Public Key Systems (2)

The RSA algorithm

Two components:

- Selecting the keys
- Applying the encryption and decryption algorithm

Selecting the keys (by Bob):

1. Choose two large primes, *p* and *q*
2. Compute $n = p \times q$ and $z = (p-1) \times (q-1)$.
3. Choose a number relatively prime to *z*, smaller than *n* and call it *e* (*e* is used for *e*ncryption) .
4. Find *d* such that *exd - 1 is dividable by z without any remainder* (ed mod z = 1) (*d* stands for decryption) .
5. The public key is (*n*,*e*), the private key is (*n*,*d*).

Encryption (by Alice) of a bit pattern (number) *m* such that $m < n$ by means of Bob's public key (*n*,*e*).
The resulting cipher *c* is:

$c = m^e \bmod n$

Decryption (by Bob) of *c* by means of his private key (*n*,*d*) in order to get the plaintext *m*:

$m = c^d \bmod n$

Public Key Systems (3)

Example of the RSA algorithm

p=5, q=7 ---> n = 35, z=24. Further, Bob selects e=5, d=29 (5*29 - 1 can be divided by 24)

----> public key of Bob: (35,5), private key of Bob: (35, 29)

Alice wants to send the message “LOVE” to Bob by encrypting each letter separately and interpreting each letter as the corresponding number (a maps to 1, ..., z maps to 26)

Tabelle 7.1 Die RSA-Verschlüsselung von Alice, $e = 5$, $n = 35$

Klartextbuchstabe	m: numerische Darstellung	m^e	Chiffretext $c = m^e \bmod n$
L	12	248832	17
O	15	759375	15
V	22	5153632	22
E	5	3125	10

Tabelle 7.2 Die RSA-Verschlüsselung von Bob, e = 29, n = 35

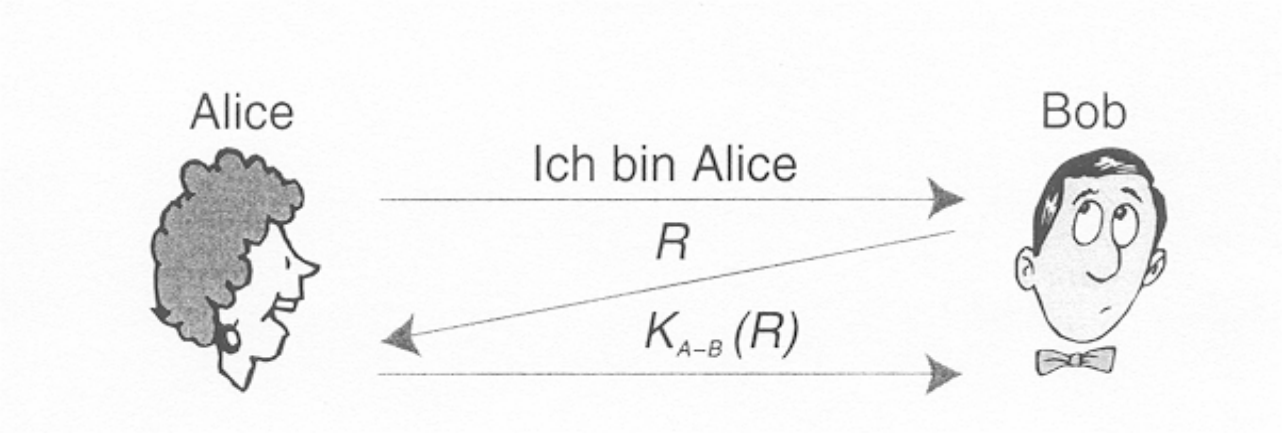
Chiffre-text c	c^d	Chiff-retext $m = c^d \bmod n$	Klartext-buch-stabe
17	48196857210675091509141182522307200 0	12	l
15	12783403948858939111232757568359400	15	o
22	8.516433190865377011956194499721 11e+38	22	v
10	10000000000000000000000000000	5	e

