Mag. iur. Dr. techn. Michael Sonntag

Introduction to Cryptography

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

E-Mail: sonntag@fim.uni-linz.ac.at
http://www.fim.uni-linz.ac.at/staff/sonntag.htm

Thanks to Rudolf Hörmanseder for some slides (esp. drawings!)

Introduction

- 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
- XML Signature/Encryption

Why cryptography?

- Security is a very important aspect, especially if money (or equivalents) are affected by transactions
- Not every 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")

Motivation

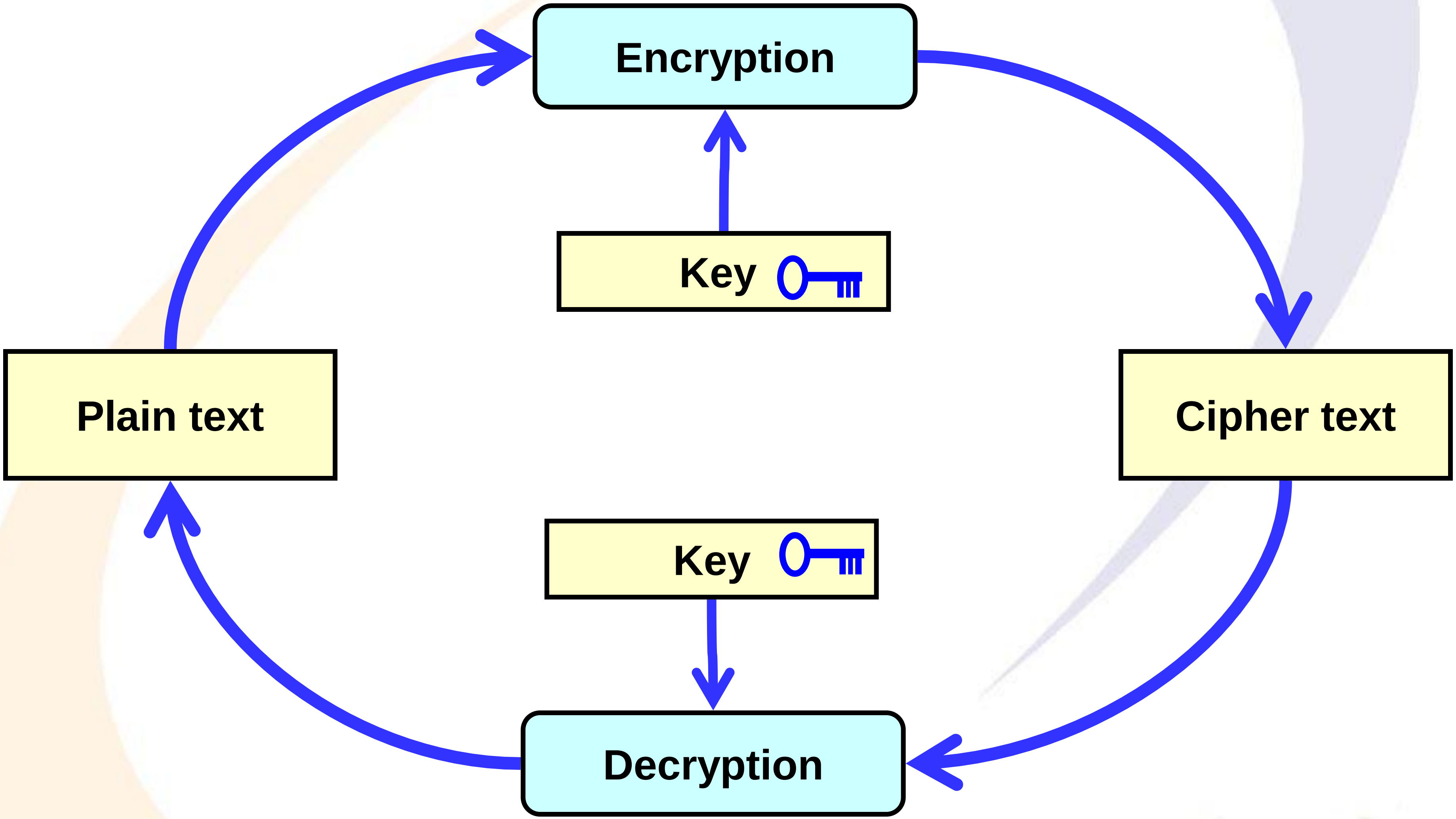
- Data is being
  - → transmitted
  - → stored
  - → processed
- and exposed to the following attacks:
  - → Inspection: without / <u>with</u> understanding
  - → Modification: without / <u>with</u> understanding
  - → Deletion: random / <u>targeted</u>
  - → Addition: random / <u>targeted</u>
  - →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

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

Software components

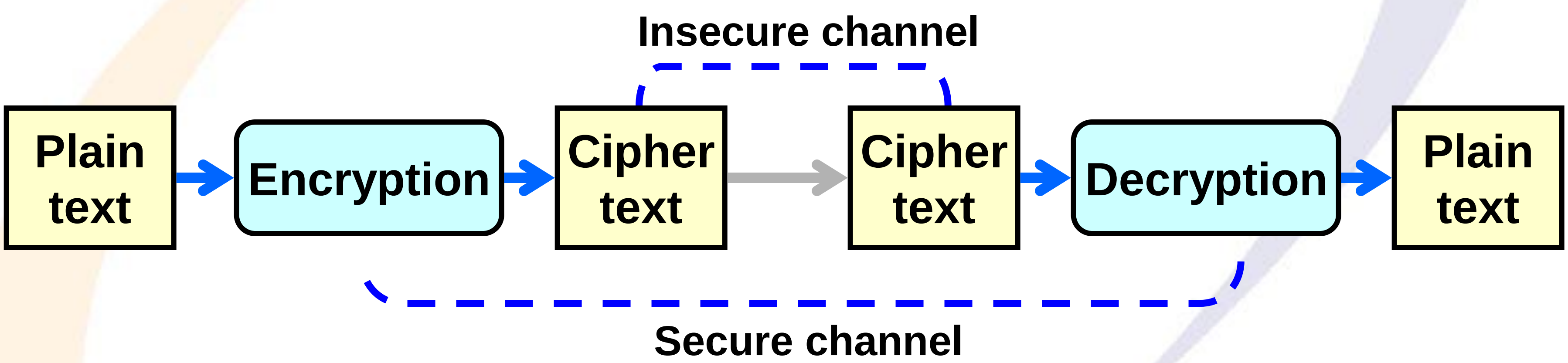
- Several different classes of algorithms required:
  - → Hash functions: Handling the whole document takes too long
    - » Drawback: Content could be substituted (collisions)!
  - → Encryption/Decryption: The same algorithm for symmetric, but different one for asymmetric encryption/signatures
    - » Encryption: Combining a document with a public key
      - – Decryption: Combing encrypted document with private key
    - » Signature: Combining a document with a private key
      - – Verification: Checking the 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 many algorithms exist: Good & bad

Basic functionality cryptography



Common requirements

- Key comparatively very small
- Data to encrypt very large
  - → In comparison to the other; not necessarily absolute

