



# Introduction to Cryptography

**Security and Privacy**  
**VSE Prag, 9 - 13.6.2008**

Institute for Information Processing and  
Microprocessor Technology (FIM)  
Johannes Kepler University Linz, Austria

E-Mail: [sonntag@fim.uni-linz.ac.at](mailto:sonntag@fim.uni-linz.ac.at)  
<http://www.fim.uni-linz.ac.at/staff/sonntag.htm>



- General aspects
  - Why and where to use
- Technical aspects
  - Symmetric vs. asymmetric cryptography
  - Algorithms and their strength, required environment
- Encryption/signing: Diffie-Hellman, RSA, AES
- Hash algorithms: MD5, SHA-1, SHA-256
- Certificates
  - Content, PKI, revocation
- SSL/TLS
  - Modes, protocol
- VPNs: IPSec, PPTP, OpenVPN



# Why cryptography?

- Security is a very important aspect, especially if money (or equivalents) is contained in transactions
- Not everything information should be available to everyone
  - Note: Data is sent in the Internet over numerous "open systems", where anyone can listen it!

**Security is needed!**

- The technical aspect of security is cryptography
  - Encrypting data against disclosure and modifications
  - Signing data against modifications and repudiation
- Note: Cryptography **does not** solve **all** security problems!
  - Example: Communication analysis (who talks to whom when)
  - Other aspects of security are also needed
    - » E.g.: Do you know what your employees actually do with data?
  - Solutions: DRM, deactivation codes, anonymizers, ...



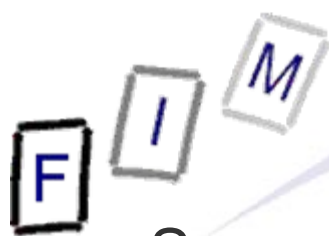
# Application areas

- Storing data in encrypted form
  - Even access will not lead to disclosure (Stolen Laptops!)
  - Example: File/file system encryption programs
- Transmitting data securely
  - Enc. transmission prevents eavesdropping and tampering
  - Example: TLS
- Identifying your partner
  - Preventing man-in-the-middle attacks
  - Example: TLS with uni-/bidirectional certificates
- Proof of identity
  - Avoiding impersonation
  - Example: GPG E-Mail signatures, digital signatures ("Bürgerkarte")



- Data is being
  - transmitted
  - stored
  - processed
- and exposed to the following attacks:
  - Inspection: without / with understanding
  - Modification: without / with understanding
  - Deletion: random / targeted
  - Addition: random / targeted
  - Replay: with / without knowledge of consequences

**Those cases where the attacker can understand the data/consequences (here underlined), are more problematic!**

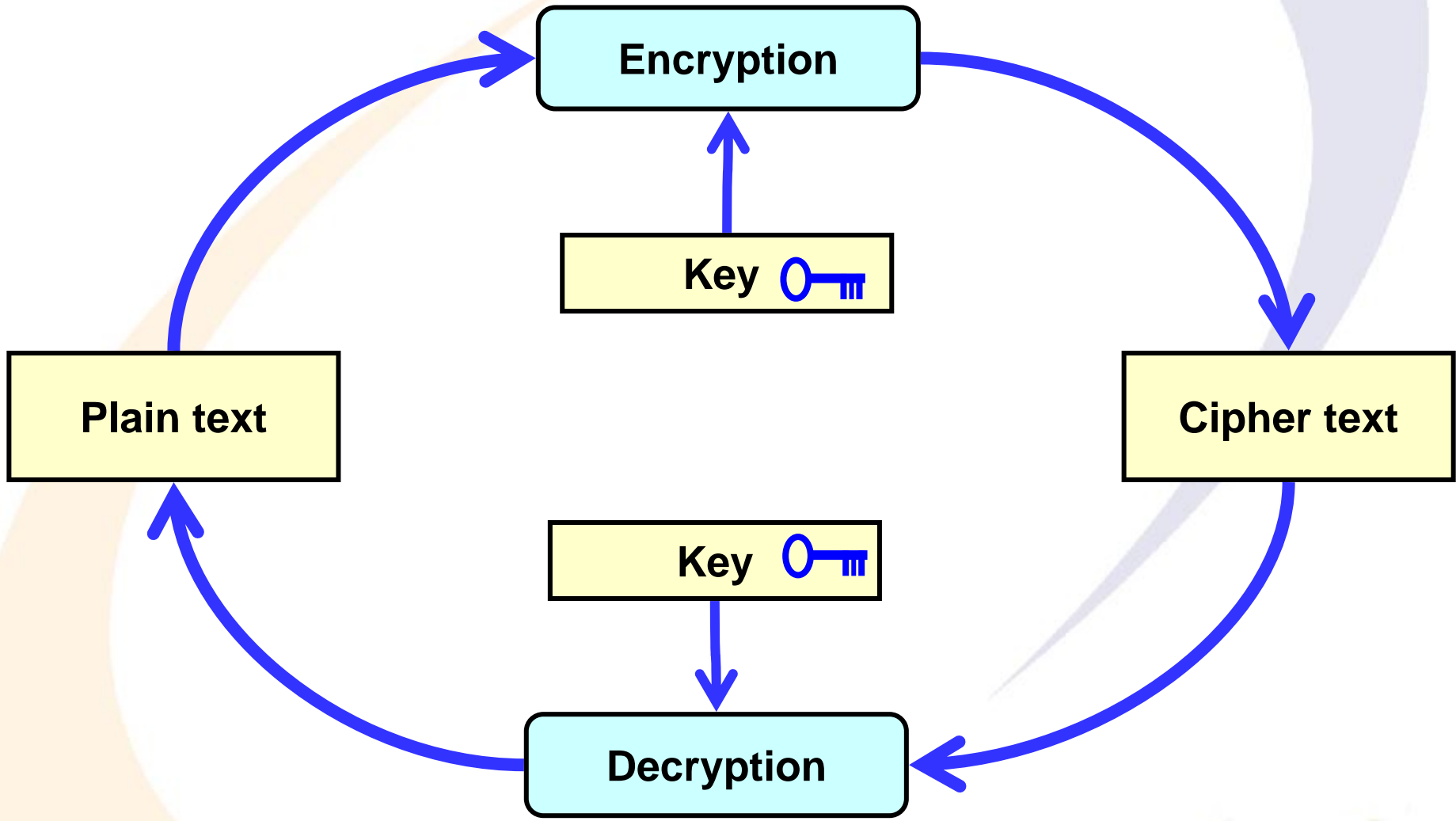


# Software components

- Several different classes of algorithms required:
  - Hash functions: Handling the whole document takes too long
    - » Drawback: Content could be substituted (collisions)!
  - Encryption/Decryption: The same algorithm for symmetric, but different one for asymmetric encryption/signatures
    - » Encryption: Combining a document with a public key
    - » Signature: Combining a document with a private key
    - » Verification: Checking a document + signature with public key
  - Key agreement: Creating a shared secret
    - » Even if both parties do **not** have a shared secret to start with!
    - » Especially useful if the communication channel is insecure
  - Key generation: Creating secure keys
    - » Requires e.g. secure random generators
    - » From passwords: Creating keys suitable for algorithms
- For each class several/many algorithms exist
  - Some good, some bad (=broken, erroneous, ...)



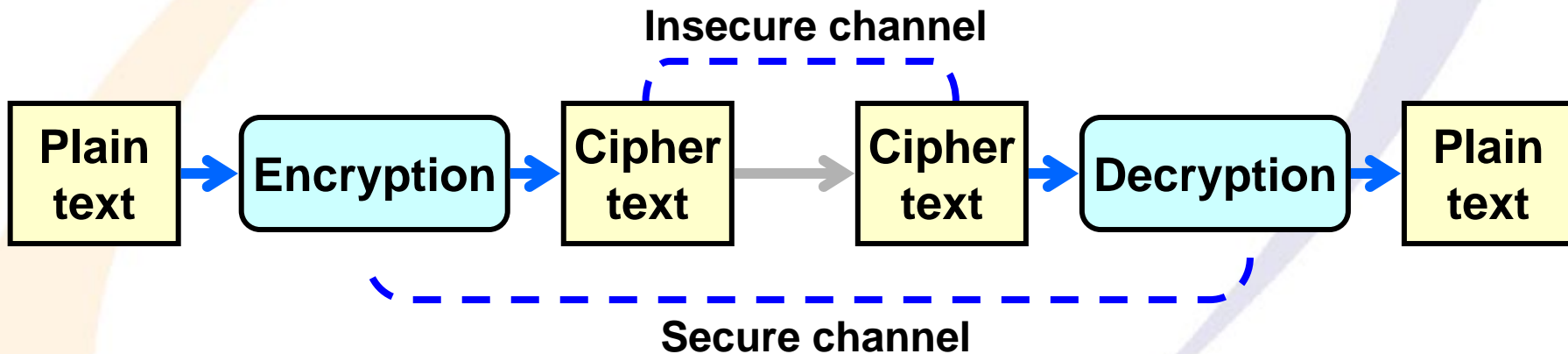
# Basic functionality cryptography



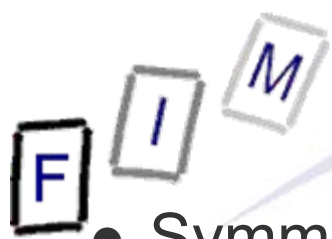


# Common requirements

- Key comparatively very small
- Data to encrypt very large



Gutmann P.: „Network Security: Security Requirements“, University of Auckland,  
<http://www.cs.auckland.ac.nz/~pgut001>

