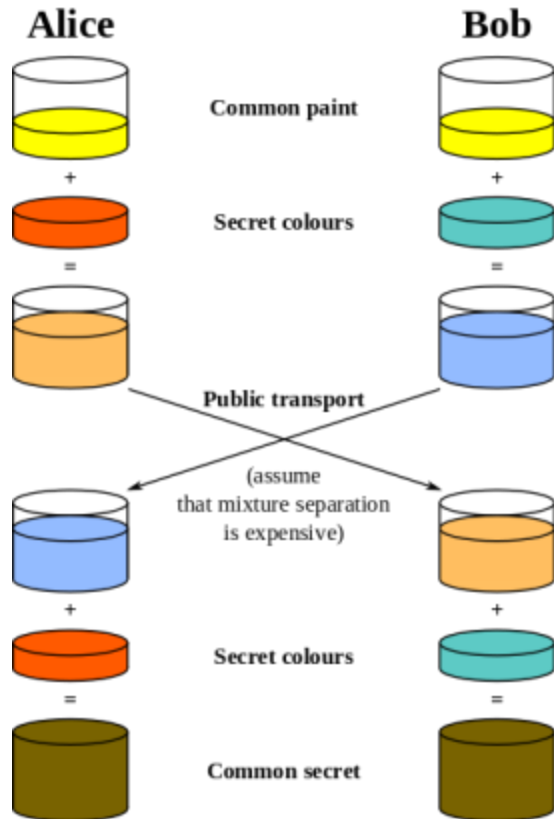


# Modern cryptography 2

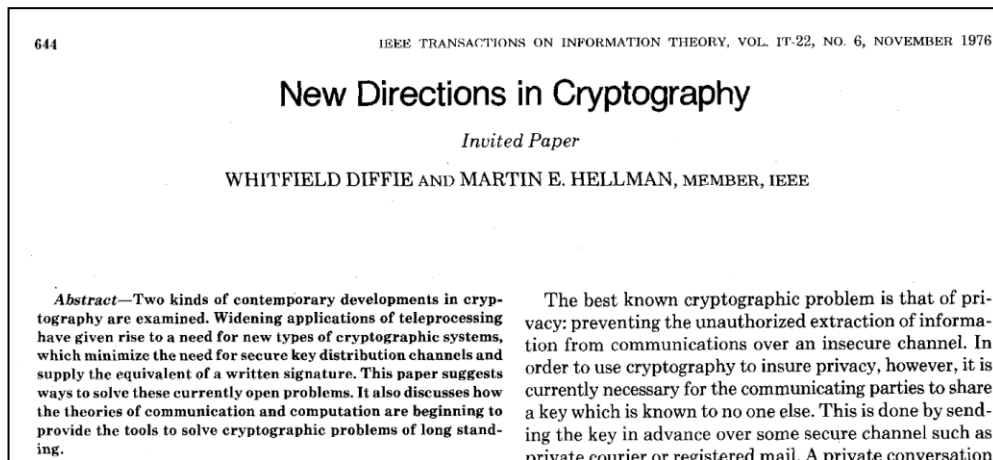


# Overview

- Modern cryptography
  - Asymmetric cryptography
    - Diffie-Hellman key exchange (last time)
    - Public key: RSA
    - Pretty Good Privacy (PGP)
  - Digital signing
  - Public key infrastructure (PKI)
  - Securing web commerce
    - SSL / TLS
    - https