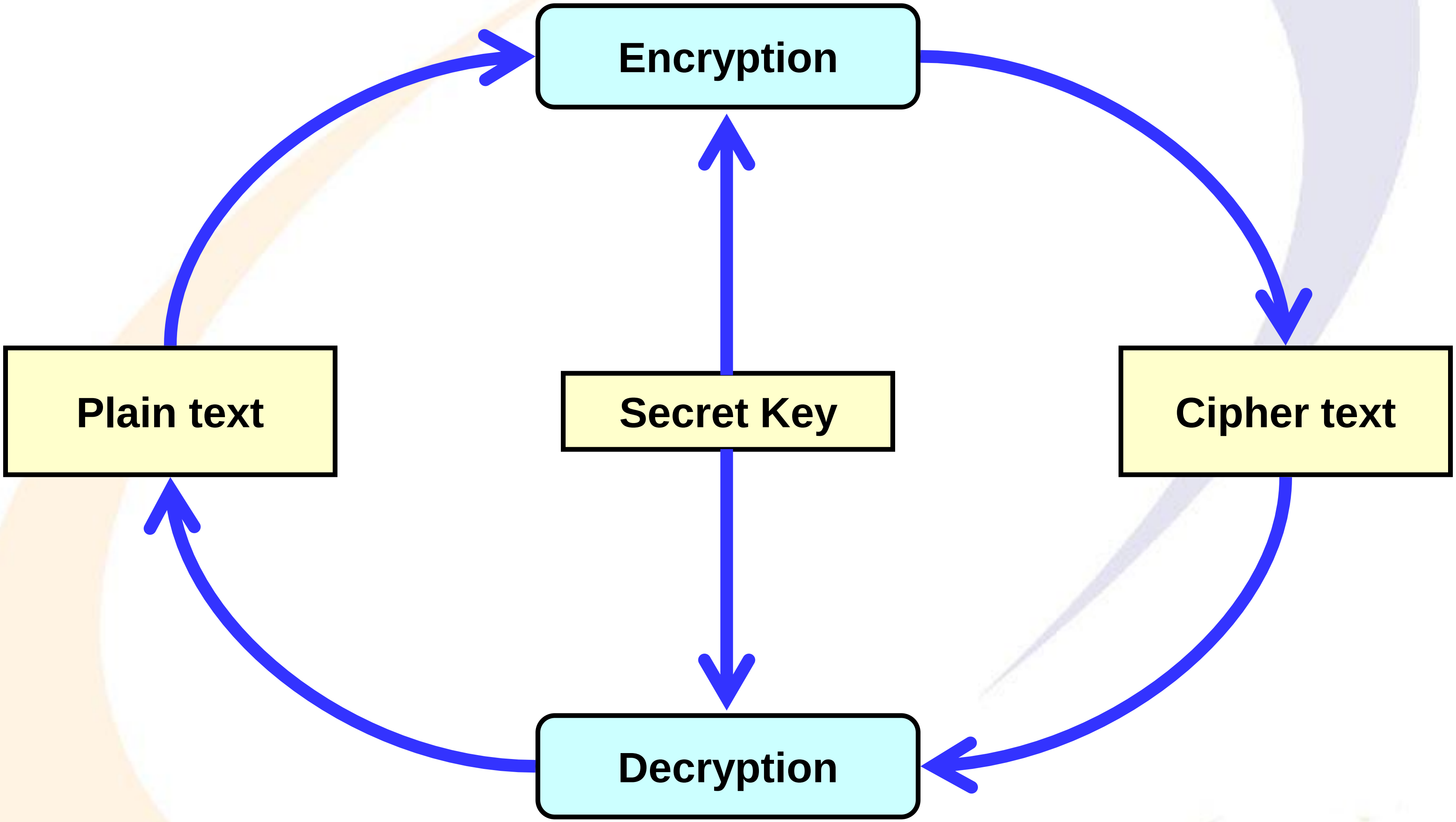


Gutmann P.: „Network Security: Security Requirements“, University of Auckland, http://www.cs.auckland.ac.nz/~pgut001

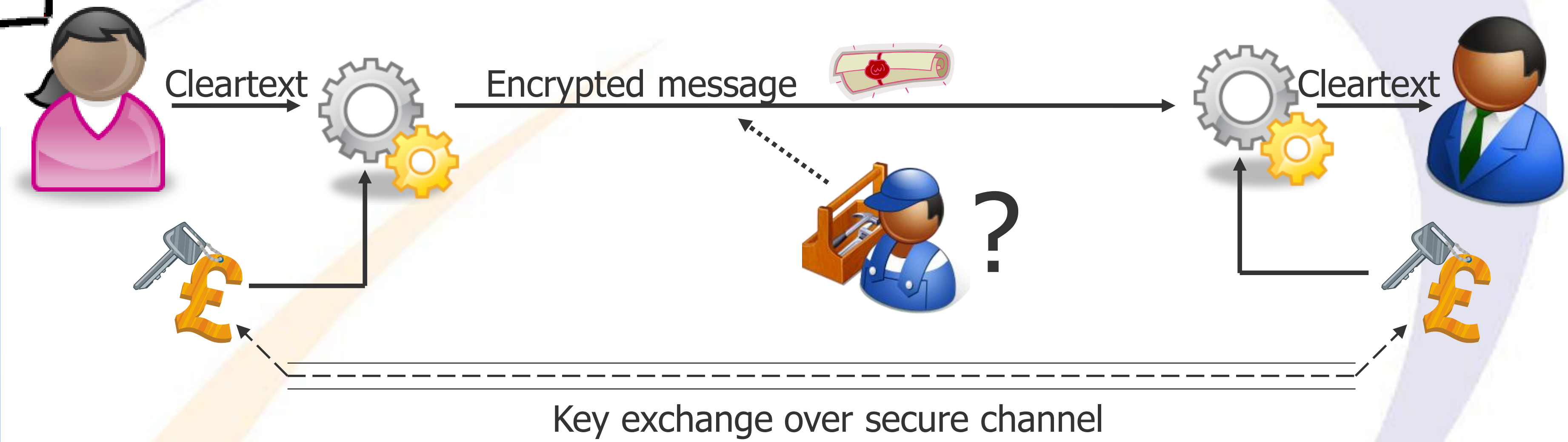
Symmetric vs. asymmetric cryptography

- Symmetric cryptography: Very old
  - → Keys are short: >=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 (≈1970)
  - → Keys are long: >=512 Bits (RSA/DSA; Ell. curves >=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