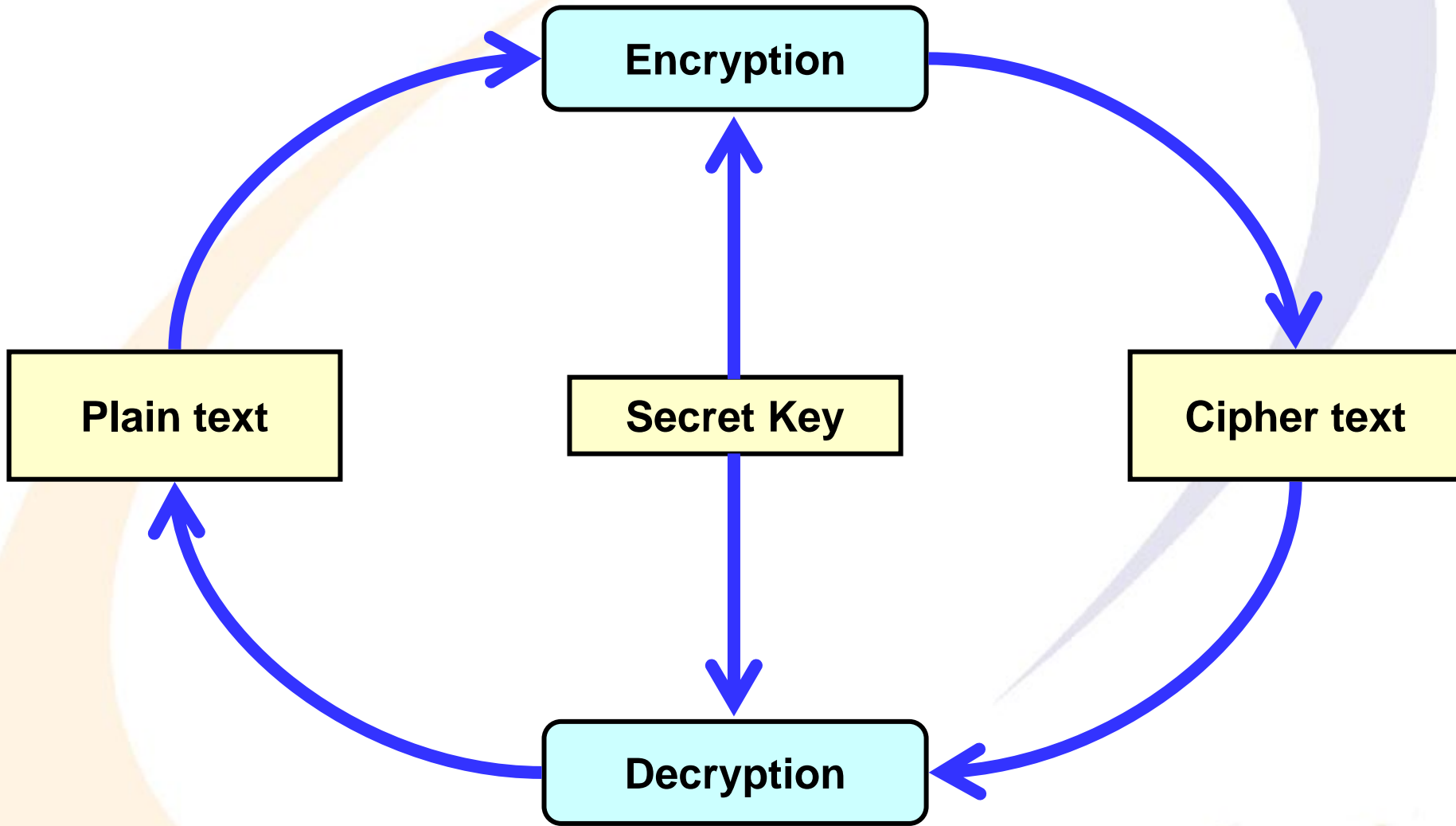


# Symmetric vs. asymmetric cryptography

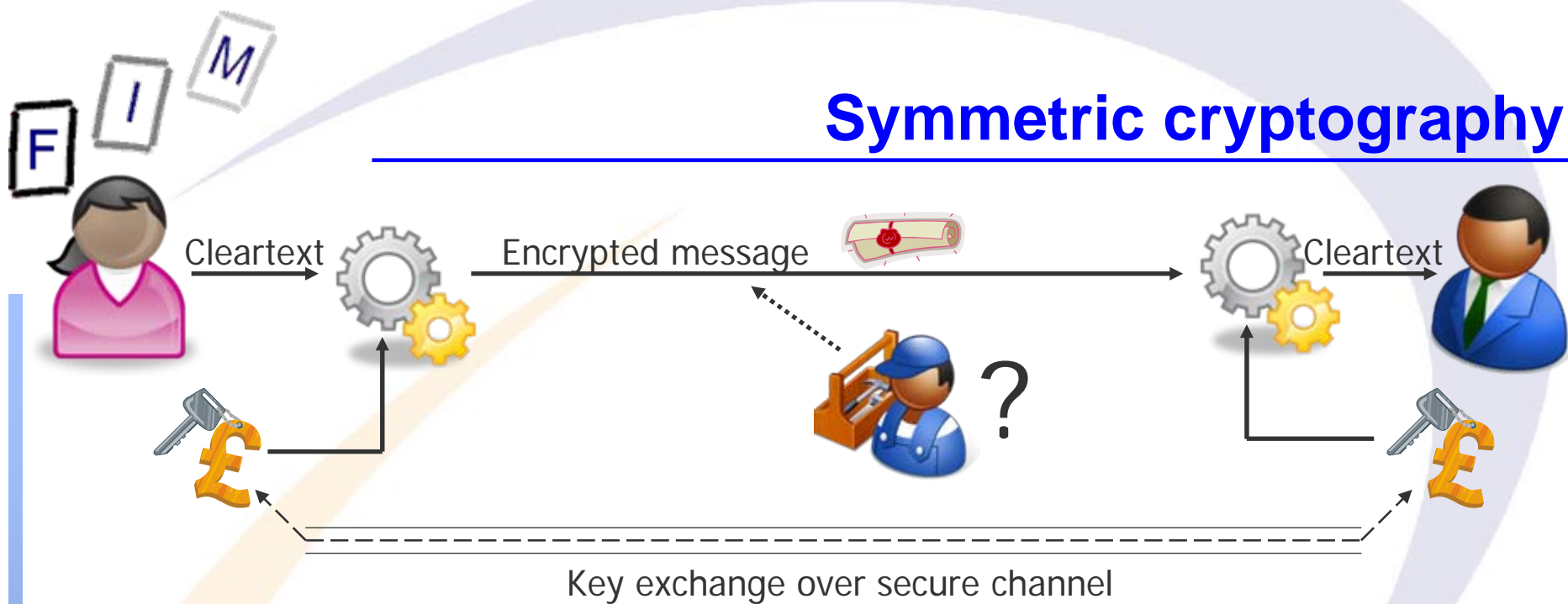
- Symmetric cryptography: Very old
  - Keys are short:  $\geq 56$  Bits (DES)
    - » Many algorithms known (many of them now insecure!)
  - Well suited to a large (homogenous) group of participants
    - » Everyone knows the same key and can en-/decrypt messages from all others
  - Key distribution: Problematic (Secure channel needed)
  - Computationally very fast (1/100 – 1/1000 of asymm.)
- Asymmetric cryptography: Very new ( $\approx 1970$ )
  - Keys are long:  $\geq 512$  Bits (RSA/DSA; Ell. curves  $\geq 112$ )
    - » Few algorithms known (most of them very secure)
    - » Keys consist of two parts: One public, one private
  - Only suitable for a single person
    - » Encrypt to this person or verify signatures from this person
  - Certification distribution: Problematic (common TTP needed)!
  - Computationally slow: Calculation difficult/time-consuming



# Symmetric (secret key) encryption



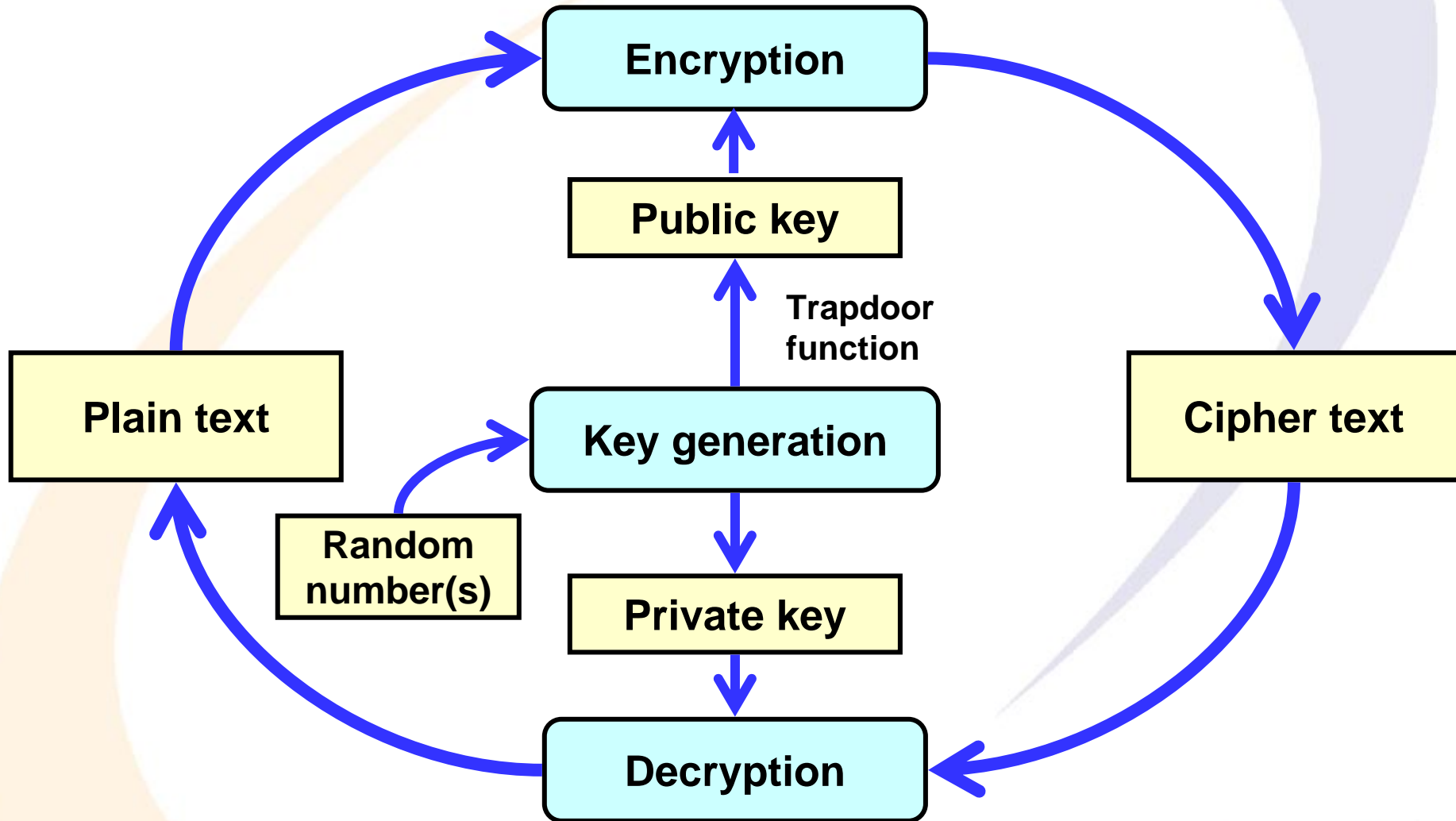
# Symmetric cryptography



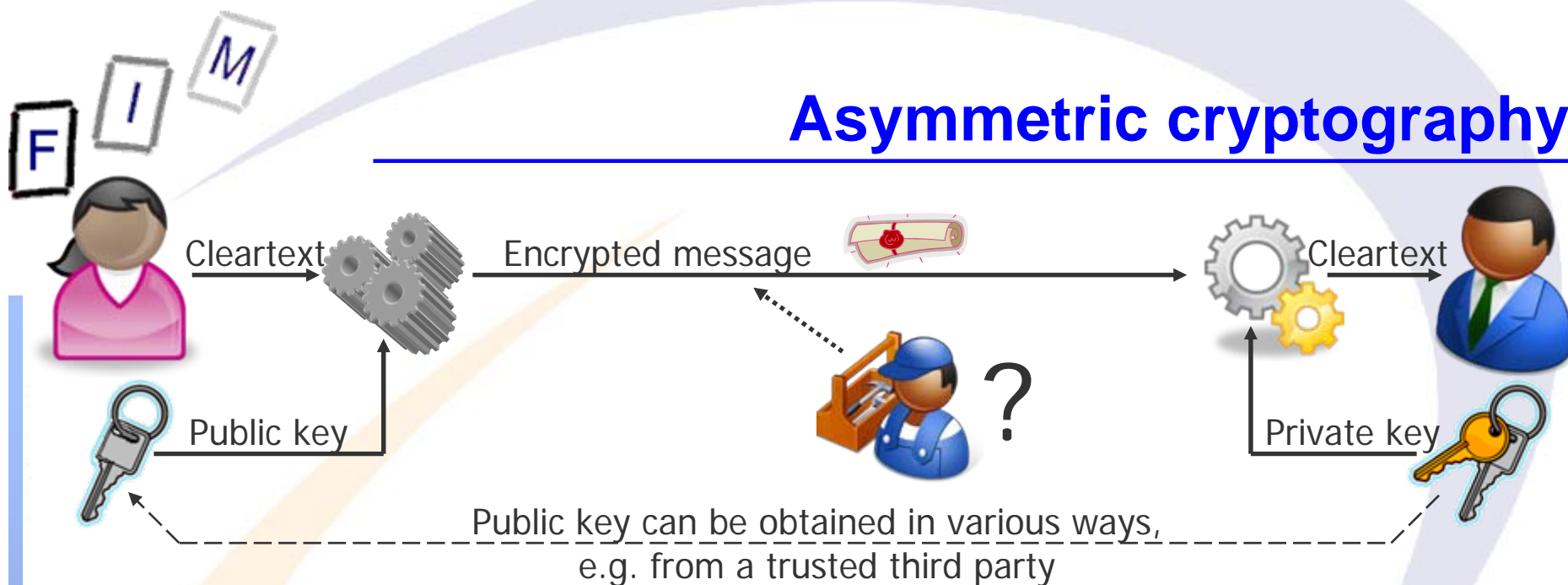
- The same key is used for encryption and decryption
  - The key must therefore remain absolutely secret!
  - So it must be transported to each other securely
    - » This is only possible through a different channel!
  - Keys must be changed regularly (much encrypted content renders breaking the encryption easier!)
- Also used for authentication (repudiation possible!)
  - "MAC" Message Authentication Code