# Diffie-Hellman

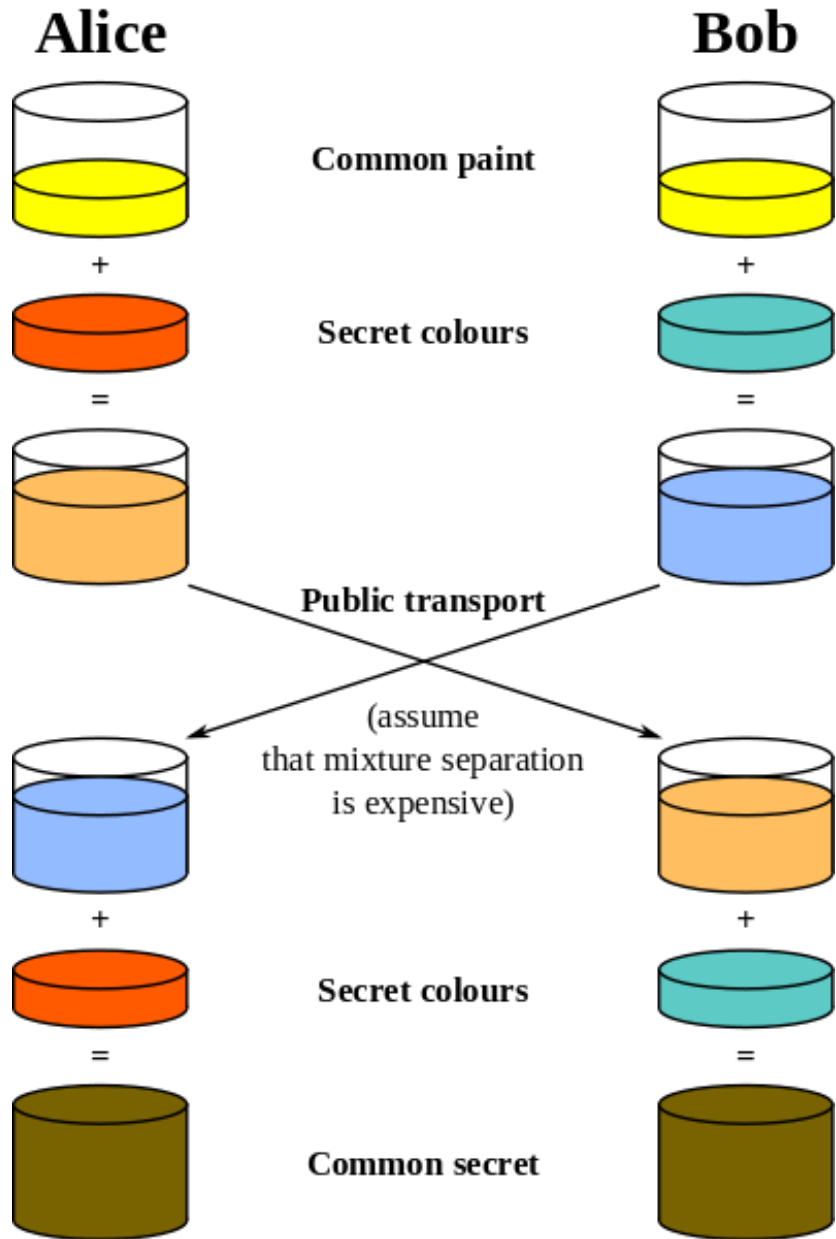
- Diffie-Hellman (DH) key exchange
  - 1976, Whitfield Diffie & Martin Hellman
  - Alice and Bob **agree on a private secret:**
    - On a public channel <http://www.youtube.com/watch?v=3QnD2c4Xovk>
    - Where Eve hears all the traffic
    - Only Alice and Bob end up knowing the secret
  - Relies on **one-way function**
    - Function must be easy to do, but difficult to undo



*Whitfield Diffie*



*Martin Hellman*



Alice	Bob
Alice and Bob agree publicly on values for Y and P for the one-way function: $Y^x \pmod{P}$ , e.g. $Y = 7, P = 11$	
Alice chooses secret number: $A = 3$	Bob chooses secret number: $B = 6$
$\alpha = 7^A \pmod{11}$ $= 7^3 \pmod{11}$ $= 343 \pmod{11}$ $= 2$	$\beta = 7^B \pmod{11}$ $= 7^6 \pmod{11}$ $= 117649 \pmod{11}$ $= 4$
Sends $\alpha = 2$ to Bob	Sends $\beta = 4$ to Alice
Using Bob's result: $\beta^A \pmod{11}$ $4^3 \pmod{11} = 9$	Using Alice's result: $\alpha^B \pmod{11}$ $2^6 \pmod{11} = 9$
Why the same? $4 = 7^6 \pmod{11}$ $= 4^3 \pmod{11}$ $= (7^6)^3 \pmod{11}$ $= 7^{B \cdot A} \pmod{11}$	$2 = 7^3 \pmod{11}$ $= 2^6 \pmod{11}$ $= (7^3)^6 \pmod{11}$ $= 7^{A \cdot B} \pmod{11}$

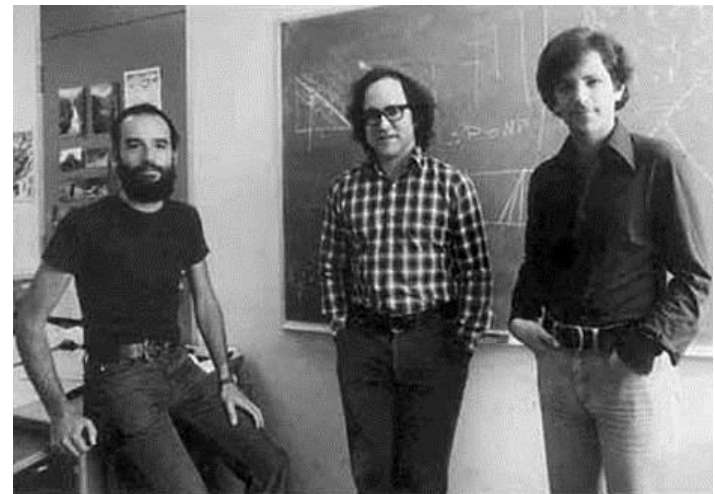
# Public key cryptography

- Diffie-Helman key exchange
  - Both parties had to be around to negotiate secret
- Symmetric encryption
  - Encrypting message  $M$  with key  $K$ :  $E_K(M) = C$
  - Decrypting ciphertext  $C$  with key  $K$ :  $D_K(C) = M$
- Asymmetric encryption
  - 1975, Diffie conceives of idea
  - Users have a **private key** and a **public key**
    - Alice encrypts plaintext with Bob's public key
    - Only Bob can (tractably) decrypt using his private key
  - Special one-way function
    - Hard to reverse unless you know something special