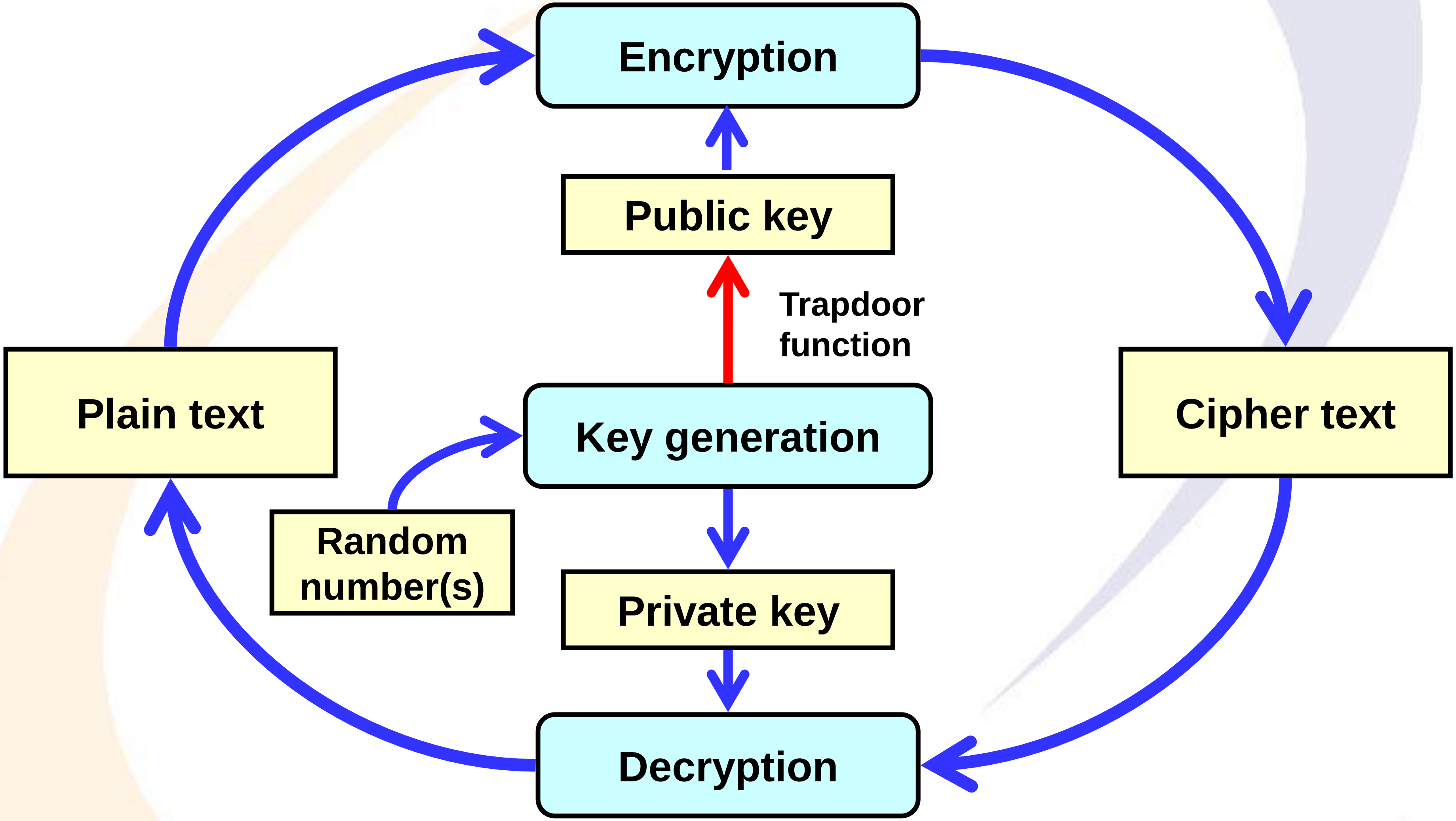


Symmetric cryptography

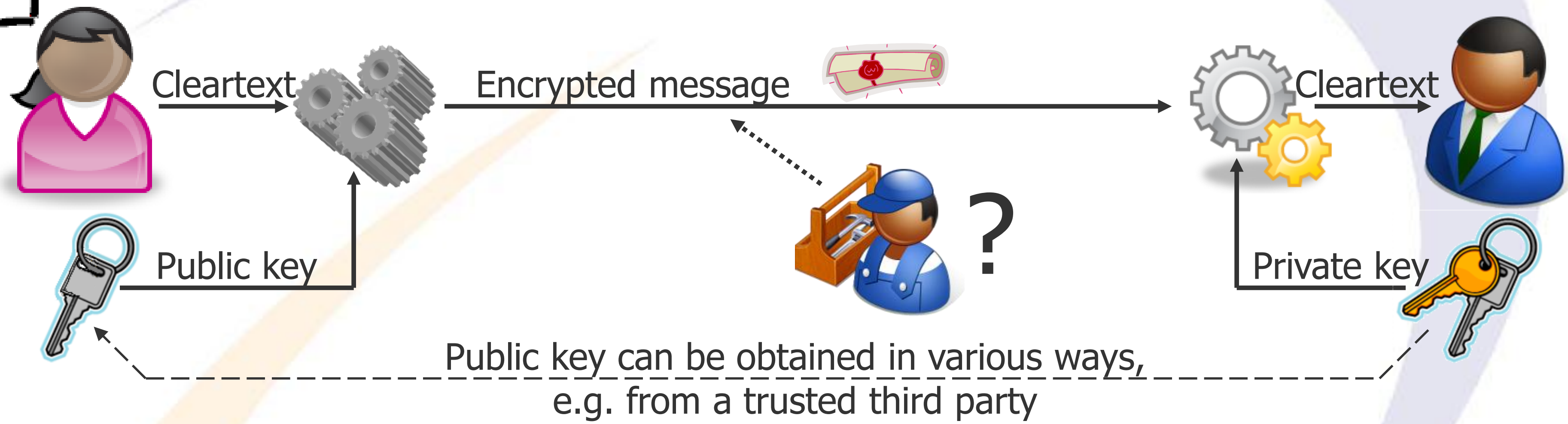


- The same key is used for encryption and decryption
  - → The key must therefore remain absolutely secret!
  - → So it must be transported securely
    - » This is only possible through a different channel!
  - → Keys must be changed regularly (much encrypted content renders breaking the encryption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