# Asymmetric (public key) encryption

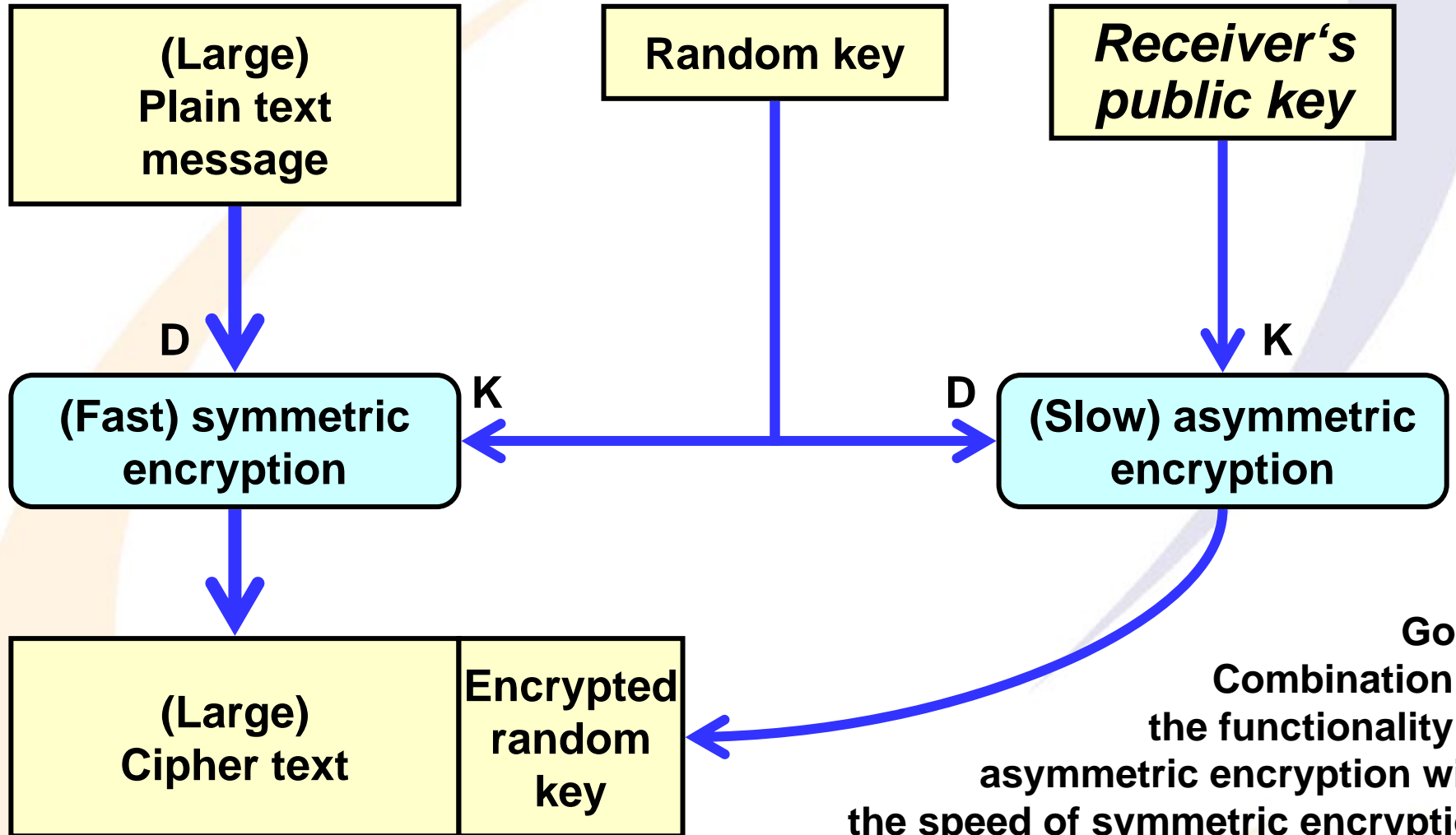


# Asymmetric cryptography



- One part (private) of the key is used for decryption/signature
  - A encrypted to B: B's private key is used to decrypt
  - A signed to B: A's private key is used to sign
- The other (public) is used for encryption/verification
  - A encrypted to B: B's public key is used to encrypt
  - A signed to B: A's public key is used to verify the signature
- The public key is available to everyone
  - Problem: Association "Key  $\leftrightarrow$  Person" must be ensured

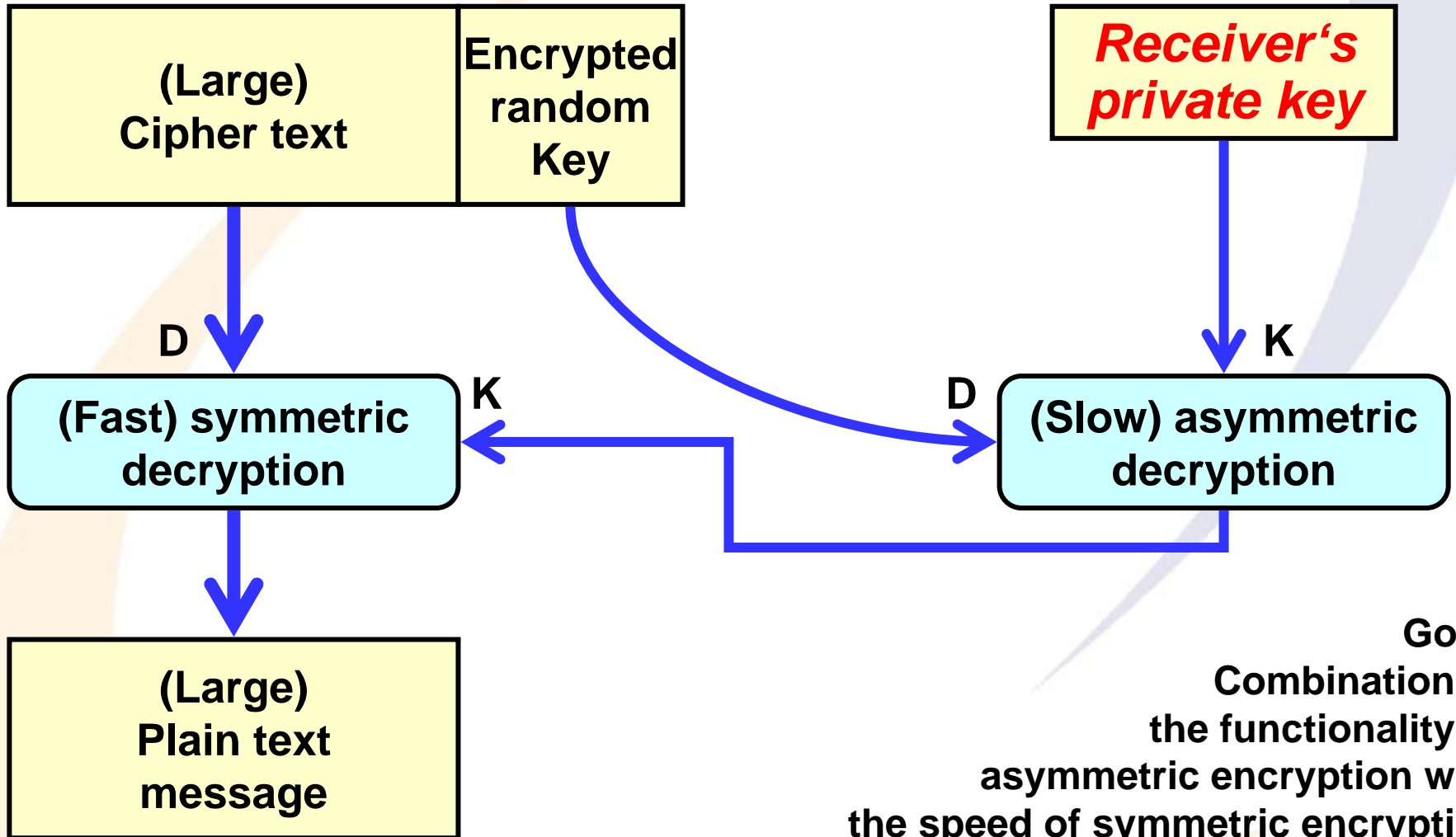
# Combining Symmetric & Asymmetric Encryption (1)