# RSA

- RSA public key encryption
  - 1977: Rivest, Shamir, Adleman
  - Choose two prime numbers,  $p$  and  $q$ 
    - Public key:  $N = pq$
    - Private key:  $p$  and  $q$
    - If  $N$  is product of two large primes, factoring is hard
  - 1973: equivalent algorithm, Clifford Cocks (GCHQ)

[http://www.youtube.com/watch?v=wXB-V\\_Keiu8](http://www.youtube.com/watch?v=wXB-V_Keiu8)



# RSA example



Alice	Bob
<p>Alice picks two giant primes, <math>p</math> and <math>q</math> e.g. <math>p = 61</math>, <math>q = 53</math></p> <p><math>N = p * q = 61 * 53 = 3233</math></p> <p><math>(p - 1) * (q - 1) = 60 * 52 = 3120</math> Find number <math>1 &lt; e &lt; 3120</math>, <math>e</math> is relatively prime with 3120, say <math>e = 17</math></p> <p><b>Alice's public key: <math>N = 3233</math>, <math>e = 17</math></b></p>	
	<p>Bob wants to send message 65 to Alice, looks up her public key.</p> <p><math>C = M^e \pmod{N}</math> <math>C = 65^{17} \pmod{3233} = 2790</math></p>

# RSA example



Alice	Bob
	<p>Bob wants to send message 65 to Alice, looks up her public key.</p> $C = M^e \pmod{N}$ $C = 65^{17} \pmod{3233} = 2790$
<p>Compute special number <math>d</math></p> $e * d = 1 \pmod{(p - 1) * (q - 1)}$ $17 * d = 1 \pmod{3120}$ $d = 2753 \text{ (using Euclid's algorithm)}$ <p><b>Alice's private key <math>d = 2753</math></b> (derived from <math>p</math> and <math>q</math>)</p> <p>Decrypt message:</p> $M = C^d \pmod{N}$ $M = 2790^{2753} \pmod{3233} = 65$	



# Security of RSA

- Attacks on RSA

- Brute force

- Try all possible private keys

- Use a large key space, but large keys slows things down

- Mathematical

- Factoring product of 2 large primes

- Timing

- Keep track of how long it takes to decipher messages

- Chosen ciphertext

$$O\left(\exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right)$$

*General number field sieve, b-bit number*

2009: 768-bit RSA factored using hundreds of machines in 2 years

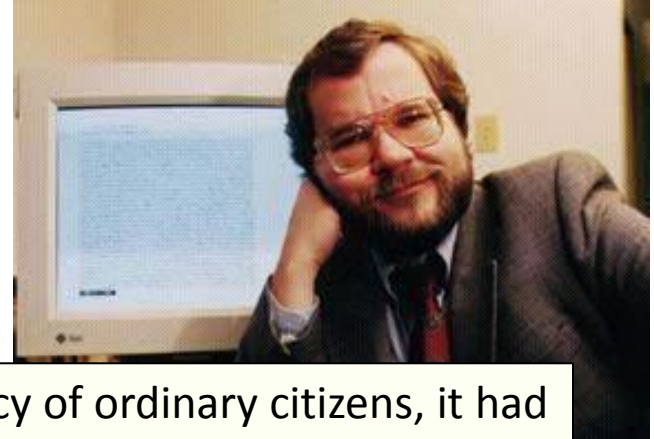
**Unsolved problems in computer science**

*Can integer factorization be done in polynomial time?*



# PGP

- **Problem:** RSA hard to use, resource intensive
- **Pretty Good Privacy (PGP)**
  - 1991 Phil Zimmermann



"In the past, if the Government wanted to violate the privacy of ordinary citizens, it had to expend a certain amount of effort to intercept and steam open and read paper mail, and listen to and possibly transcribe spoken telephone conversation. This is analogous to catching fish with a hook and a line, one fish at a time. Fortunately for freedom and democracy, this kind of labor-intensive monitoring is not practical on a large scale.