Asymmetric (public key) encryption

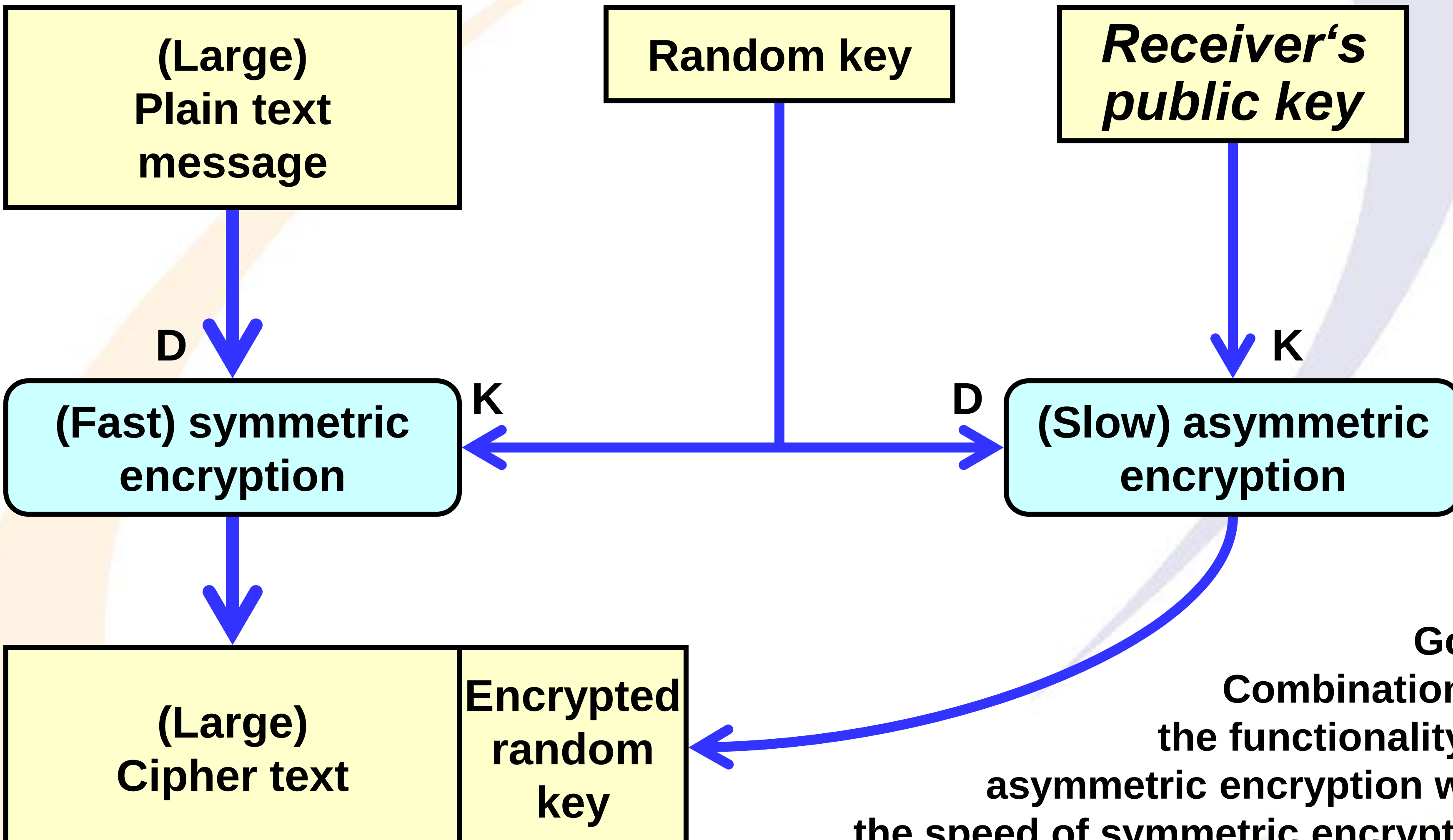


Asymmetric cryptography



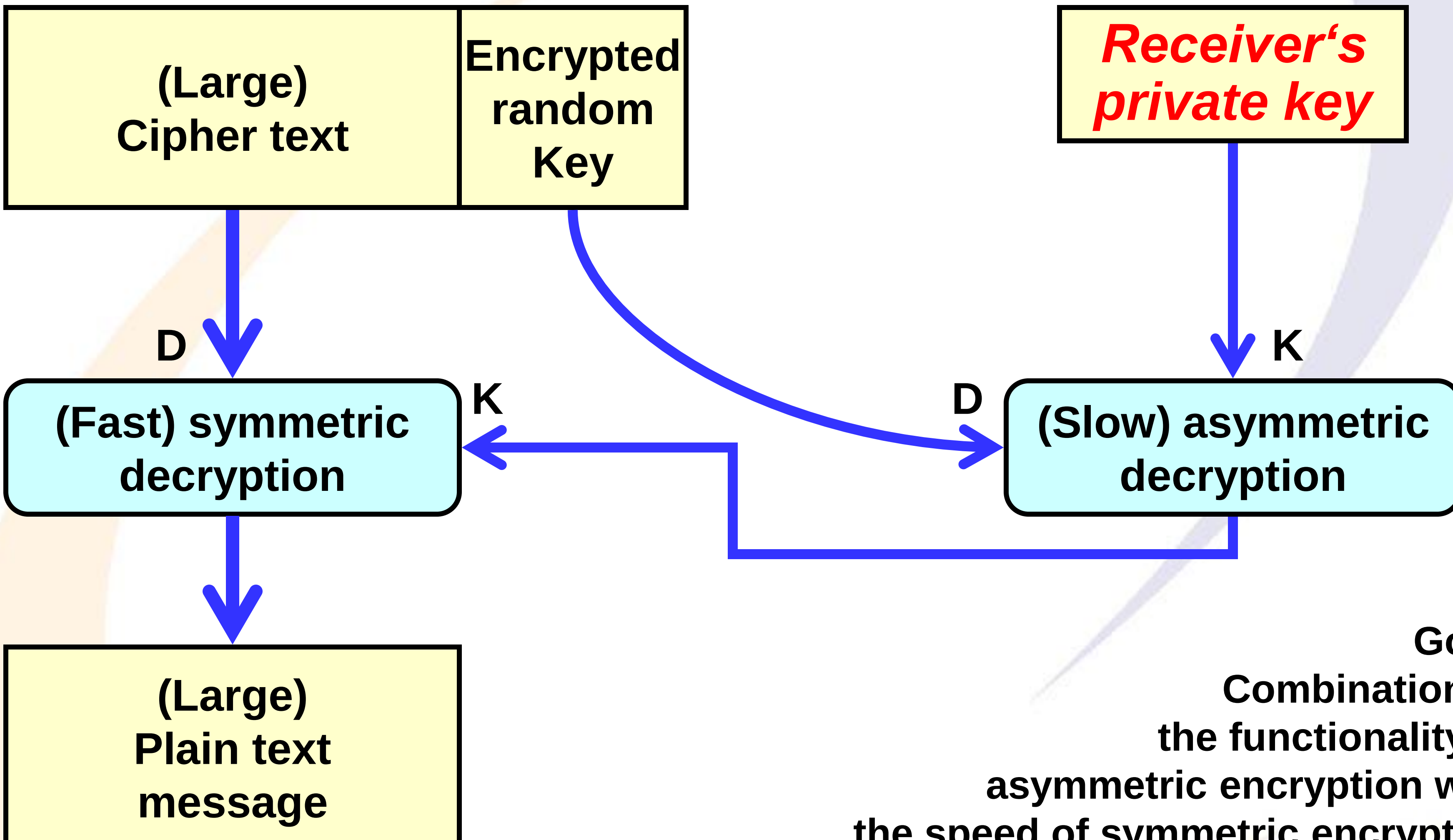
- One part (private) of the key is used for decryption/signature
  - → A encrypted to B: B's private key is used to decrypt
  - → A signed to B: A's private key is used to sign
- The other (public) is used for encryption/verification
  - → A encrypted to B: B's public key is used to encrypt
  - → A signed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

Combining Symmetric & Asymmetric Encryption (1)



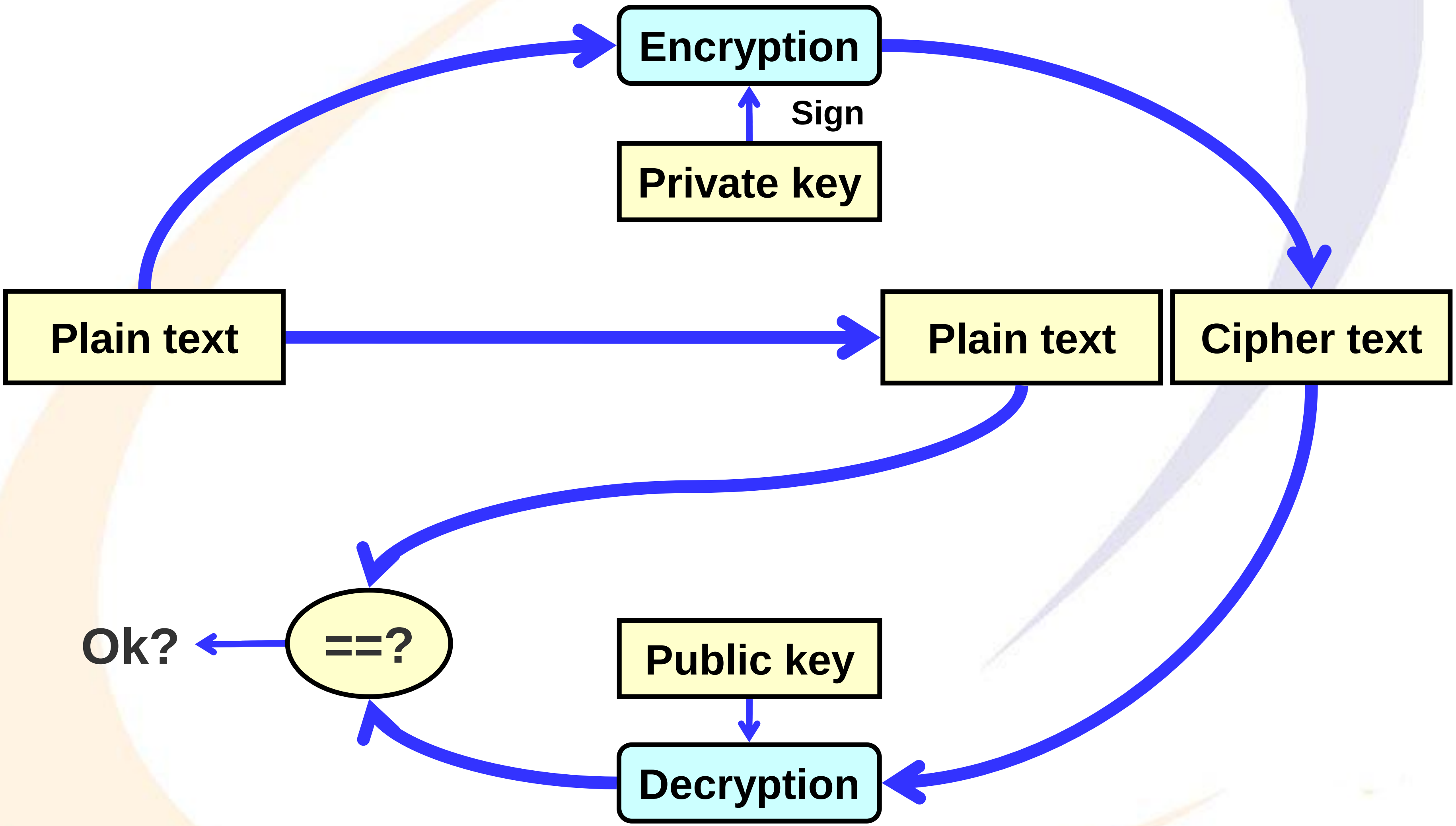
Goal:
Combination of the functionality of asymmetric encryption with the speed of symmetric encryption

Combining Symmetric & Asymmetric Encryption (2)