**Goal:**  
Combination of  
the functionality of  
asymmetric encryption with  
the speed of symmetric encryption



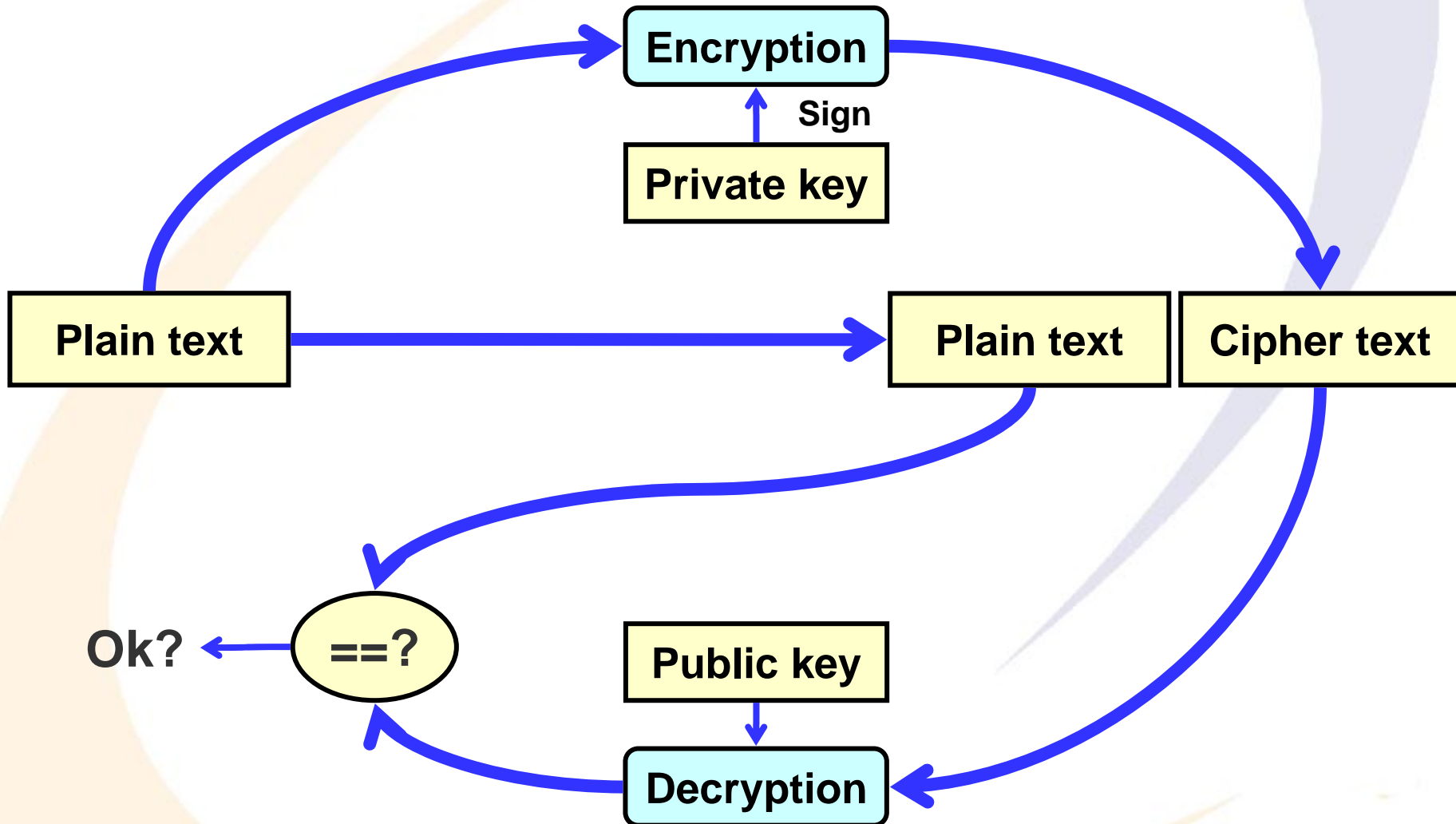
# Combining Symmetric & Asymmetric Encryption (2)



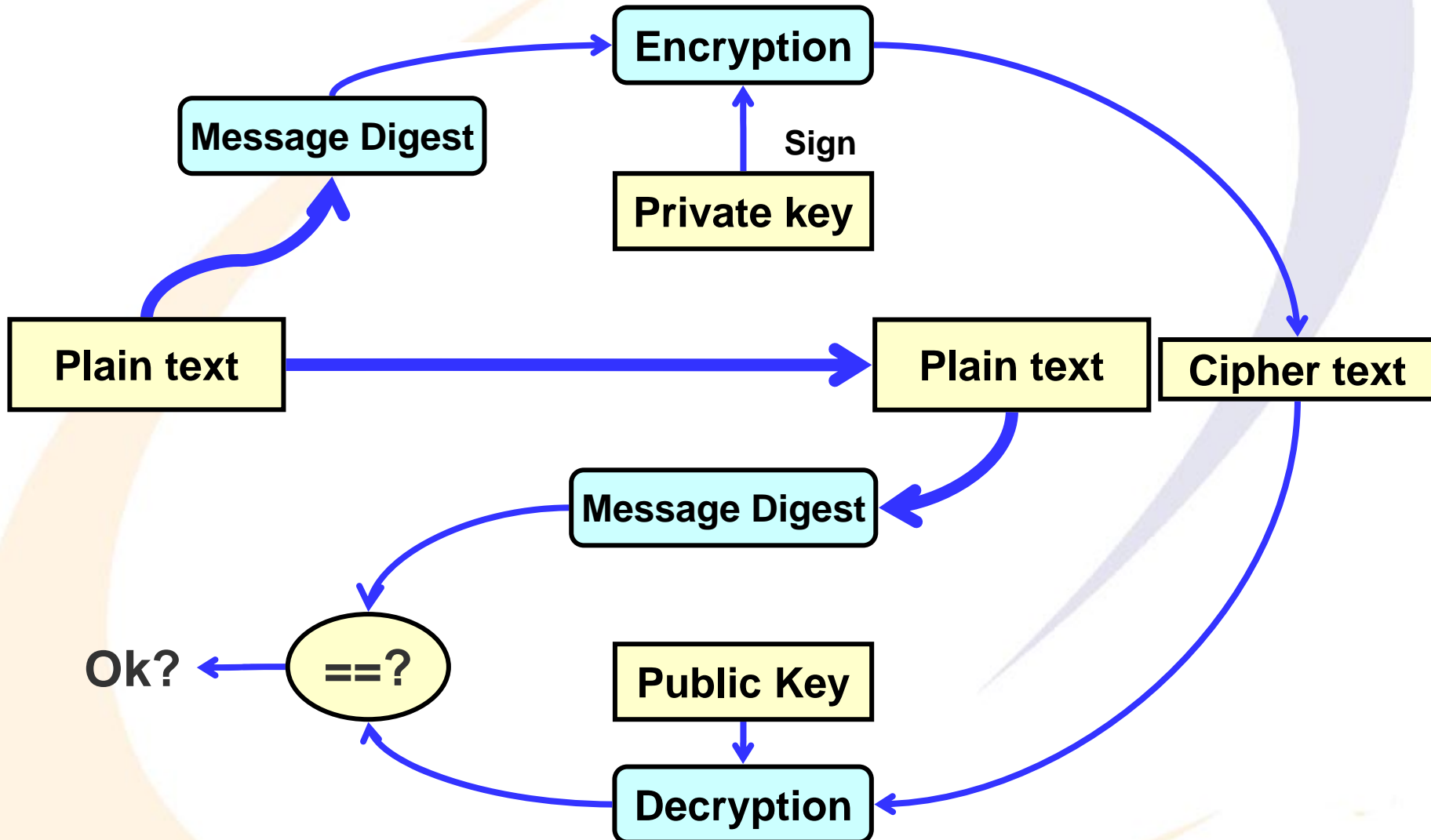
**Goal:**  
Combination of the functionality of asymmetric encryption with the speed of symmetric encryption

F I M

# Sign: Encryption with private key + ...



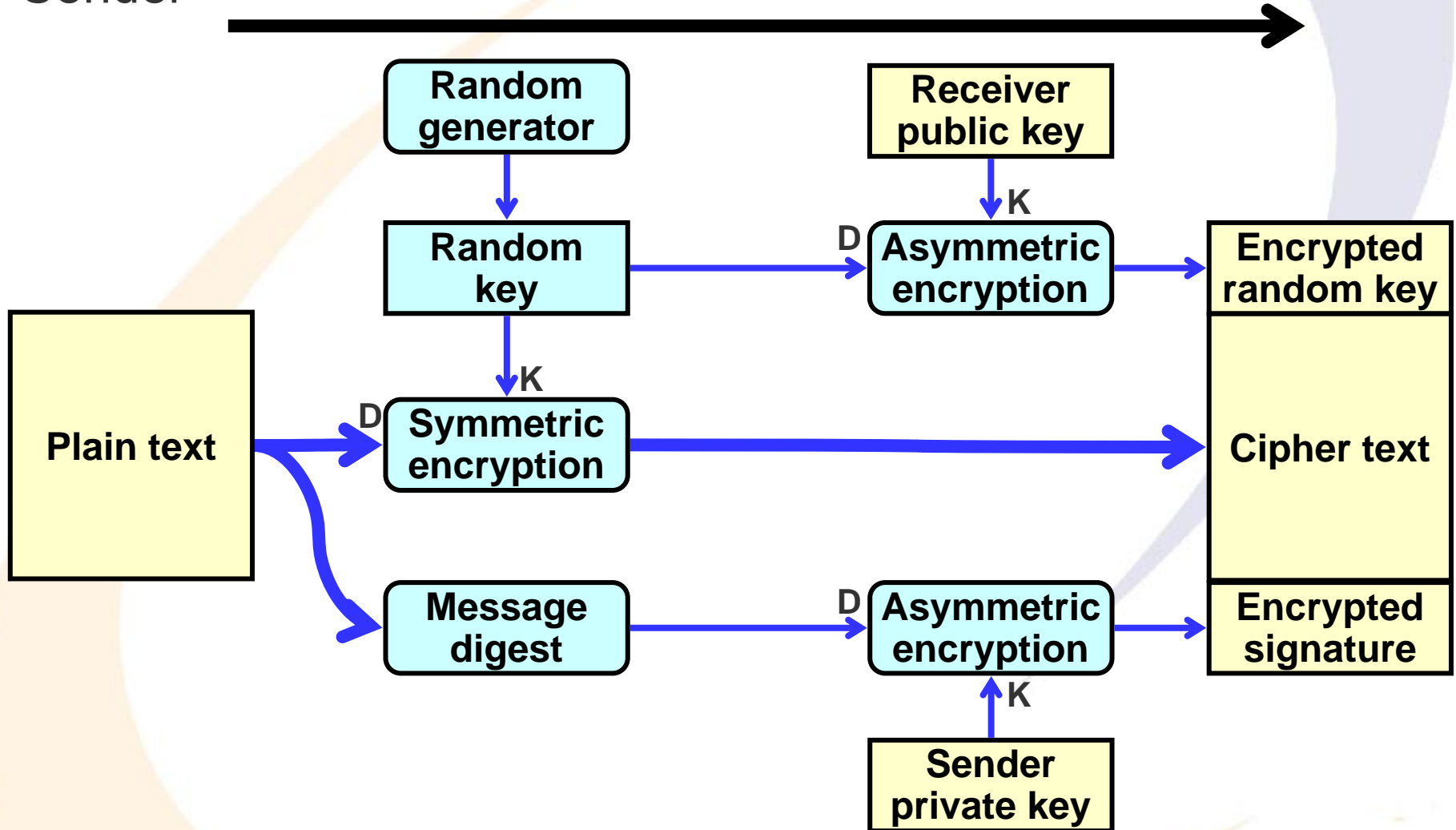
# Check: Decrypt with public key + compare