Today, electronic mail is gradually replacing conventional paper mail, and is soon to be the norm for everyone, not the novelty it is today. Unlike paper mail, E mail messages are just too easy to intercept and scan for interesting keywords. This can be done easily, routinely, automatically, and undetectably on a grand scale. This is analogous to driftnet fishing -- **making a quantitative and qualitative Orwellian difference to the health of democracy.**"

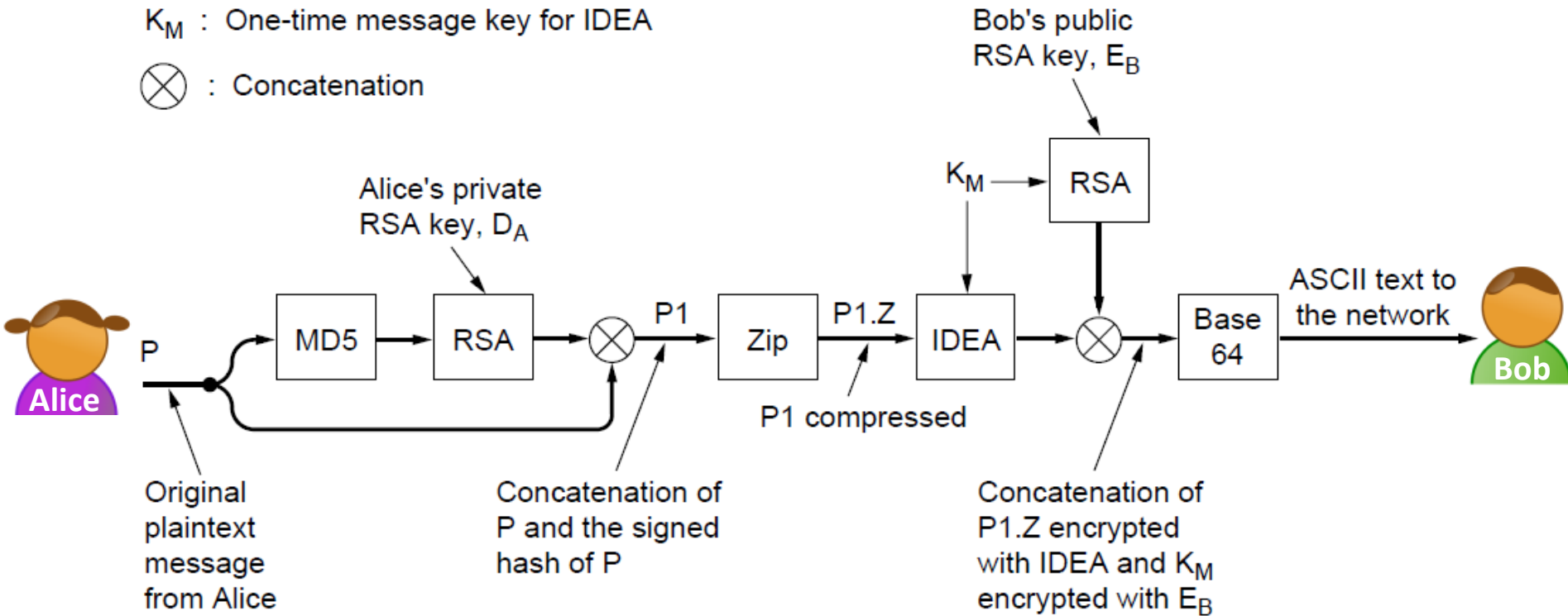
*-Philip Zimmermann, testimony to Congress*

# PGP

- Pretty Good Privacy (PGP)
  - Focus on efficiency:
    - RSA for symmetric key exchange
    - Symmetric cipher (IDEA) for bulk of encryption
  - Focus on ease of use:
    - Allow average Joe to use strong cryptography
    - User clicks to encrypt/sign an email
  - First widely available public-key crypto
    - Released via friend to the Usenet
  - Problems:
    - RSA was patented by RSA Data Security, Inc.
    - Strong encryption considered a munition by US