Authentication

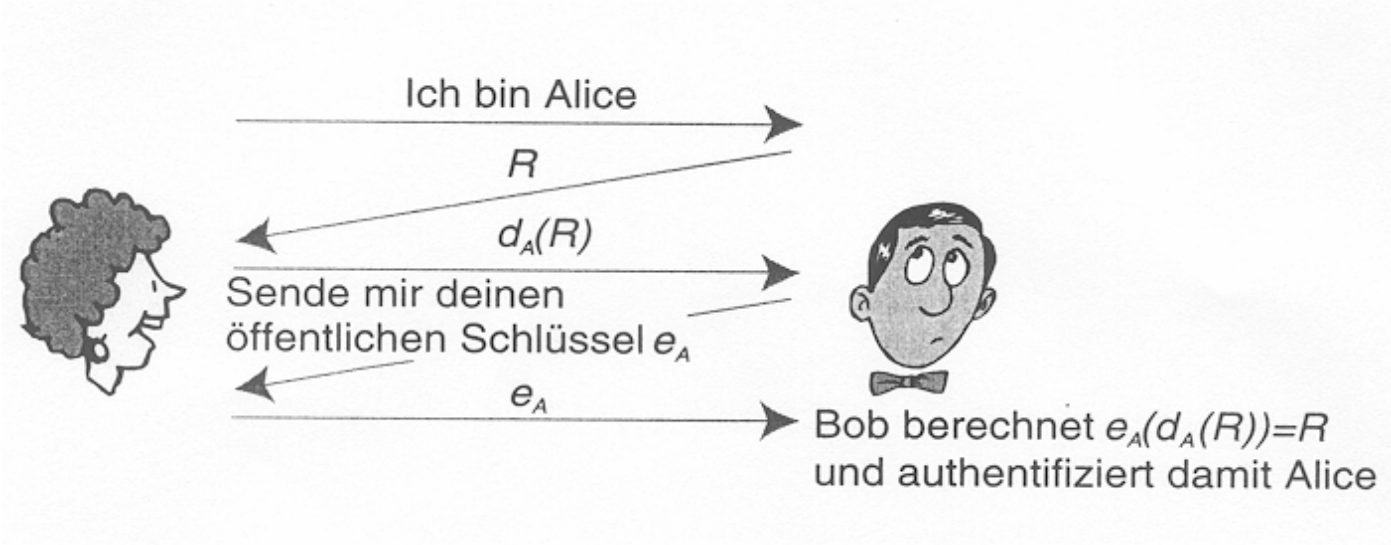
Authentication Protocols

- technique by which a process verifies that its *actual* communication partner is who it is supposed to be
- normally done before the partners start to exchange data messages, e.g. e-mails

Version with symmetric keys



Version with public keys

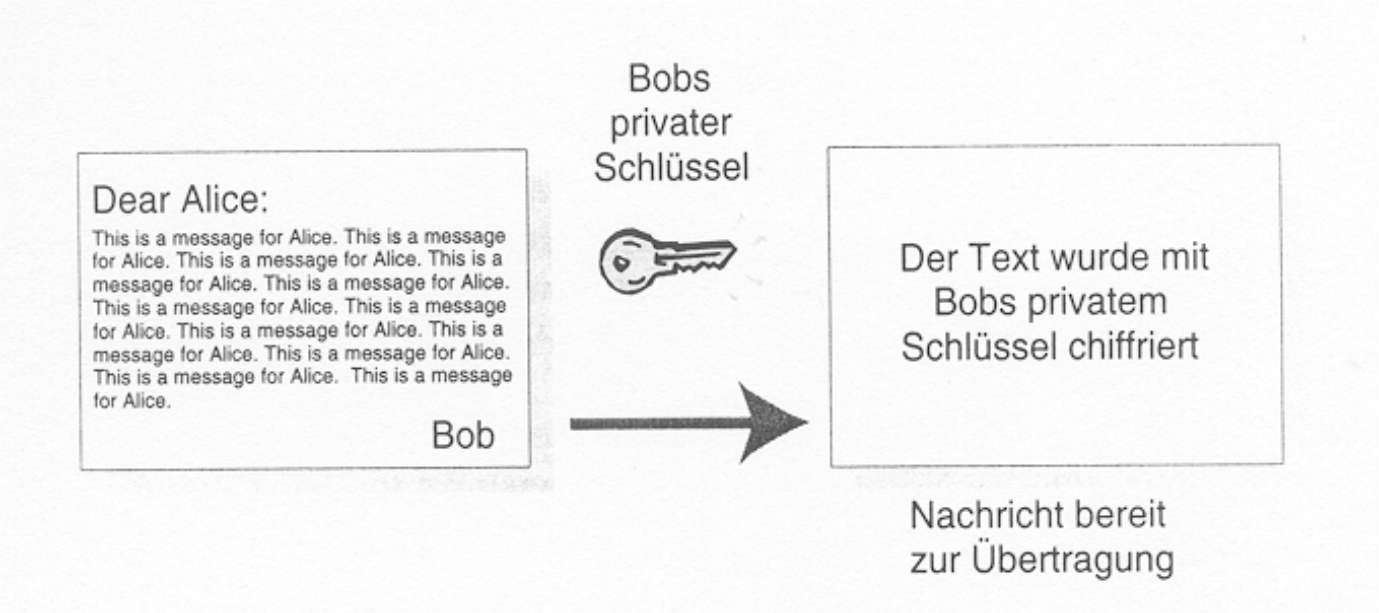


Digital Signatures (1)