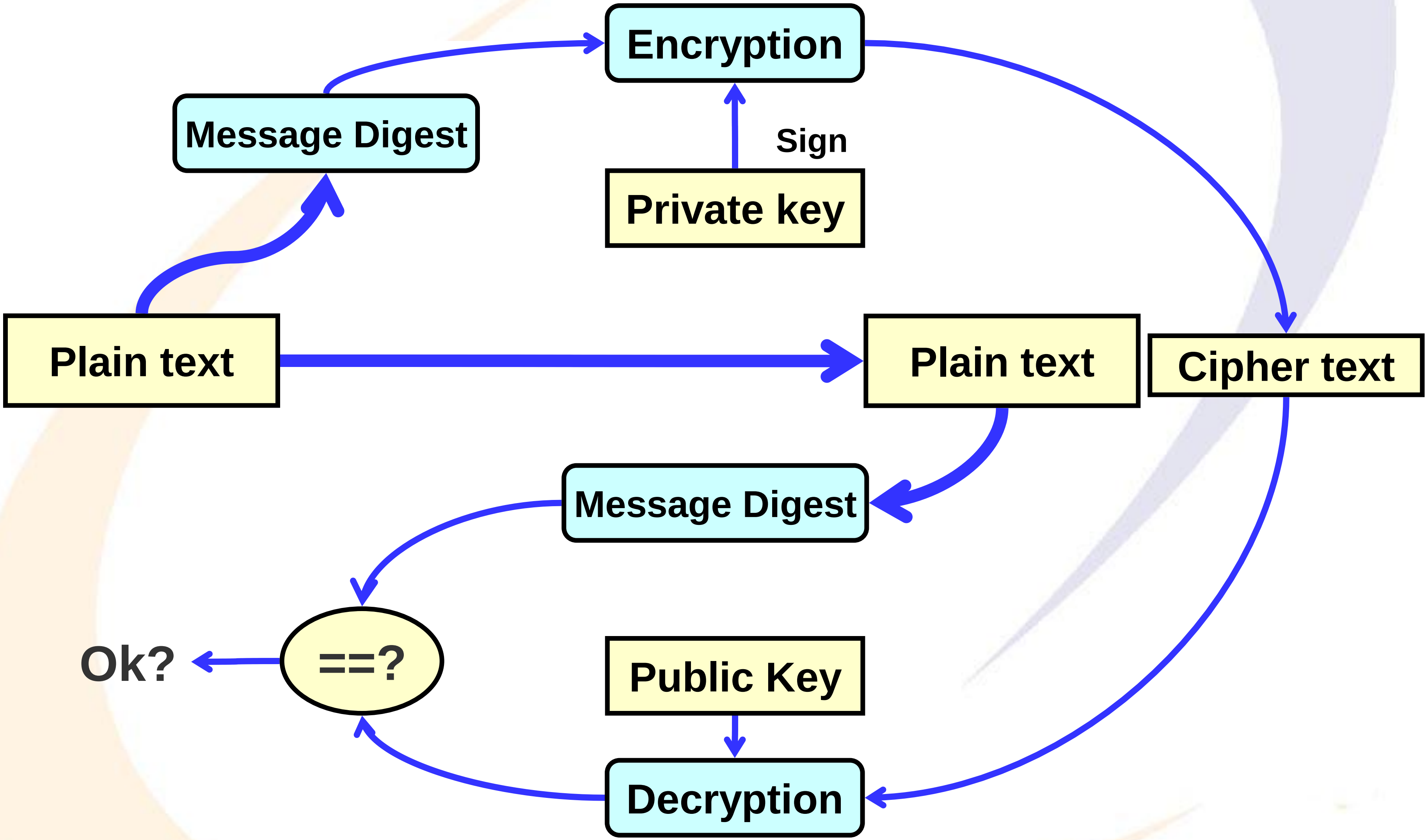


Goal:
Combination of the functionality of asymmetric encryption with the speed of symmetric encryption

Sign: Encryption with private key + ...