$K_M$  : One-time message key for IDEA

$\otimes$  : Concatenation

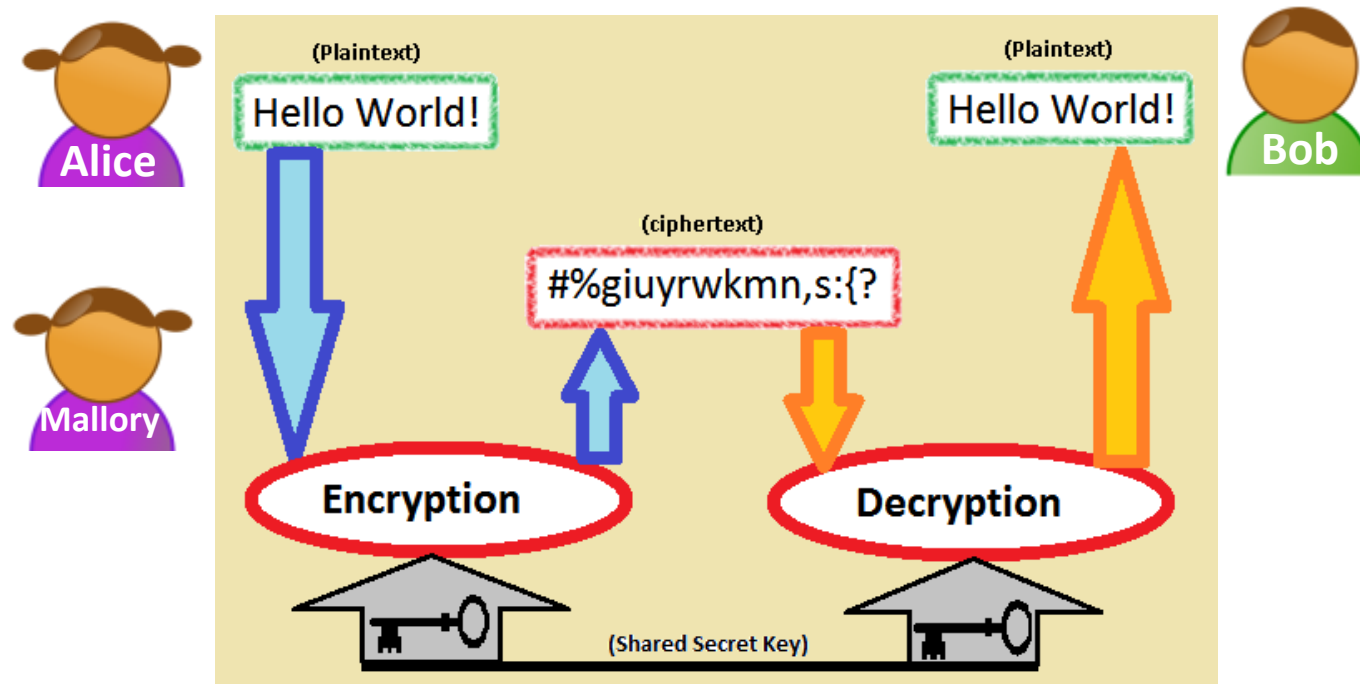


## • Asymmetric key lengths

– Need to be longer than symmetric keys

- 384 bits = casual, broken easily today
- 512 bits = commercial, breakable by 3-letter orgs
- 1024 bits = military, not breakable on earth
- 2048 bits = alien, unbreakable on other planets