Problem:
Finding an electronic adequate for the handwritten signature such that one party can send a signed message to another party in such a way that the following conditions hold:

- The receiver can verify the claimed identity of the sender (authentication)
- The sender later cannot repudiate having sent his message (nonrepudiation)
- The contents of the message cannot have been modified, e.g. by the receiver himself (integrity)

Solution 1: Creation of digital signatures by means of public keys



Drawback:

It couples secrecy on the one side with the triple (authentication, nonrepudiation, integrity) on the other side ---> it needs, often unnecissarily, too much computational overhead for encrypting/decrypting

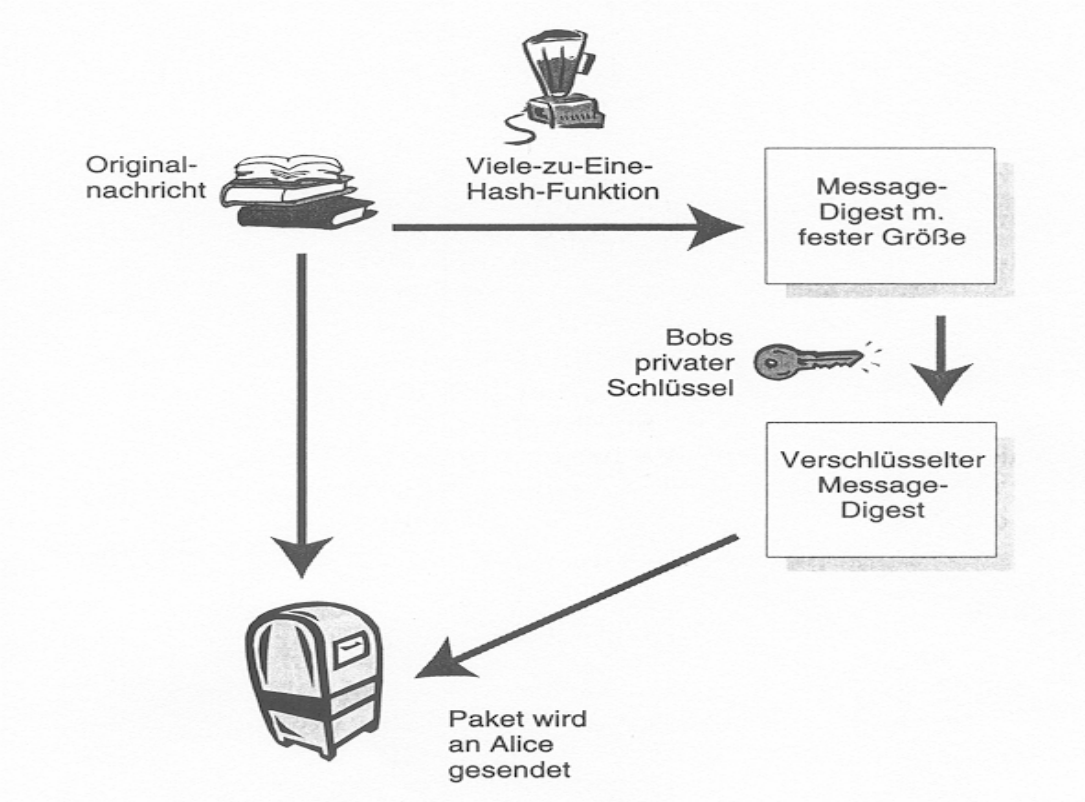
Digital Signatures (2)

Solution 2: Creation of digital signatures without encrypting the whole text.
Idea: Using a so-called *message digests* to create a "fingerprint" from any plaintext.
Message Digest H(m): a message *m* of any length is mapped to a bit string *H(m)* of fixed length such that