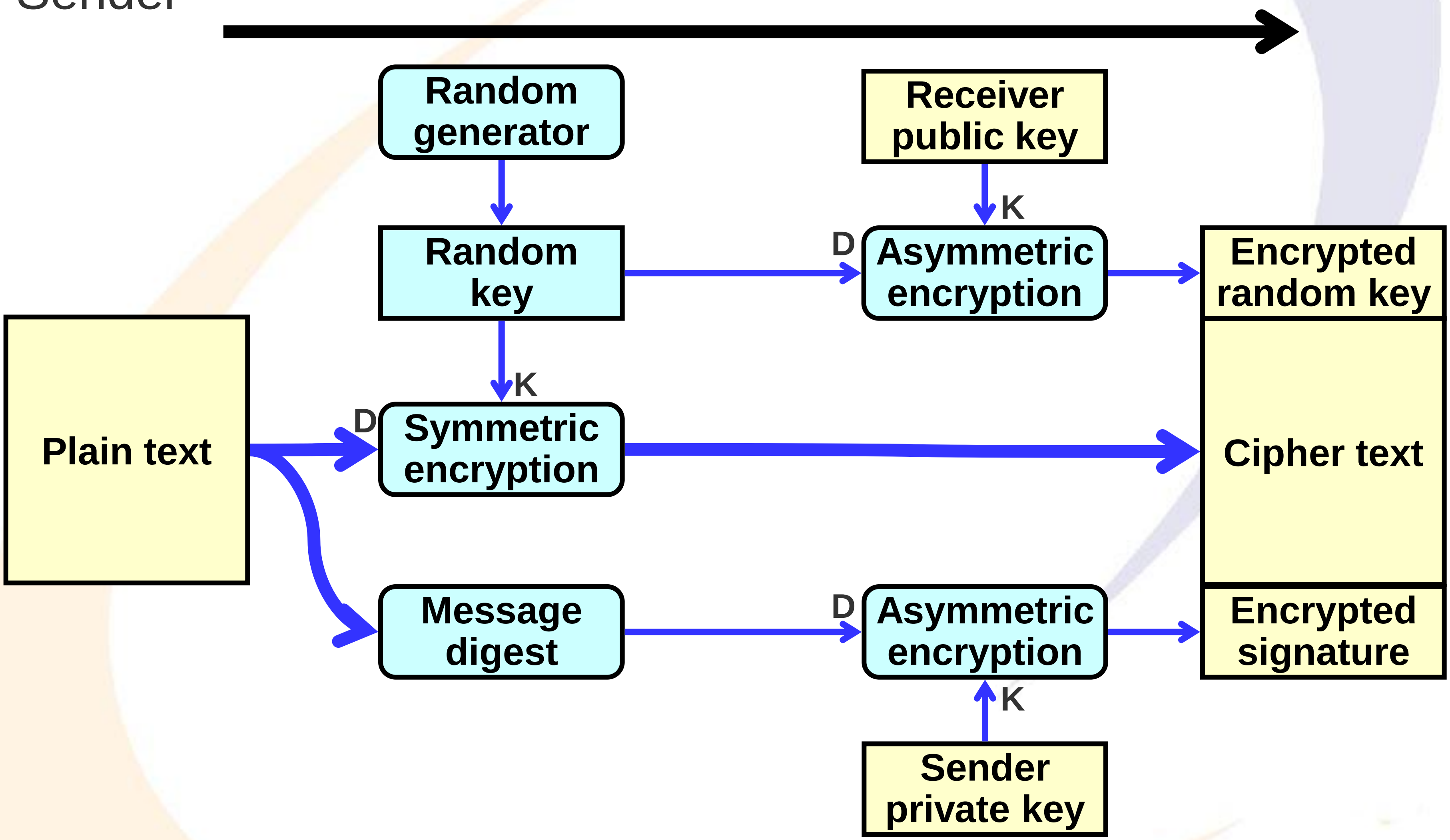


Check: Decrypt with public key + compare



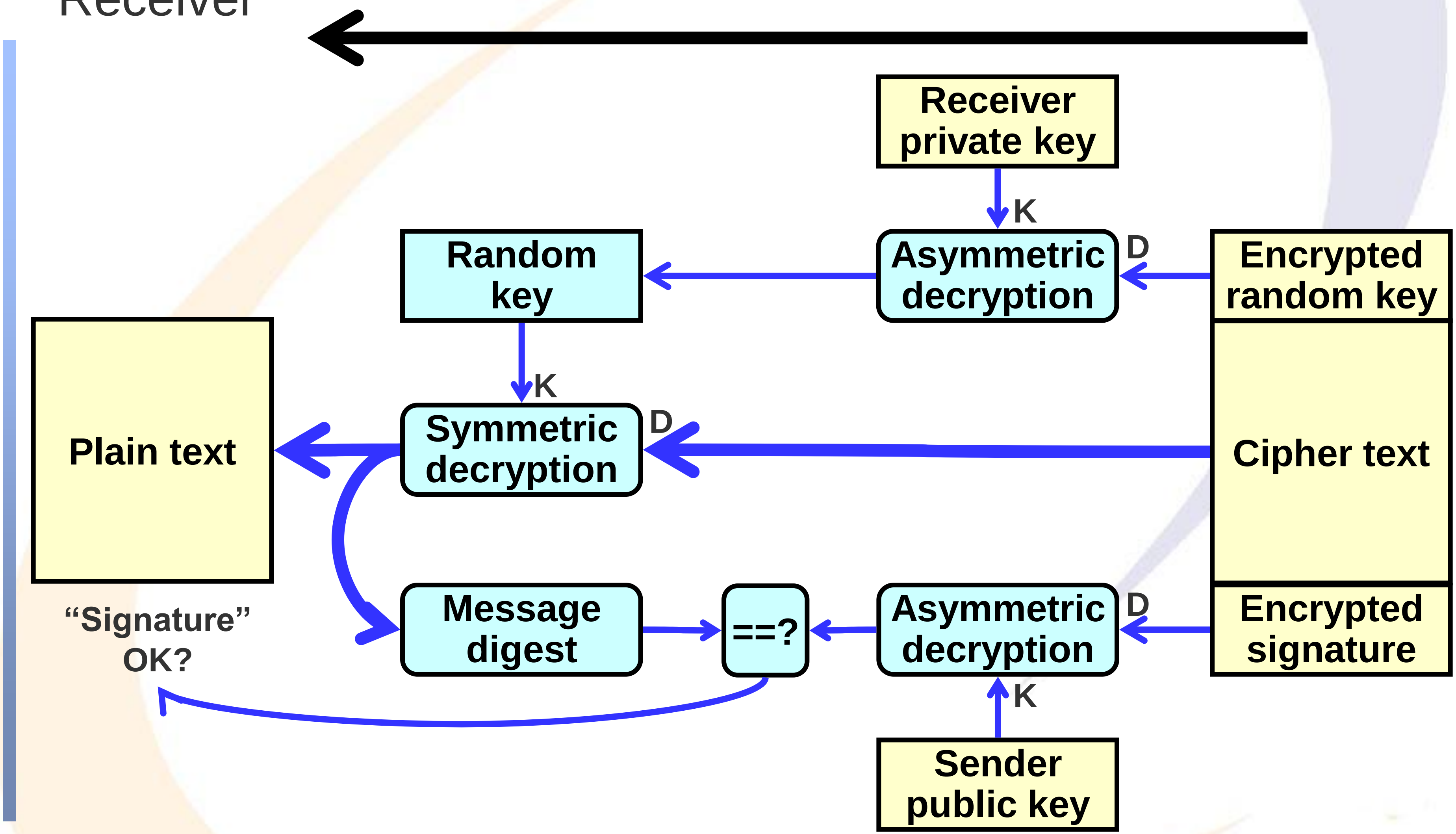
Signature & Encryption

Sender



Signature & Decryption

Receiver



Algorithms

- Symmetric:
  - → DES/3DES: Based on permutations, substitution; block cipher
    - » 56/112 Bit; DES is now insecure
      - – 3DES (=DES three times with different keys) is sufficient for commercial use (frequent key changes recommended)
  - → AES (=Rijndael): "New & standard" algorithm
    - » Several key sizes available
- Asymmetric:
  - → RSA: Classic asymmetric cipher (rather slow)
    - » Arbitrary key size (>=1024 recommended); no longer patented!
  - → Elliptic curves: Based on discrete logarithm
    - » 160 Bit ≈ 1024 Bit RSA
    - » DSA (=one variant): Only signatures, no encryption possible
  - → Diffie-Hellman: Key agreement without previous knowledge
    - » Generates a shared secret key
    - » Original source of all asymmetric cryptography!

Algorithms

- Hash:
  - → SHA-1, RIPEMD-160: 160 Bit
    - » SHA-1: Deemed to be not quite secure anymore
  - → SHA-2: 224-512 Bit
    - » Bit length varies
  - → MD5: 128 Bit (not recommended any more; insecure)
- Look unimportant, but in practice the security of many signatures (and other procedures) absolutely rely on them!
  - → Don’t break the signature, just create a new document with the same hash as a known one and copy the signature!
- Currently there is a selection procedure going on in the USA for choosing a new standard hash algorithm
  - → Started 2007, probable end: 2012
    - » So long because it includes intensive scientific scrutiny of all candidates in several rounds!

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/RFID (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 15 years (values below: 2024)
      - – Symmetric: ≈ 89 Bit
      - – Asymmetric: RSA, ...: ≈ 2113 Bit; DSA: ≈ 157 Bit
      - – Corresponds to a budget for an attack in 1 day of ≈ 732 million US$

Source: Lenstra, A. K.,Verheul, E. R.: Selecting Cryptographic Key Sizes. DuD 24 (2000), 166
Full article: http://security.ece.orst.edu/koc/ece575/papers/cryptosizes.pdf

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 mod 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 mod p B→ A: g^b mod p
- A calculates: $(g^b \bmod p)^a \bmod p$ = SECRET
- B calculates: $(g^a \bmod p)^b \bmod p$ = SECRET

Encryption: DES

- Symmetric encryption algorithm with 56 Bit key length
  - → Now seen as insecure because of short key length
  - → Specifically secure against a much later published attack
    - » Differential cryptoanalysis → Was already known but kept secret!
  - → Currently attacks with brute force possible
- Superseded by 3DES and AES
  - → Triple-DES: Encrypt(key1,Decrypt(key2,Encrypt(key3)))
    - » Because of a special attack, security is still 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)

Encryption: AES

- 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, therefore …
  - → 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
  - → good random number generators are needed
- Prime numbers are usually only checked probabilistically
- Key length: 1024 Bit might be broken in the near future
  - → Recommendation: Use 2048 Bit key

Signing: DSA

- 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

Hashing: MD5

- 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!
  - → Certain attacks are trivial (find some bytes that attached to a freely selectable file produce a chosen hash value) or simple (find "any" two documents with same hash value); for some still strong (find another document for a given hash value)

Hashing: MD5

- Function:
  - → Cut message up into 512 Byte blocks (use padding for last)
  - → Based on addition, shifting, non-linear function
- Still often used for storing password: "store MD5(pwd)"
  - → Use at least "salting“
  - → Or: "Better" algorithms, i.e. requiring more time to calculate
- Salting: Should be used with all kinds of hash algorithms!
  - → Create a random value (for each operation anew!): Salt
    - » Not only for each account, but for each new password to encrypt!
  - → Create hash of random value concatenated with data
  - → Store result + cleartext of random value: “x, MD5(x || pwd)”
  - → Verification: Concatenate password with salt, hash it, and compare the result to the stored value

Hashing: SHA-x

- 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 know attacks, but mathematics has proved weaknesses
  - → Originally thought: 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 know weaknesses, but not much investigation either!

Hashing: SHA-3 (Keccak)

- Selected after public world-wide competition in 2012
  - → Not yet (2/2013) officially announced as a standard
- Not a replacement for SHA-2 (no problems known), but rather as an alternative
  - → Should SHA-2 prove to be susceptible in the future
- Inner working:
  - → 5*5 array of 64-bit words
  - → Compute parity of columns and XOR it into two columns
  - → Bitwise rotation of all words for a different number of bits
  - → Permutation of the 25 words in a fixed pattern
  - → Bitwise combine with next two rows (non-linear function!)
  - → XOR of one word of state with linear shift register output

Hashing: SHA-3 (Keccak)

- Calculation:
  - → XOR some bits into the state, perform block permutation, XOR next bits, ……
  - → End: Leading bits of the state are the hash
  - → Number of bits to add per round depends on the hash size
    - » 144 bytes → 224 bit hash; 72 bytes → 512 bit hash
      - – "Better" hash → Add data to function more slowly to allow it to "permute" through all parts of the state!
- Speed:
  - → Very fast in hardware
  - → Software: Allegedly 12.6 CPU cycles per input byte
    - » On x86-64 with 64-bit code

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?
      - – And how to invalidate them afterwards!
    - » Who is allowed to do what? Does the equipment support these different security levels?
    - » Securing keys/certificates etc. against loss

Certificates

- 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 "Dagobert Duck" (or a certain person using this pseudonym)?
  - → "Someone" guarantees, that these two belong together