# Digital signing

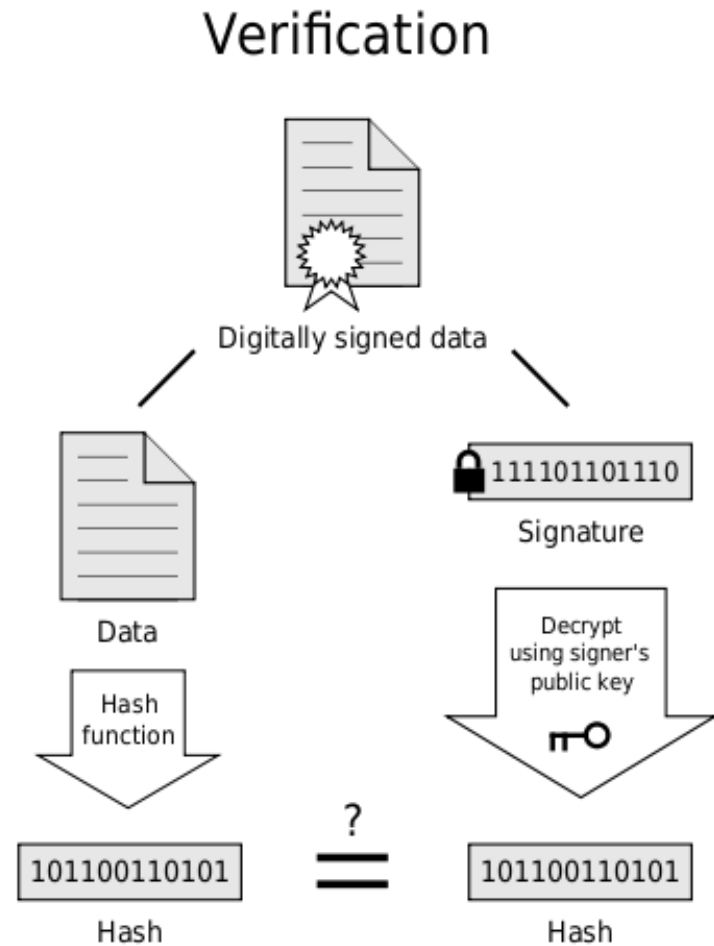
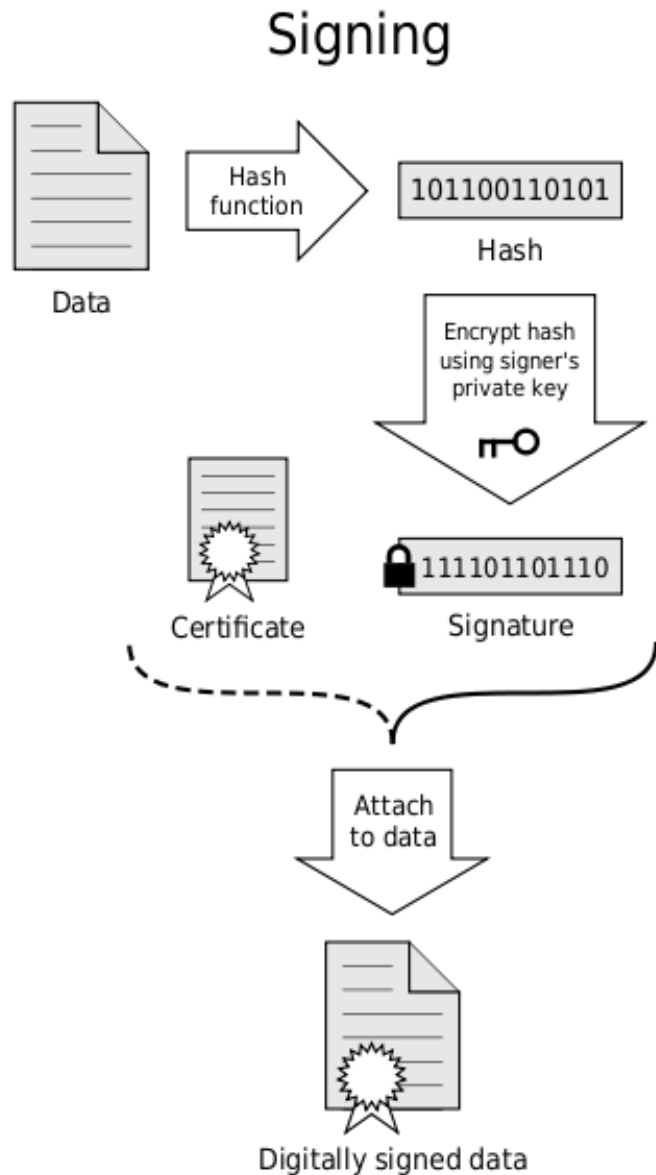


- **Problem: Impersonation in public-key crypto**
  - Mallory encrypts message with Bob's public key
  - Only Bob can decrypt using his private key
  - Message is a love letter claiming to be from Alice

# Digital signing

- Digital signing via public key crypto
  - Alice encrypts message with her *private* key
    - Everybody can decrypt using Alice's public key
    - But proves message came from Alice since no one else has her private key
  - Alice can additionally encrypt using Bob's public key
    - Only Bob can decrypt using his private key
    - Verify authorship by decrypting with Alice's public key
- **Problem:** Signing entire message expensive
  - Hash the message
  - Encrypt just the hash

# Hash-based digital signing



If the hashes are equal, the signature is valid.

# Distributing public keys

- Alice needs Bob's public key
  - Downloads Bob's key from some web site
    - How does she know it is really Bob's key?
  - Man in the middle attack:
    - Mallory fools Alice into using fake Bob public key
    - Mallory decrypts using fake Bob's private key
    - Mallory reads message
    - Re-encrypts using Bob's real public key and sends on
    - Alice and Bob think there are communicating securely but actually aren't
- **Problem 1:** How to distribute public keys?
- **Problem 2:** How to establish trust of keys?



# PKI

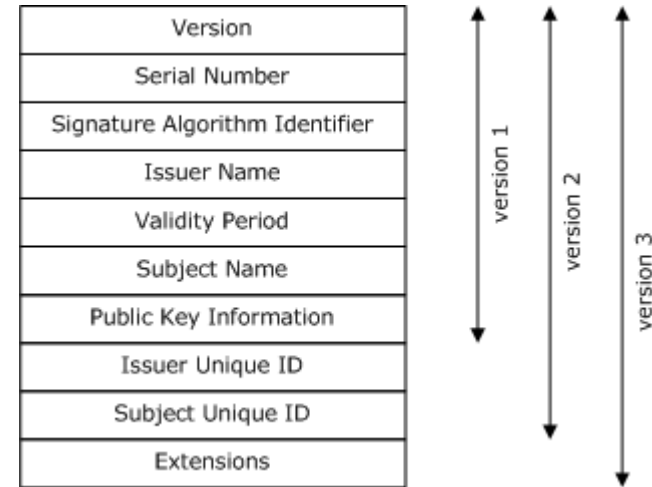
- **Public Key Infrastructure (PKI)**

- Digital certificate

- Prove ownership of a public key
- e.g. X.509

- Certificate Authority (CA)