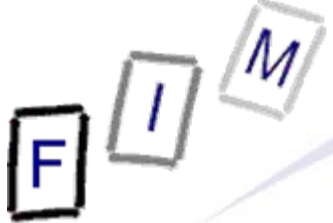




# Signature & Encryption

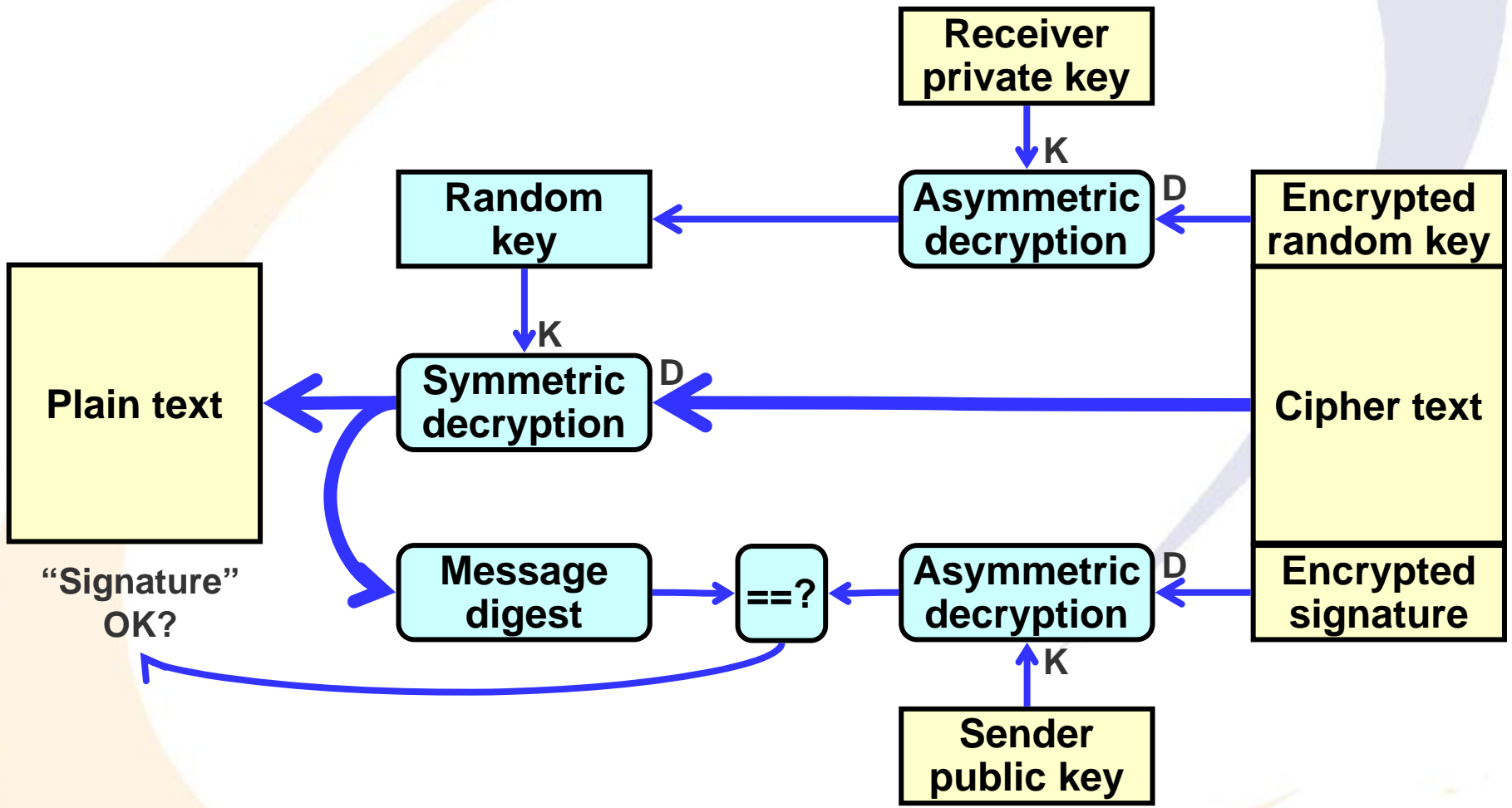
Sender





# Signature & Decryption

Receiver





- Symmetric:
  - DES/3DES: 56/112 Bit; DES is now insecure; 3DES sufficient for commercial use (frequent key changes recommended)
  - AES (=Rijndael): New "standard" algorithm; different key sizes
- Asymmetric:
  - RSA: Classic asymmetric cipher (rather slow)
    - » Arbitrary key size ( $\geq 1024$  recommended); no longer patented!
  - DSA: Only signatures supported, no encryption possible
- Hash:
  - SHA-1, RIPEMD-160: 160 Bit
    - » SHA-1: Deemed to be not quite secure anymore
  - MD5: 128 Bit (not recommended any more; insecure)
- Other:
  - Diffie-Hellman: Key agreement without previous knowledge
    - » Generates a shared secret key



# Strength of algorithms for the future

- The necessary key length is not static:
  - Faster computers
  - Advances in mathematics
  - New attacks (most dangerous of all!)
- Decision for length must incorporate:
  - Time/power required for en-/decryption
    - » See e.g. smartcards (computing and electrical power)!
  - Degree of security (=amount of money required for breaking)
  - Absolute time the calculated value should remain secure!
    - » Very often ignored!
    - » Guideline: For the next 20 years (values below: 2024)
      - Symmetric:  $\approx 89$  Bit
      - Asymmetric: RSA, ...:  $\approx 2113$  Bit; DSA:  $\approx 157$  Bit
      - Corresponds to a budget for an attack in 1 day of  $\approx 732$  million US\$

Source: Lenstra, A. K., Verheul, E. R.: Selecting Cryptographic Key Sizes. DuD 24 (2000), 166



# Key exchange: Diffie-Hellman

- Establishing a shared secret without prior knowledge over an insecure communication channel
  - Note: Vulnerable to man-in-the-middle attacks!
    - » Attacker establishes shared secrets with both parties independently → Some method of authentication is needed
- Incorporated in numerous protocols as a part
- Idea: One-way function (easy to compute, hard to invert)
  - Here: Given  $x^y \bmod p$ ,  $x$ ,  $p$  → Calculate  $y$ 
    - » Exponentiation: Simple; Discrete logarithm: Very hard
- A: Selects  $a$ ,  $p$ ,  $g$  ( $p$ =prime number)
- B: Select  $b$ , and obtains  $p$  and  $g$
- A → B:  $g^a \bmod p$       B → A:  $g^b \bmod p$
- A calculates:  $(g^b \bmod p)^a \bmod p = \text{SECRET}$
- B calculates:  $(g^a \bmod p)^b \bmod p = \text{SECRET}$



- Symmetric encryption algorithm with 56 Bits keysize
  - Now seen as insecure because of short key length
  - Specifically secure against a much later published attack
    - » Differential cryptanalysis → Was already known but kept secret!
  - Currently attacks with brute force possible
- Superseded by 3DES and AES
  - Triple-DES:  $\text{Encrypt}(\text{key1}, \text{Decrypt}(\text{key2}, \text{Encrypt}(\text{key3})))$ 
    - » Because of a special attack, security is only 112 Bits!
- Because of its design DES is fast to implement in hardware, but slow to implement in software
- Basic idea:
  - Permutations, Expansion, Substitution, Permutation
  - Encryption and decryption are identical (only: key reversed)



- Replacement for DES – Symmetric encryption
  - Key length: 128, 192 or 256 Bits
  - 192, 256 Bits are allowed for US classification "Top secret"
- Fast to implement in hard- and software
  - Now widely used in various devices and software
- No known weaknesses of the algorithm
  - Note: Implementations may still suffer from problems
    - » Example: Side channel attacks, like timing encryption steps, measuring power usage, ...
- Basic idea:
  - Expansion, Substitution, Transposition, Mixing



- First public-key algorithm suitable for signing and encryption
  - Still secure, if long enough keys are used!
- Basic idea:
  - Choose two very large prime numbers and multiply them
    - » Factorization of this number is very hard
  - From these numbers a private and a public key are derived
- Some mathematical weaknesses exist:
  - Use a random padding, so each encryption of the same text produces different output
  - The same key should not be used for signing and encryption
  - Prime numbers are usually only checked probabilistically
  - Good random number generators are needed
- Key length: 1024 Bit might be broken in the near future
  - Recommendation: Use 2048 Bit keylength



- Public key algorithm useable **only** for signatures
- Based on the same method as Diffie-Hellman
  - Exponentiation modulo  $p$
  - A variant exists: Elliptic Curve DSA
- Requires a secure hash function and a good random number generator
- Basic idea:
  - Choose public  $p, q, g$  (with certain mathematical relations)
  - Select  $x$  by random ( $0 < x < q$ )
  - Calculate  $y = g^x \bmod p$
  - Public key:  $(p, q, g, y)$
  - Private key:  $(x)$
- Signing: Involves the hash function and a new random value



- MD5 = Message Digest algorithm 5
  - Hash length: 128 Bit
    - » Typically: 32 characters (hexadecimal encoding)
  - Very widely used
- Attention: Several flaws are known!
  - Collisions (= 2 texts with identical hash) can be constructed
  - Interesting: MD5 is used in many webpages → Google can be used as a lookup tool!
- Often used for storing password: "store MD5(pwd)"
  - Use at least "salting"!
  - Or: "Better" algorithms, i.e. requiring more time to calculate
- Function:
  - Cut message up into 512 Byte blocks (use padding for last)
  - Based on addition, shifting, non-linear function



- SHA-1 = Secure Hash Algorithm
  - SHA-1: 160 Bits
  - SHA-2: 224, 256, 384 or 512 Bits (Individually: SHA-224, ...)
- Not completely secure any more!
  - No known attacks, but mathematics has proved weaknesses
  - Original strength: 80 Bit (birthday paradox)
    - » = 1/2 length; maximum value possible →  $2^{80}$  tries for a collision
  - Current state:  $2^{63}$  tries
    - » Just barely useful for very-large-budget organizations (NSA)
- Function: Similar to MD5, but with different configuration
- SHA-2 uses different algorithms
  - No known weaknesses, but not much investigation either!



# Environmental components

- Encryption algorithms are not **all** there is, to be secure
- Many other elements must be taken care of:
  - **Technical "surroundings":**
    - » Secure viewer: Showing exactly the content to sign and not something different
    - » Secure transmission of codes/PINs from chipcards/terminals to the CPU actually calculating the signatures or the hashcode in reverse when signing takes place on the card/terminal
    - » Physical access control/restrictions?
    - » Side channels: Power, temperature, timing, cache access, ...
  - **Organizational issues:**
    - » Who knows the encryption keys and where are these stored?
    - » How to get at them in case of illness/dismissal?
    - » Who is allowed to do what? Does the equipment support these different security levels?
    - » Securing keys/certificates etc. against loss



- Public keys must be connected to a certain individual/device
  - Everyone can use/create a key, but how do you know that the person holding the private key is actually "Donald Duck" (or a certain person using this pseudonym)?
  - "Someone" guarantees, that these two belong together

## **Certificates connect a public key to a name**

- Certificates may contain other information
  - E.g. server certificates may contain the administrator's E-Mail
  - Personal certificates may contain restrictions or special permissions/empowerments
    - » May act on behalf of a company, transaction restrictions etc.
- Certificates are signed too, so nobody can tamper with them
  - Chicken-egg problem: Who signed the certificate?
    - » Pre-shared "master" certificate/Public Key Infrastructure (PKI)



# Certificate content

- Currently only certificates of type X.509 are of importance
  - Several versions available; current one: 3 → X509v3
  - Standard is not too clear
    - » Certificates from one vendor might be incompatible with those from another vendor or with some software
    - » Special problem: What data, which form, which "schema"
- No problem:
  - Public key including algorithm
  - Issuer: Who "guarantees" for the association key ↔ name
  - Version, serial number, validity, unique IDs
- Problems:
  - Subject (=associated name): Different elements (E-Mail, additional/missing parts, ...)
  - Extensions: Key usage, CRL distribution, constraints, etc.