Certificates connect a public key to a name

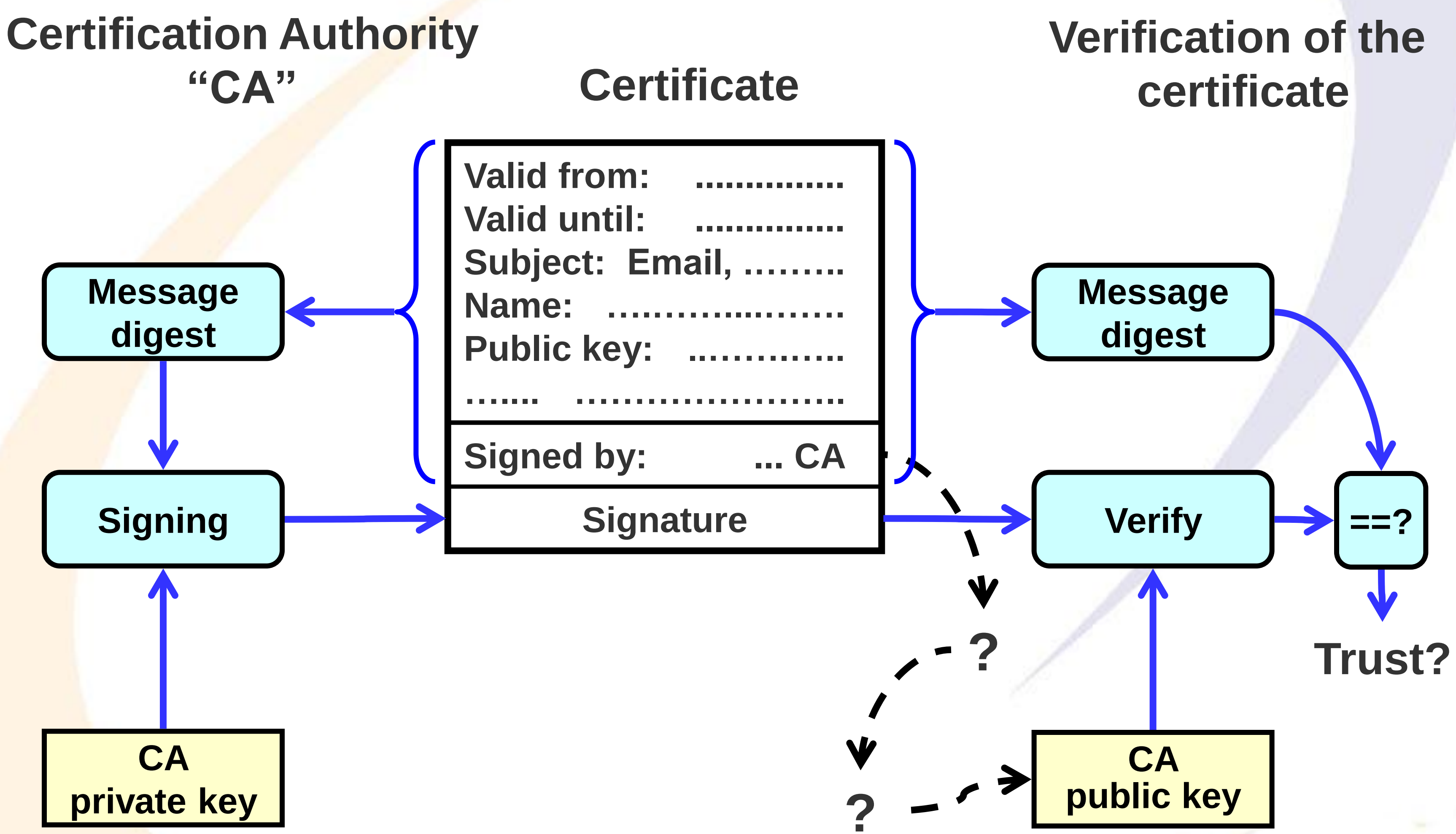
- Certificates may contain other information
  - → E.g. server certificates may contain the administrator's E-Mail
  - → Personal certificates may contain restrictions or special permissions/empowerments
    - » "May act on behalf of company A", transaction restrictions etc.
- Certificates are signed 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 is three → 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.

See http://www.hack.org/mc/texts/x509guide.txt

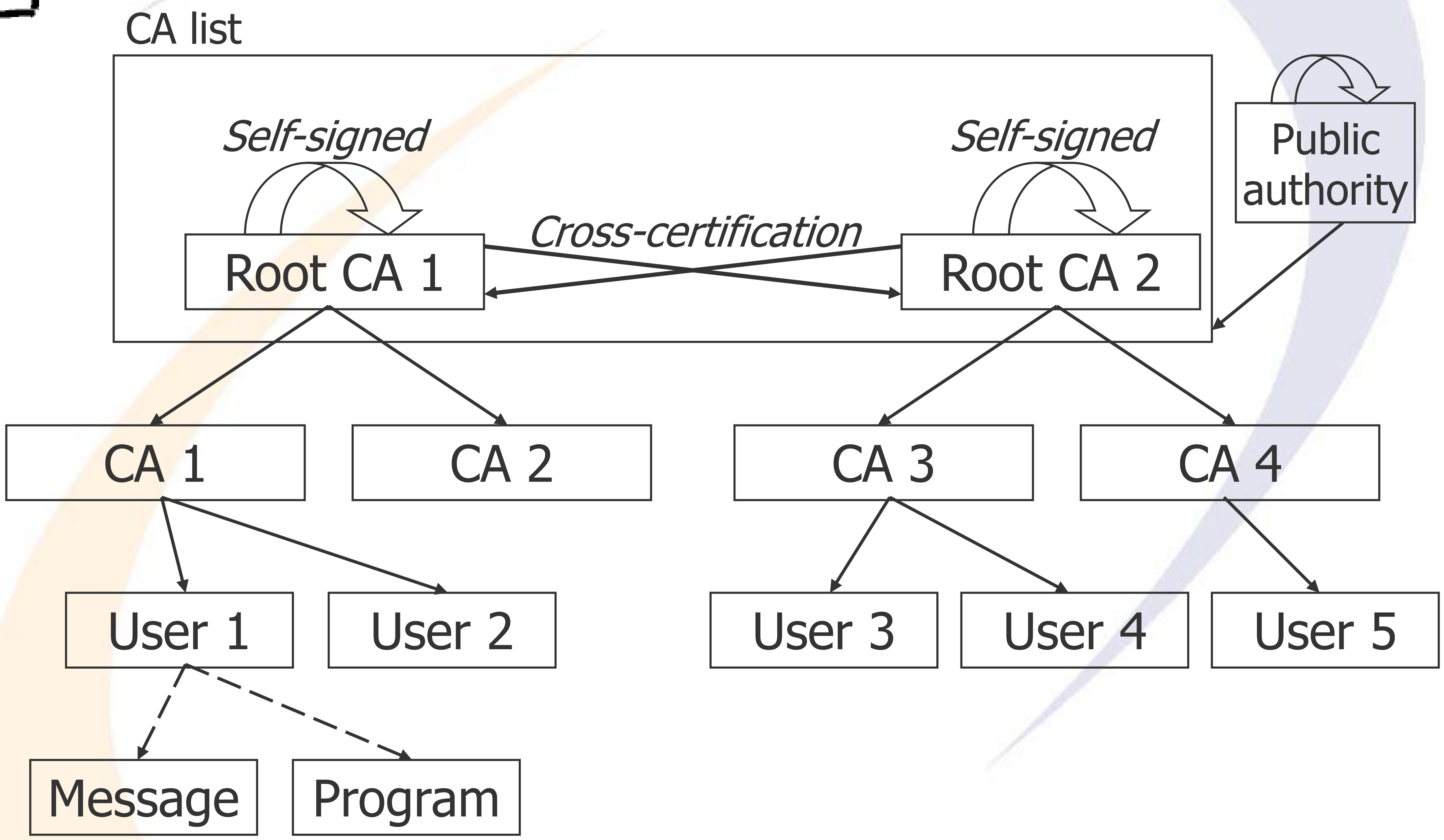
Certificate



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 public key contained in it
  - → 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 only (otherwise: $O(N^2)$!)

PKI Example



CA list
Self-signed
Self-signed
Public authority
Cross-certification
Root CA 1
Root CA 2
CA 1
CA 2
CA 3
CA 4
User 1
User 2
User 3
User 4
User 5
Message
Program

Certificate revocation

- Sometimes certificates must be "removed", e.g. when
  - → some attributes are incorrect (name/profession changes)
  - → private key is disclosed
  - Or: Private Key of CA disclosed/broken!
  - → 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, …)

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!), all other major browsers

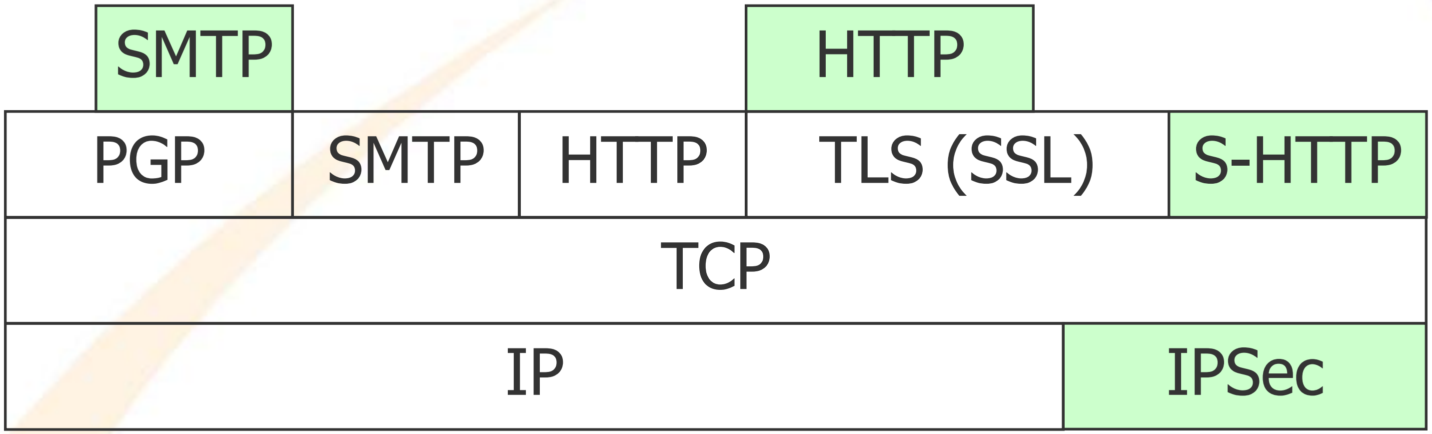
Certificates and digital signatures

- 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 (⇒https; NOT: **s**http)
    - » Option to use http and switch internally to TLS (=STARTTLS)!
  - → TLS (Transport Layer Security): SSL successor, very similar
- TLS provides:
  - → Encrypted communication: Eavesdropping impossible
    - » Depends on the actual algorithm/key length used
    - » Uses symm. cryptography for speed; numbering 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!
  - → Normally the server alone is authenticated
    - » Client authentication only in closed systems (→ cert. distrib.!)
- 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**