- Trusted 3<sup>rd</sup> party, validates identity of person/org
- Digitally signs and publishes public key bound to a user
- Signed with CA's private key
- CA's public key trusted by user, e.g. by web browser

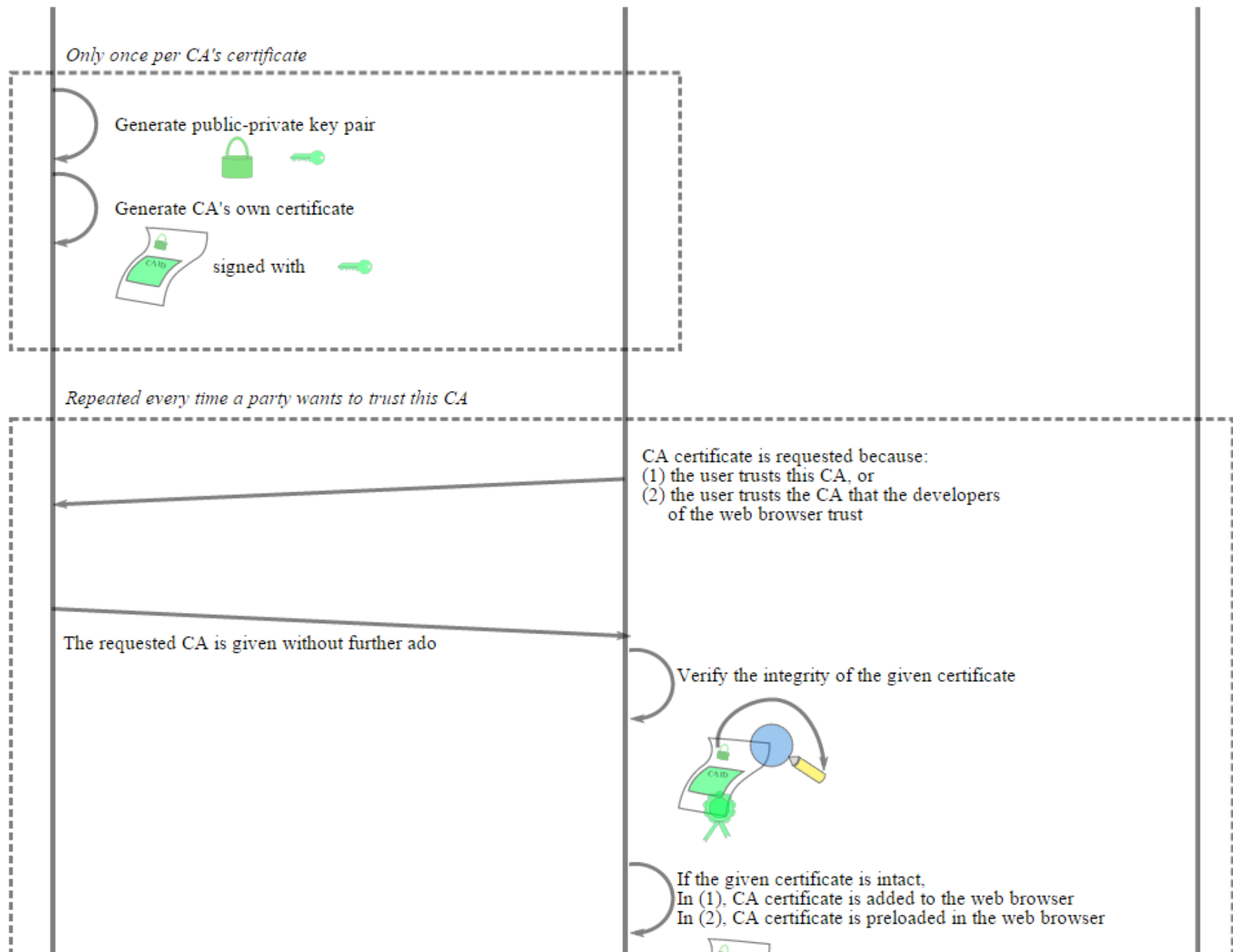


*Fields in a X.509 certificate*

Rank	Issuer	Usage	market share
1.	Comodo	6.6%	33.6%
2.	Symantec Group	6.5%	33.2%
3.	Go Daddy Group	2.6%	13.2%
4.	GlobalSign	2.2%	11.3%
5.	DigiCert	0.6%	2.9%

# PKI

- **Public Key Infrastructure (PKI)**
  - Registration Authority (RA)
    - Optional component
    - Handles administration functions:
      - Accept requests
      - Authenticate person/organization
      - Make request to CA
  - Certificate repository
    - Publically accessible location of certificates/keys