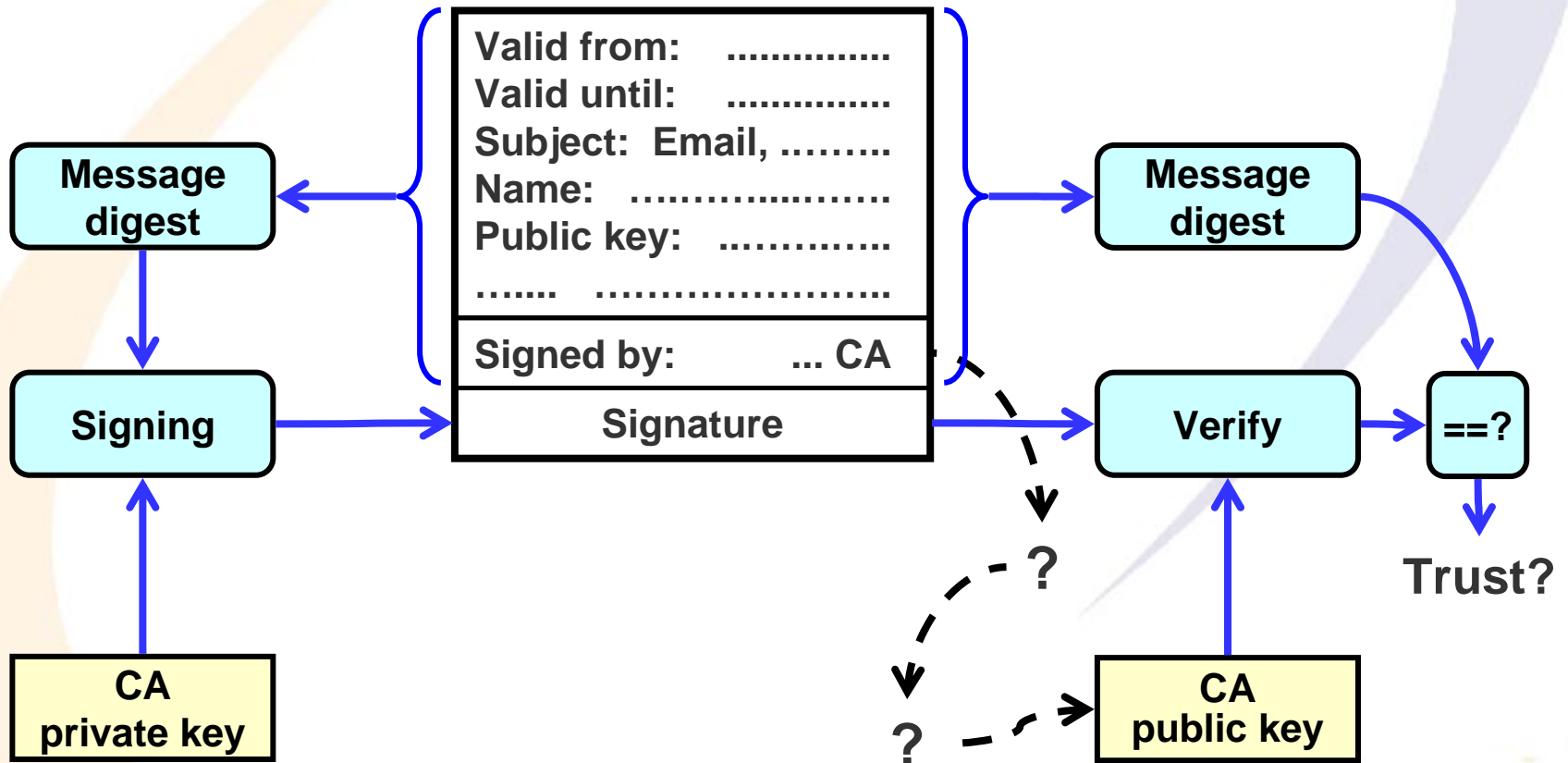


# Certificate

## Certification Authority "CA"

## Certificate

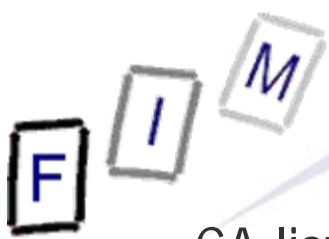
## Verification of the certificate





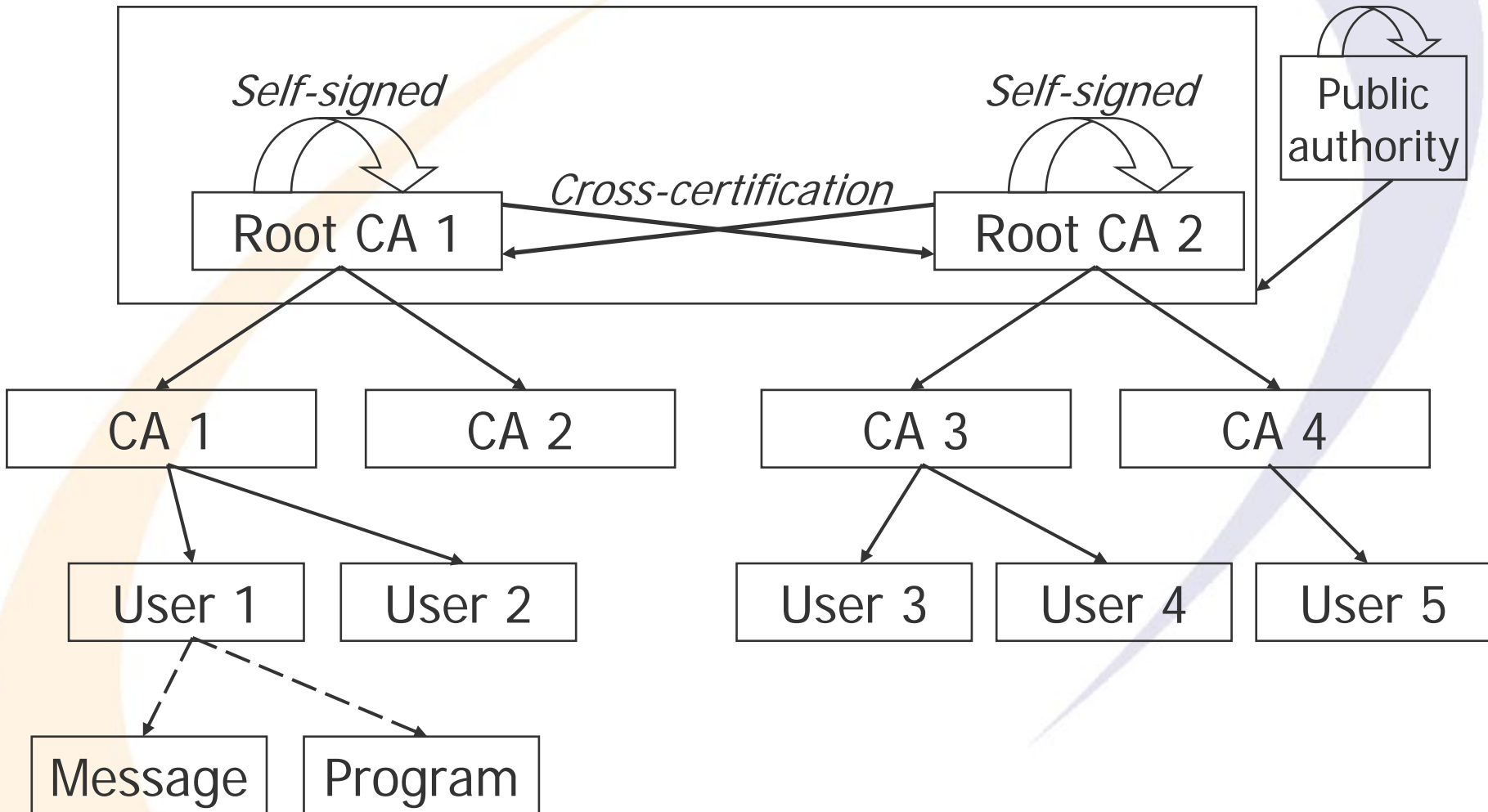
# Public Key Infrastructure (PKI)

- Who guarantees, that the certificate is "correct"?
  - » I.e. that the key belongs to exactly this person and that this person was securely identify (and not some impostor)
  - The issuer through his signature of the certificate
  - Who guarantees, that this signature is "correct"?
    - » ...
    - ...
- Solution: The "top-level" certificate is self-signed
  - Key used for signing is the one for the public key contained
  - This certificate you "just have to trust"
    - » Obtained from a secure source, verified (e.g. fingerprint), ...
  - Can also be "cross-certified": One top-level certificate is used to sign another top-level certificate and in reverse
    - » Good for few CAs (otherwise:  $O(N^2)$ !)



# PKI Example

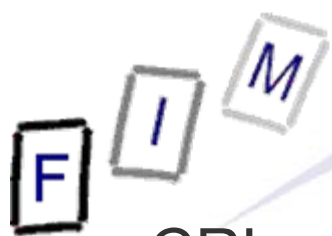
CA list





# Certificate revocation

- Sometimes certificates must be "removed", e.g. when
  - some attributes are incorrect (name/profession changes)
  - private key is disclosed
  - algorithm is now insecure
  - no longer used (e.g. server certificates)
- although they are still valid (looked at them alone)!
- Solution: Revocation lists
  - Must (should) be consulted on each verification of a signature
  - Must happen fast e.g. on lost signature cards
    - » In the meantime someone else could sign "for you"!
  - Contains a timestamp
    - » Signatures before the revocation must remain valid indefinitely
- Biggest problem: This requires continuous online connection!
  - Every transaction must check the revocation status for the very moment it is made (→ DoS, Security and Privacy: Introduction to Cryptography)



# Certificate revocation: OCSP

- CRLs are lists, which continuously become longer
  - Distribution/lookup is therefore problematic
- Online Certificate Status Protocol makes this easier!
  - Note: The basic problem, online access required, remains!
- Security issues:
  - The status request reveals an interest in a specific person
    - » At least to CA; depending on request encryption also publicly!
  - Where to get the OCSP URL from?
    - » Typically included in the certificate → Check first against root!
- Basic idea:
  - Send certificate to CA (name, key, serial number, ...)
  - CA checks list and generates response
  - CA signs response and sends it back
  - Client checks signature and retrieves result
- Support: IE 7 (>=Vista only!), Firefox



# Certificates and digital signatures

- Creating/Verifying a digital signature:
  - Encrypt values (see below) with private key
  - Send document and/or encrypted value to recipient
  - Recipient obtains certificate of signer (however) and checks it
    - » Certificate chain, root certificate, revocation, expiry date, etc.
  - Recipient decrypts value with public part of key and checks it
- Two kinds of signatures possible
  - "Internal": The document is "encrypted" with the private key
    - » Verification=Decryption; reading the document takes long
      - "Avalanche property" of good (!) algorithms: Minimal modifications lead to complete gibberish on decryption
  - "External": A hash value is calculated and then signed
    - » Verification=Comparing the decrypted hash with the (newly) calculated one from the plaintext document; quite fast
      - Possible problem: Finding a similar text with same hash value
      - Quality of hash algorithm is therefore very important here!