- *H(m)*is much shorter than *m* and is computed much easier (faster) than encrypting *m*
- it is almost impossible to find *m'* ‡ *m* and *H(m) = H(m')* (ensuring data integrity)

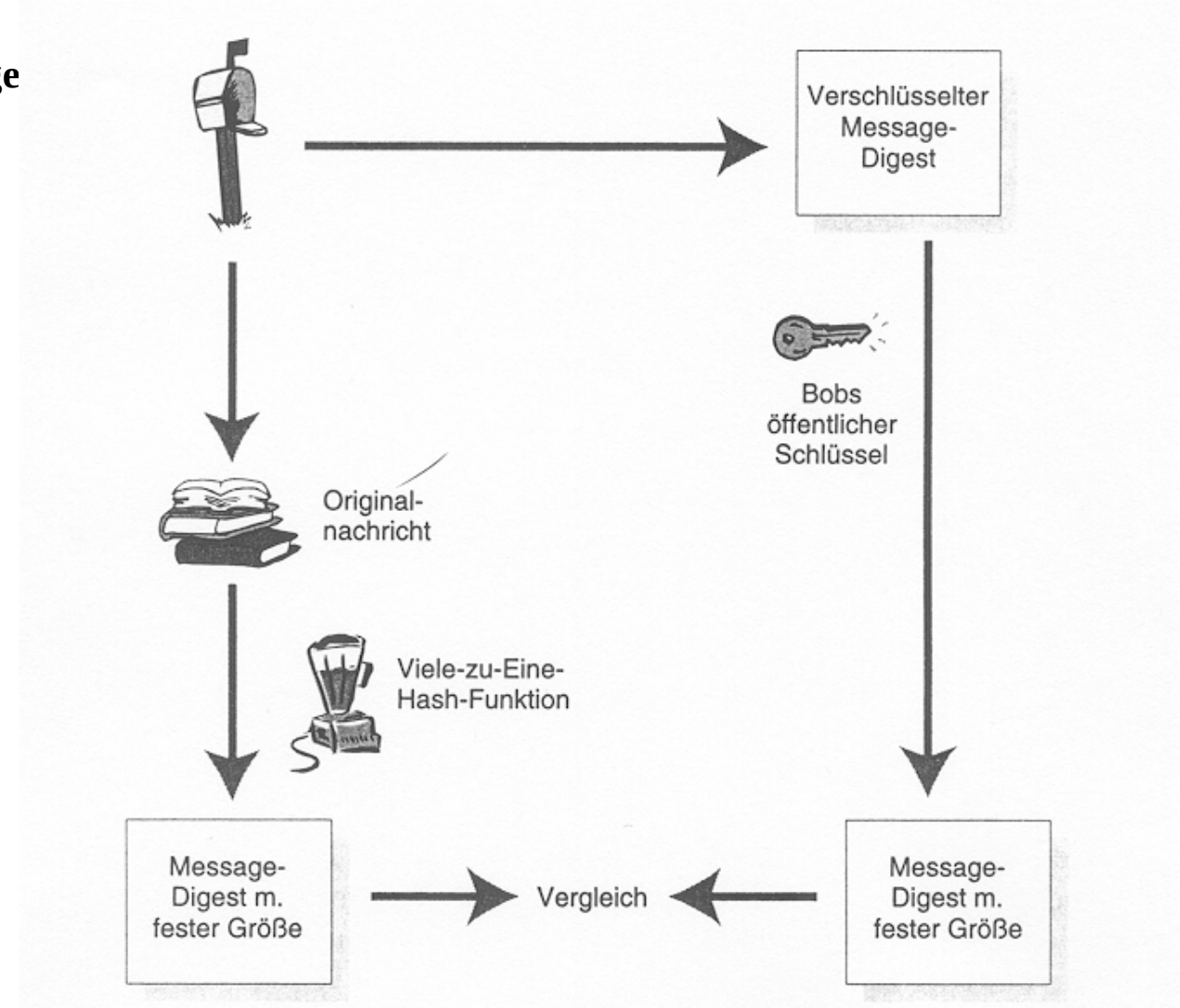
Now, in order to get the effect of a digital signature, we only have to encrypt (sign) the digest of a message.

Sending a digitally signed message



Digital Signatures (3)

Checking a digitally signed message



The most widely used message digest functions are MD5 (128bits long) and SHA-1 (160 bit long). They operate by mangling bits in a sufficiently complicated way such that every output bit (bit of the digest) is affected by (dependent on) some input bit (bit of the message).