Certificate Authority (CA)

User's Web Browser

Developer

*Only once per developer*


Generate public-private key pair



*Repeated for each CA by which the developer wants to be certified*

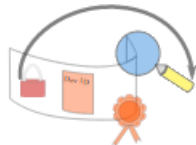
Generate CSR (Certificate Signing Request) per PKCS10 (Public-key Cryptography Standard #10)



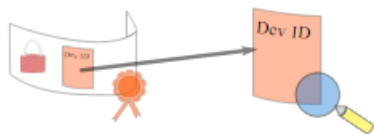
signed with 

Send CSR

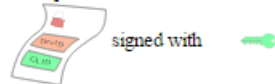
Check the integrity of the CSR



If the CSR is intact, verify that the developer's true identity matches the one contained in the CSR



If CSR integrity is intact and the true identity matches the one contained in the CSR, generate developer's certificate



Send the signed developer's certificate

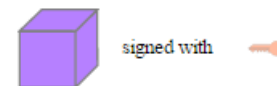
The signed certificate is kept



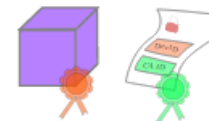
*Repeated for each digital object for distribution*

Request the digital object

Sign a digital object to be distributed



Distribute the digital object and the certificate



If the user has the signing CA certificate in the browser, the user has decided to trust the signing CA



If the signing CA is trusted, the CA's public key is used to verify the integrity of the developer's certificate



If the developer's certificate is intact, which implies that the CA has guaranteed that the information is correct, the user that trusts the CA can trust that the developer's ID is valid and the public key really belongs to the developer. Then the public key can be used to verify the integrity of the requested object



If the object is intact, the user can assume that the object really originates from the developer because only the developer can sign the object with the right private key and the developer that signs the object is really the developer with the claimed ID since the CA has verified it.

# Securing web commerce

- Customer fills out order with credit card #
  - **Problem 1:** Keep data secure from customer's browser to the web server
  - **Problem 2:** Keep data secure on server or in transit to order fulfillment

# HTTPS

- Hypertext Transfer Protocol Secure (HTTPS)
  - https://
  - Typically running on port 443

Application (HTTP)
Security (SSL)
Transport (TCP)
Network (IP)
Data link (PPP)
Physical (modem, ADSL, cable TV)

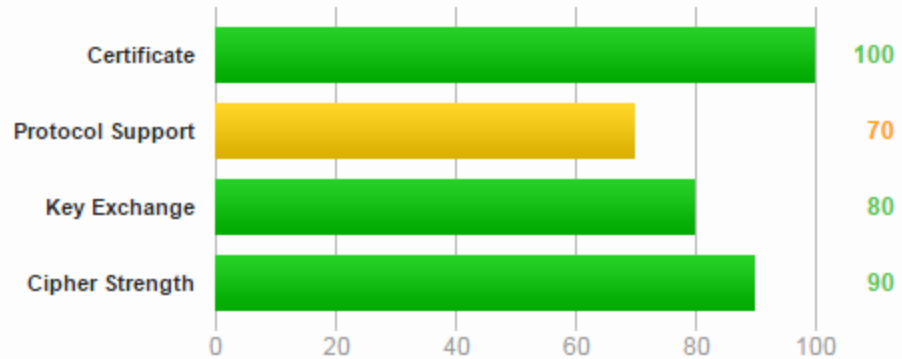
# SSL

- Secure Sockets Layer (SSL) / Transport Layer Security (TLS)
  - Client requests secure connection from server
  - Client sends supported ciphers & hashes
  - Server picks strongest mutual cipher & hash
  - Server sends back digital certificate
  - Client contacts CA to confirm key belongs to site
  - Client generates session key by encrypting random number with server's public key
  - Client and server switch to symmetric cipher



# Summary

## Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This server is vulnerable to the POODLE attack. If possible, disable SSL 3 to mitigate. Grade capped to C. [MORE INFO »](#)

This server does not mitigate the [CRIME attack](#). Grade capped to B.

Certificate has a weak signature and expires after 2016. Upgrade to SHA2 to avoid browser warnings. [MORE INFO »](#)

The server supports only older protocols, but not the current best TLS 1.2. Grade capped to B.

This server accepts the RC4 cipher, which is weak. Grade capped to B. [MORE INFO »](#)

The server does not support Forward Secrecy with the reference browsers. [MORE INFO »](#)

# Summary

- **Modern cryptography**
  - Asymmetric cryptography
    - Diffie-Hellman: exchange of secrets on public channel
    - RSA: public key encryption
    - PGP: end-user application of public key encryption
  - Digital signing
    - Prove authorship via asymmetric cryptography
  - Public key infrastructure (PKI)
    - Publicize/verify public keys
  - Securing web commerce
    - SSL/TLS