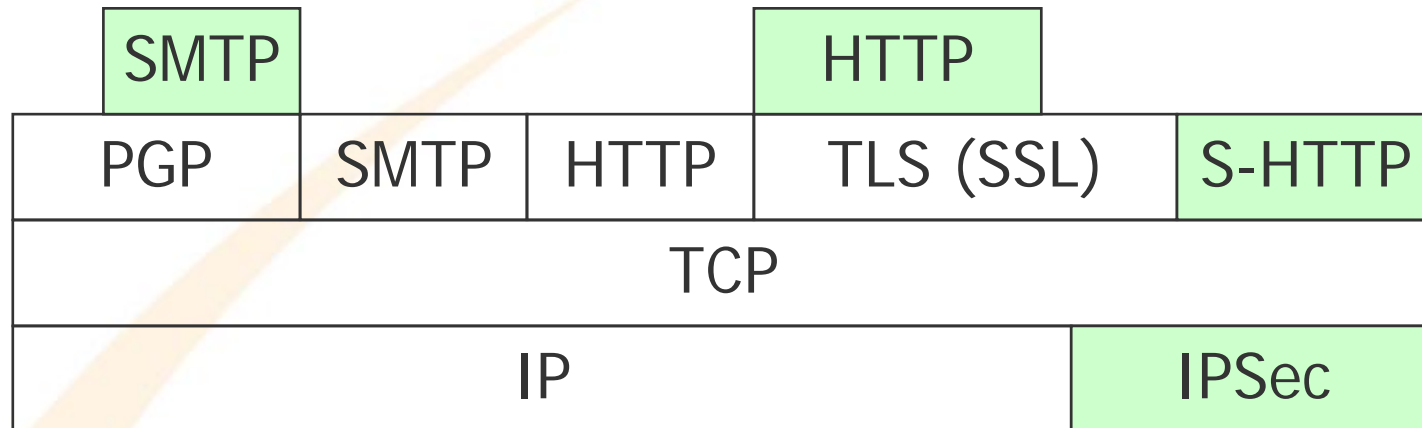


# Encryption for the WWW

- When transmitting sensitive information on the Web, the communication should be encrypted
  - Examples: Credit card numbers, company-internal forms, ...
- Currently one method is widely used: TLS
  - Secure Socket Layer: A general solution for encrypted TCP traffic; most common use with http ( $\Rightarrow$ https; NOT: shttp)
    - » Option available to use http and switch internally to TLS!
  - TLS (Transport Layer Security): SSL successor, very similar
- TLS provides:
  - Encrypted communication: Eavesdropping impossible
    - » Depends on the actual algorithm/keylength used
    - » Uses symmetric cryptography for speed; numbers against replay
    - » Asymmetric cryptography used for key exchange
  - (Mutual) authentication by asym. cryptography supported
  - Configuration very important (algorithms, cert. storage, ...)



# Security for the WWW



- PGP: Pretty Good Privacy
- TLS (SSL): Transport Layer Security (Secure Socket Layer)
  - The whole communication is secured
- S-HTTP: HTTP + security extensions
  - Single messages are secured
  - HTTPS: HTTP over TLS
- IPSec: IP Security
  - Every communication is encrypted and/or authenticated



# Authentication modes

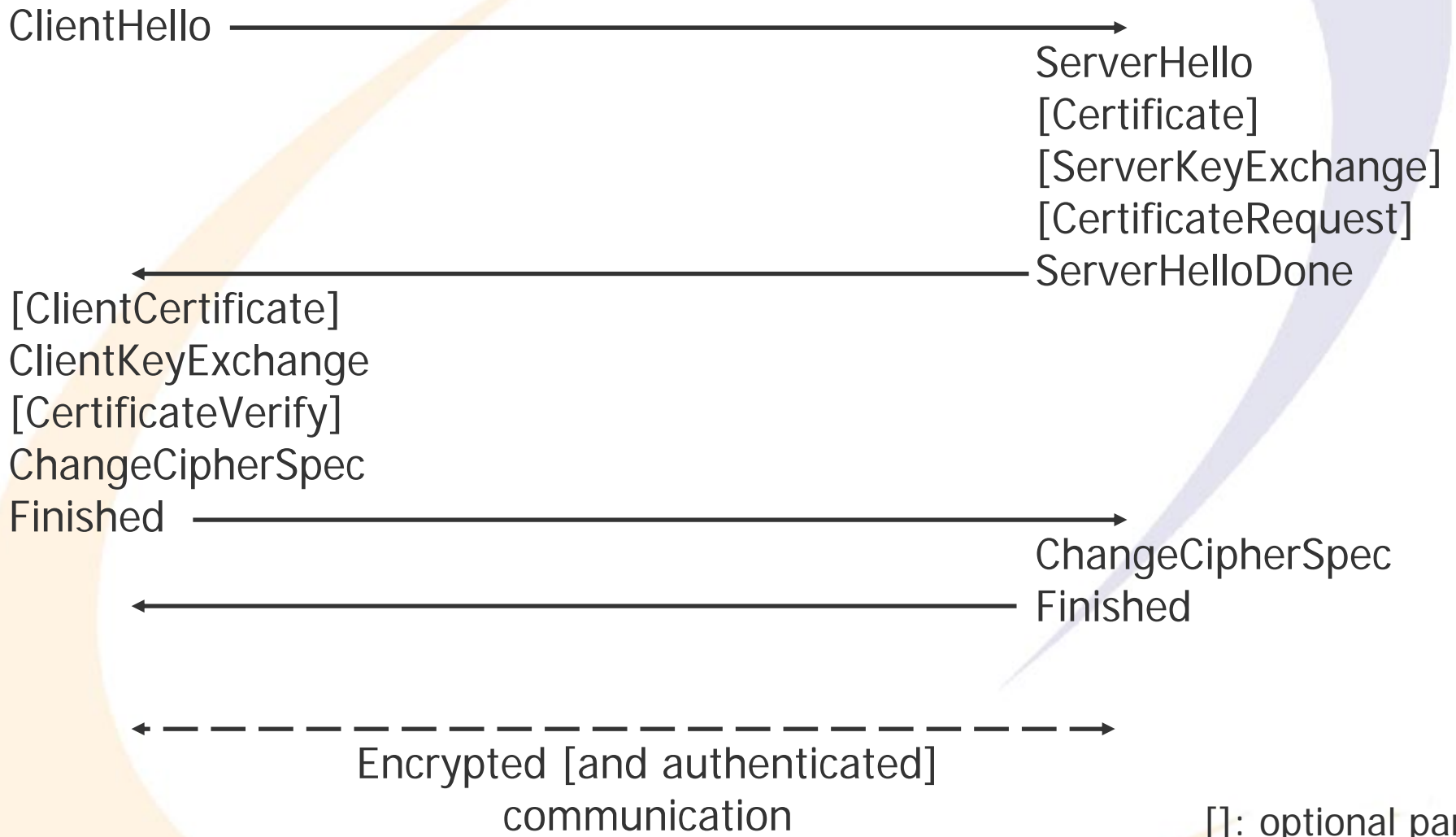
- Either the server alone, or both server and client can be authenticated; but never the client alone
  - For the WWW this means, authenticating **only** the web browser is not possible!
  - Commonly the server alone is authenticated
    - » Client authentication only in closed systems
- Authentication requires a certificate
  - Most browsers come with a list of top-level CA certificates
  - Unknown certificates can be imported or accepted ad-hoc
    - » Large part of CA business is based on this: No questions!
  - For smaller companies: Create their own certificate and distribute it to partners
    - » For public: Present it on website (but is this really secure?)
  - Webserver must have access to private key: Must be secured very well within the system!



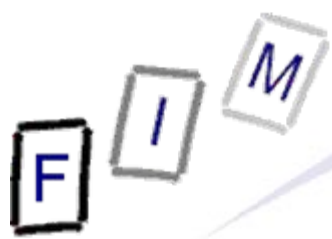
# TLS: The protocol (1)

Client

Server



[ ]: optional part



## TLS: The protocol (2)

- Client-/ServerHello: Contains a random number and encryption/compression capabilities
  - Random number: Prevents replay attacks
- Certificate: Server certificate including chain to top-level CA
- ServerKeyExchange: If the server has no certificate or it cannot be used for encryption
  - Commonly uses Diffie-Hellman Key Exchange protocol
  - Signed by certificate to avoid man-in-the-middle attacks
- CertificateRequest: Non-anonymous server can request a client certificate
  - Contains list of certificate types understood
  - Contains list of DNs of acceptable CAs
    - » DN = Distinguished Name; format for name in X.509 certificates



## TLS: The protocol (3)

- ServerHelloDone: Hand-off to tell client: that this is all
- ClientCertificate: Certificate of the client or warning that no (matching) one is available
  - Server can accept without certificate or terminate protocol
- ClientKeyExchange: Client part of key exchange protocol
  - Always required!
- CertificateVerify: Signed digest of messages
  - To prove the knowledge of the private key for the certificate
- Finished: Encrypted & signed with (new) negotiated values
  - Content may be sent immediately (no wait for reply required)
- ChangeCipherSpec: Switch to encryption
  - This message is still handled according to the old algorithms!
    - » At the beginning this means, it is sent unencrypted



## What you (don't) get!

- Server (=counterpart) is the one specified in the certificate
  - Not necessarily the actual webserver; this is verified by the browser, however!
    - » Difficulties for servers with different domain names, as in the TLS handshake there is no place for the hostname (as is in http!)
      - Virtual hosts: Separate and matching certificate should be provided
- Client knows private key for its own certificate (if provided)
- Certificate revocation was checked
  - Depends on the browser; not in protocol itself!
- Encryption, authentication, integrity, non-repudiation, no manipulation, no replay
- What you don't get:
  - Additional certificate content (e.g. attributes) often ignored
  - Hiding who talks to whom

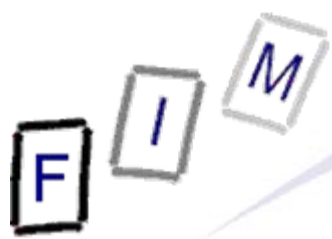


# Alternatives: Pre-shared keys

- Only suitable for very small group of partners communicating
  - See VPN later; especially VPN tunnels!
- Keys must be exchanged over a trusted channel
  - I.e. **NOT** over the channel used for communicating!
- Protocols must use "Challenge-Response": The key may never be sent in clear!
  - » Before you don't know who is on the other side ...
  - Common way: Random value sent, hashed with secret key, sent back, compared to expected response
    - » No eavesdropper/man-in-the-middle can retrieve the key from it
- Not possible with SSL or TLS!
- Advantage: Usually very simple to manage
  - Agree on a keyphrase, telephone call ⇒ works!
    - » No additional infrastructure needed (PKI, CRL, etc.)



- Similar to PKI, but distributed model
  - Signing someone other's keys to certify, that the association is correct; diverse servers for storing keys and signatures
- Based on transitivity of trust (=the signatures):
  - A trusts B, B trusts C, C trusts D  $\Rightarrow$  A trusts D
- Not possible with SSL!
  - Uses different certificate format
  - Currently mainly used for E-Mails
- Advantage: No single point of failure
- Problem: No guaranteed decision
  - Perhaps just no trusted connection exists; still valid & correct!
  - CA's are possible, but not necessary
  - System reliable only, if keys signed by many people
  - Such people are not found easily everywhere



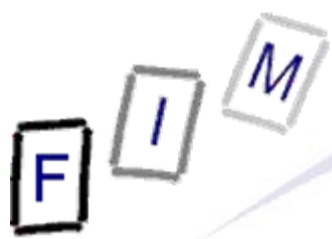
# "Official" certificates: Advantages / Disadvantages

---

- + Identity of person/company verified accurately
  - » More trust than a self-signed certificate
- + No warning messages for browsers
- + Interoperability with many browsers
  - » Creating a "good" certificate is not easy!
- + Key length issues, etc. are taken care of
- + Provides reliable directory servers and CRL/OCSP services
- Costs money (and expires regularly, requiring a new one!)
- May take some time to obtain (depending on CA/location)
- Guarantees for **content** are small or non-existing
- Result:
  - Public website: Indispensable (browser warning)
  - Private/internal use: Very few reasons
    - » Except: Large companies, where managing secure and available directories and CRLs are difficult!