ClientHello →

ServerHello
[Certificate]
[ServerKeyExchange]
[CertificateRequest]
← ServerHelloDone

[ClientCertificate]
ClientKeyExchange
[CertificateVerify]
ChangeCipherSpec
Finished →

ChangeCipherSpec
← Finished

← - - - - - - →
Encrypted [and authenticated] communication

[]: Optional parts

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 several 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!
    - » Sometimes: “Try again later” is accepted as “valid”
- 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 in a telephone call ⇒ works!
      - » No additional infrastructure needed (PKI, CRL, etc.)

Alternatives: Web of trust

- 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 TLS!
  - → 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
  - → The system reliable only, if keys are signed by many people
  - → Such people are not found easily everywhere

"Official" certificates: Advantages / Disadvantages

- \+ No warning messages for browsers
- \+ Identity of person/company verified accurately
  - » More trust than a self-signed certificate
- \+ Browser interoperability (creating a "good" one 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!)
  - At least one free provider now available (http://cert.startcom.org/)!
- − 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 (rare combination!)

"Official" certificates: Obtaining one

- Fill in form for certificate (or local software)
  - → Creates a "Certificate Signing Request" (CSR)
    - » Contains the certificate data, but not the private key!
- Pay the price
- CA verifies the content
  - → Usually through notarized/official documents
    - » Perhaps also personally (depending on application)
- CA creates the certificate
  - → Signed by its own private key
- CA makes the certificate available
  - → To the customer
  - → Usually also in the directory
    - » Everyone can download it

VPNs

- 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
  - → But possible regarding what is sent through the VPN!
- Sometimes difficult to set up; interoperability difficult
  - → Becomes better with IPSec
  - → Easier with TLS-based VPNs
- Powerful hardware needed for encrypting larger bandwidth
  - → Dedicated devices/daughtercards, "VPN concentrators", …
- Overhead; more bandwidth required
  - → This is today usually only a small problem!
- Data is physically outside: Not necessarily secured as well!

XML Security

- Consists of two independent parts:
  - → XML Signature: Providing non-repudiation
  - → XML Encryption: Providing secrecy
- Both trivially possible by existing technologies/standards
  - → But only for the complete file!
    - » This prevents e.g. writing the signature into the XML file itself!
    - » Locating parts is no longer possible in encrypted files
    - » Tags are also encrypted ⇒ known plaintext attack possible
    - » No schema validation while encrypted
  - → Solution: Standards for encrypting/signing parts of XML files
- Problem: XML may differ binary, but be logically the same
  - → E.g. LF, blanks, entity style/replacement, CDATA sections,…
  - → Solution: Canonical XML
    - » Specific "formatting" always producing the same binary result

C14N: XML Canonicalization

- Produce unique physical representation of an XML fragment
  - → Not foolproof: Even more strict is "Exclusive XML Canonic."
  - → Works not really well for parts which are not well-formed
- Unifies:
  - → Character set: Always UTF-8 in NFC (=Normalization Form C)
  - → Linebreaks: Always #xA
  - → Attribute values: Normalized, double quotes, default attributes added
  - → Content text: CDATA, entities, special characters, …
  - → Superfluous elements: XML declaration, DTD, unneeded NS
  - → Extraneous whitespace: Within tags, outside of document el.
  - → Ordering: Attributes within a tag, namespace declarations
- Limitations:
  - → Base URIs, notations, external unparsed entity references, attribute types in DTD

- Encrypted can be:
  - → The whole XML documents
  - → A single XML element
  - → XML element content: several (sub-)elements
  - → XML element content: character data
- Encrypted data can again be encrypted without problem
- Encrypted data is represented by the following information
  - → Encryption method: The algorithm used
  - → Key information: How to find the decryption key
    - » Symmetric encryption: The key itself (encrypted!)
    - » Asymmetric encryption: The public key used
    - » General: Name or pointer to the key to be used
  - → The enciphered data: Value or pointer to it
  - → Additional properties

XML Encryption Algorithms

- Algorithms are identified by URIs
  - → Some must be implemented (not used!), some are optional
- Block encryption: TripleDES, AES-128, AES-256, AES-192
- Stream encryption: None specified!
- Key transport: RSA-v1.5, RSA-OAEP
- Key agreement: Diffie-Hellman
- Symmetric key wrap (encrypting keys): TripleDES, AES-128, AES-256, AES-192
- Message digest: SHA1, SHA256, SHA512, RIPEMD-160
- Message authentication: XML digital signature
- Canonicalization: (Exclus.) canonical; with(-out) comments
- Encoding: Base64
  - → The encoded result is for almost all algorithms binary data!

Required Recommended Optional

XML Encryption Problems

- When namespaces are used, these may be inherited by the element which is to be encrypted
  - → Or explicitly removed by specifying ' xmlns:ns="" '
- When this is encrypted and later decrypted and put into a different context, the result might be invalid!
  - → With empty namespace even in the same context
    - » On canonicalization this might be stripped away, so after decryption the default namespace is inherited instead of removed!
- xml:base, xml:lang, xml:space attributes may cause problems
  - → These are also inherited!

The application must take care to specify these things explicitly or know exactly into which context to put the result of decryption!

XML Signature

- A signature consists of
  - → The actual signature value (base64 encoded)
  - → Signature information:
    - » Canonicalization, signature, digest method
    - » What was signed: URI/XPath, …; additional transformations
  - → Information on the key to use for verification
    - » E.g. certificate (X.509, PGP, …), key name, …
  - → Object information: What is actually signed
  - → Additional properties: E.g. timestamp
- Three kinds of signatures exist
  - → Enveloping: Signed data contained within *Object* information
  - → Enveloped: An ancestor of the signature is signed
    - » The signature itself must be excluded from digesting, obviously!
  - → Detached: External content (identified by *URI* or *Transform*)

XML Signature Transformations

- Describe how to obtain the data object to be digested
  - → Ordered list: Result of first is input for second, …
- Each transform consists of an algorithm and appr. Attributes
- Examples:
  - → Two enveloped signatures required: Each signature must exclude itself, but it must also exclude the other signature
- Enveloped transform: Equivalent to the following XPath transform
  - → <XPath xmlns:dsig="&dsig;">
count(ancestor-or-self::dsig:Signature | here()/ancestor::dsig:Signature[1]) >
count(ancestor-or-self::dsig:Signature)</XPath>
    - » If the direct parent signature is in the set of all outer signatures, this element is excluded from signing

XML Signature Algorithms

- Algorithms are identified by URIs
  - → Some must be implemented (not used!), some are optional
- Digest: SHA1
- Encoding: Base64
- MAC: HMAC-SHA1
  - → MAC=Message Authentication Code (=crypt. hash algorithm)
- Signature: DSAwithSHA1, RSAwithSHA1
- Canonicalization: Canonical XML omitting comment, with comm.
- Transform: Enveloped signature, XPath, XSLT

Required Recommended Optional

XML Signature + Encryption

- Both do not specify new algorithms
  - → These must be acquired separately (patent problems, …)!
- Combining both can lead to problems
  - → Signing encrypted data: How to know what is really signed?
    - » Should be avoided; task of the application!
  - → Encrypting signed data: How to know whether signature verification should be done before decryption or afterwards?
    - » If complete structure is encrypted $\Rightarrow$ no problem
    - » When only subparts are encrypted, this gets important!
    - » Example: Signing the payment information and later on en-crypting the creditcard number, but leaving the name in cleartext
    - » There exists a separate specification for this!
      - – Introduces "exception" elements to the transformation

Conclusions

- 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!

Questions?

Thank you for your attention!