- VPN = Virtual Private Network
  - A private network across a public medium
  - Replacement of leased lines by encrypted/authenticated communication using the "ordinary" and common internet
    - » In the generic case, it can also be any kind of other communication system, but the internet is by far the most important one!
- Especially important for mobile workers
  - Always "virtually" located in the home network
    - » Telephone (VoIP): Same number same functionality, ...
    - » Server access: E-Mail, file servers etc.
    - » Internal applications available
  - Can move from place to place freely
- Other application: Branch offices
  - The internet serves as the company backbone



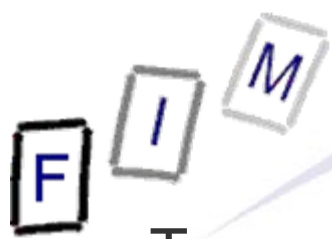
# VPNs: Advantages

- Transparent for users (apart from establishing perhaps)
  - User virtually sits on the other end of the tunnel
- Obviates the need for a firewall
  - Everything is encrypted and authenticated
    - » Filtering would be impossible anyway
  - But does **NOT** secure against "internal" attacks
    - » Internet is protected against, Intranet must be secure itself!
      - Especially important for mobile workers: The laptop is virtually inside the company, but may have been connected also to other networks and is therefore possible infected, insecure, ...
  - Does **NOT** apply in "split" configurations
    - » Some traffic is sent through the tunnel (e.g. file server access)
    - » Some traffic is sent to the Internet directly (e.g. webbrowser)
  - Practice: VPN connections are in a kind of "DMZ"
- Easy to set up if basic configuration exists (i.e. 2nd, 3rd, ...)



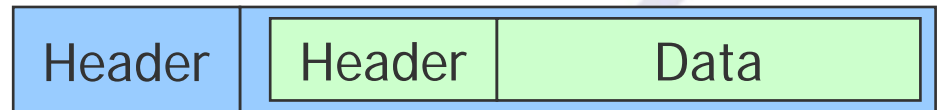
# VPNs: Problems

- Traffic can no longer be compressed
  - Must happen before or at the tunnel endpoint
    - » Modern devices support this
- No QoS (as often available with leased lines)
  - The Internet only does what it can
- Sometimes difficult to set up; interoperability difficult
  - Becomes better with IPSec
  - Much easier with TLS-based VPNs
- Powerful hardware needed for encrypting larger bandwidth
  - Dedicated devices, "VPN concentrators", ...
- Overhead; more bandwidth required
  - This is today usually only a small problem!
- Data is physically outside: Not necessarily secured as well!



- Transport of packets from one protocol over another one
  - » Generic basis for VPNs!
  - Done by "packaging" the original packets of the main protocol into new packets of the outer protocol
  - Transparent to upper layers: No application modifications!
    - » See TLS: This is also a kind of tunneling!
  - Can also be used identically, i.e. packing IP into IP
    - » This is used e.g. by IPSec (+ additional information)
  - Reasons: Not supported (e.g. IPX), unroutable (NetBUI), illegal (private) addresses (192.168.?.?), ...

- Source: Encapsulation



- Adding a new header (and perhaps a new trailer)
- Destination: Extraction
  - Interpreting and removing header (+trailer)
  - Passes the content on in some locally defined manner

- Point to Point Tunneling Protocol (version of PPP)
  - Supports IP, IPX, NetBUI
- Client-Server-Model
- Rather easy to set up (and client is integrated into windows)
- Can be transported across NAT (with additional software, ...)
- Client authentication by username/password
  - Several old and very insecure algorithms/protocols exist!
  - Server is not authenticated in most implementations!
- Encryption of content only optional
- No key management protocols
  - Key remains the same for the whole communication!
- No integrity check for packets



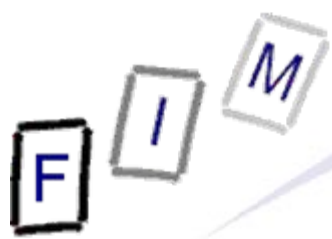
- IP Security Protocol: Intended for IPv6 (where it is mandatory!), but used also for VPN's in IPv4
  - No full layer 3 support: No multicasts, static routing only
    - » Static routing: No dynamically "redirecting" the tunnel itself
      - The encapsulated packets can be routed in any way and method!
- Allows, and implementations support, a multitude of authentication, encryption, hash, and compression protocols
  - When an algorithm is broken, just configure a different one!
- Mutual authentication of packets and endpoints
- Key exchange protocol
  - New key for each tunnel and regularly changing keys
- Encryption of complete content
- Supports IP only
  - Tunnelling other protocols via IP and then IPSec possible



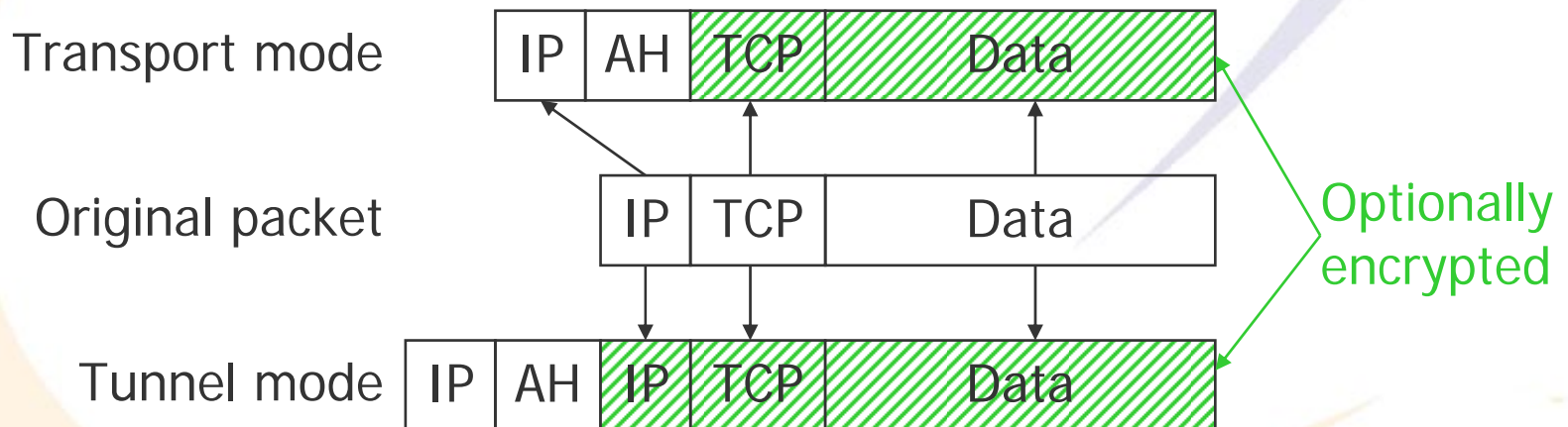
# IPSec vs. PPTP

- PPTP is easier to set up
  - Username and password; no certificates, CAs, CRLs needed
    - » IPSec also supports Pre-Shared-Keys; wizards for setup sometimes available (depending on vendor)
- IPSec is much more secure
  - Keys exchanged during usage, more secure algorithms
- PPTP can go over NAT
  - This might be good or bad, however!
  - IPSec only with additional option ("NAT traversal")
    - » But this is already supported by all modern soft-/hardware!
- IPSec implementations have fewer weaknesses
  - Microsoft PPTP implementation has (still) many weaknesses
  - But: IPSec in general is very complicated → Error-prone!
- IPSec supports IP protocol only

**When possible, use IPSec!**

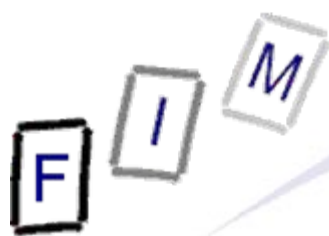


- Transport mode: Only data is encrypted
  - Header remains publicly visible!
  - Additional small header added
  - Used for secure connections (Host-Host communication)
    - » Rather rarely used
- Tunnel mode: Complete packet is encrypted
  - Completely new IP header added (in addition to ESP header)
  - Used for VPNs (LAN-LAN tunnel)





- AH protocol: Authentication Header
  - Cryptographic checksum over packet
    - » No modification on transport, identified peer was sender
  - Includes the complete header ⇒ NAT impossible
- ESP protocol: Encapsulation Security Payload
  - Encryption of whole packet
  - DES, MD5, SHA must be supported, anything else can be
- IPComp: Compression protocol
  - To be used optionally before encryption
- IKE: Internet Key Exchange
  - » Optional protocol (⇒ manual configuration otherwise)
  - Agreeing on a shared secret for authentication/encryption
  - Uses e.g. Diffie-Hellman or master keys



# IPSec limitations

- IPSec alone does not work with dynamic IPs
  - One fixed and one dynamic is still possible, as long as the dynamic side is the initiator
  - If both sides have dynamic IPs, DynDNS (and software support) is necessary
    - » IPSec works on the level of IP, therefore it only understands IP addresses; Name → IP address resolution must be external!
- No NAT: Use IPSec "afterwards", e.g. on router appliance
  - Or directly on the same router (first NAT, then IPSec)
    - » But then its probably better to use an IPSec LAN-LAN tunnel!
  - Alternative: Both sides support NAT-T
- Complex: Small errors might lead to a working solution, but reduce security significantly
- Interoperability sometimes lacking, but improving
- Debugging is difficult: Everything is encrypted, ...!



- A VPN based on TLS
  - Different from many "SSL VPNs", which are often ALGs
    - » ALG = Application Level Gateway
  - Because of TLS, a PKI is needed
    - » Certificates for the server and all clients needed!
      - But server need know only root-CA, not every single client certificate
- Simpler alternative to IPSec: Simplified security architecture
- Still requires installation of software on both ends
  - SSL VPNs: Access through every webbrowser, i.e. also from insecure machines!
- No problem with NAT
  - If you can connect with binary data, TLS is possible as well
    - » Potential problem with protocol inspection firewalls!
- Typically a software solution; no/few hardware available
  - But dedicated SSL hardware might help



## IPSec vs. OpenVPN

- IPSec typically requires kernel modifications; OpenVPN is a user-space program posing as a kind of network adapter
  - Security breaches are less severe in OpenVPN
  - OpenVPN is less efficient than IPSec
- OpenVPN has less configuration options, therefore creating an insecure configuration should be more difficult
- Hardware support for IPSec readily available (e.g. routers), not so for OpenVPN
- OpenVPN also supports dual-factor authentication
  - E.g. token + password; but then configuration and interoperability becomes difficult again!
    - » Of interest only for individual remote access, not LAN-LAN VPNs
- IPSec is a standard, OpenVPN a product
  - OpenVPN is available for most operating systems, but only from a single source



- Using VPNs, SSL, digital signatures is nice (and necessary!), but does not solve all problems:
  - Denial of Service
  - Endpoint security (storing those credit card numbers)
  - Users: Security is cumbersome and therefore circumvented
  - Cryptography is only as secure as the key storage
    - » Who uses really good passwords/passphrases?
    - » How is the "backup" of the password organized (bank safe)?
  - Physical security? Social engineering? Internal attacks?
- But security is also not self-serving:
  - Value of goods to be secured vs. cost of protection

**A holistic view is required for encompassing security!**

F I M

# Questions?

Thank you for your attention!