Sicherheit:
Fragen und Lösungsansätze

im Wintersemester 2012 / 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 8: Asymmetric Encryption and Digital Signatures with RSA

v. 27.01.2013

technische universität
dortmund

fakultät für
informatik

Themen der Vorlesung

Part I: Challenges and Basic Approaches

1) Interests, Requirements, Challenges, and Vulnerabilities
2) Key Ideas and Combined Techniques

Part II: Control and Monitoring

3) Fundamentals of Control and Monitoring
4) Case Study: UNIX

Part III: Cryptography

5) Fundamentals of Cryptography
6) Case Studies: PGP and Kerberos
7) Symmetric Encryption
8) **Asymmetric Encryption and Digital Signatures with RSA**
9) Some Further Cryptographic Protocols

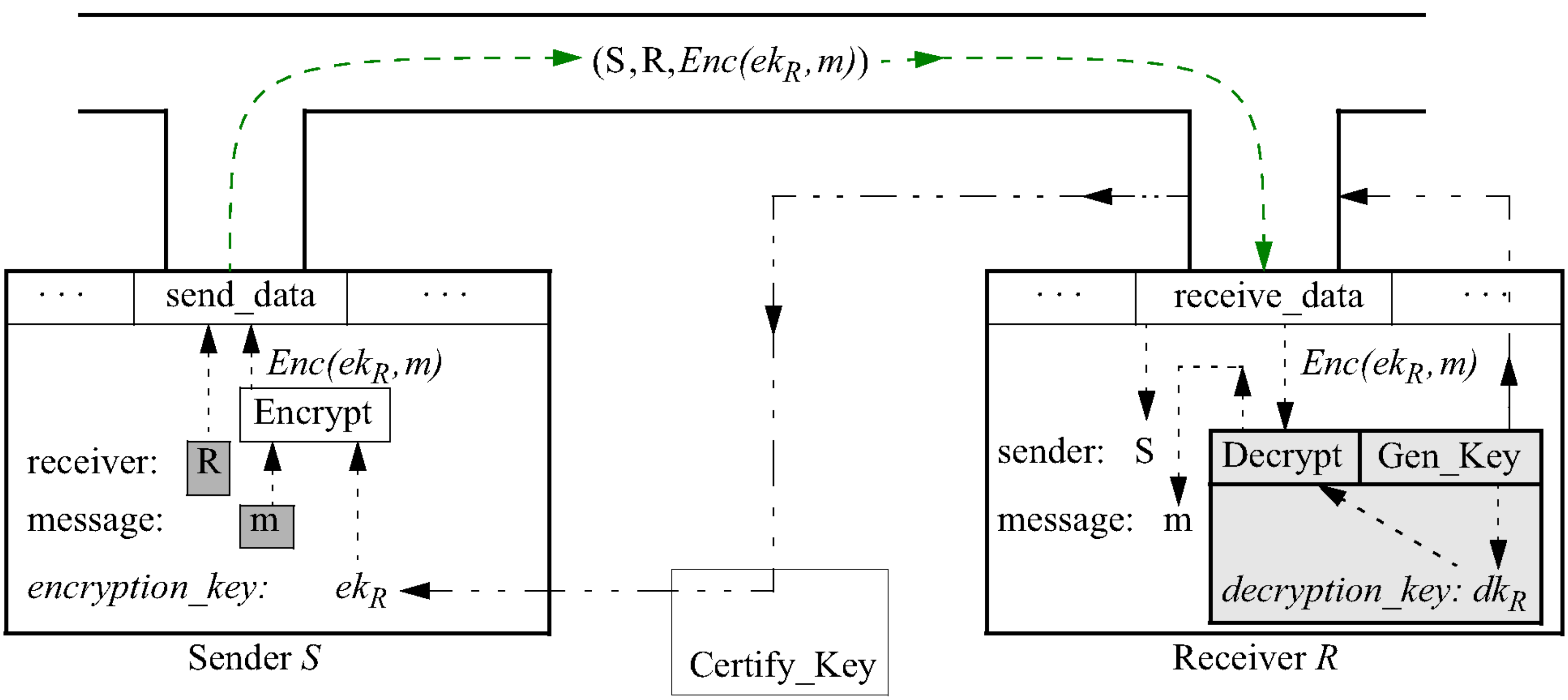
Part IV: Access Control

10) Discretionary Access Control and Privileges
11) Mandatory Access Control and Security Levels

Part V: Security Architecture

12) Layered Design Including Certificates and Credentials
13) Intrusion Detection and Reaction

Asymmetric encryption



Complexity-theoretic secrecy property (one-way function approach)

sender *Alice*

attacker *Malory*

possible plaintexts:

⋮

x_i ● —— public encryption key ——→ ● y observed ciphertext;

⋮

matching plaintext x_i
cannot be “feasibly determined”
since computational effort is too high
(without knowledge of the private key)

Family of one-way functions with trapdoors

Parameterized family of functions f_k such that for each k:

- function $f_k : D_k \rightarrow R_k$, $x \rightarrow f_k(x)$
  is *injective* and computable in *polynomial time*
- inverse function $f_k^{-1}: R_k \rightarrow D_k$, $y \rightarrow x$ where $y = f_k(x)$
  is *computationally infeasible* without a knowledge of k
  (note: we still need to refer to k to actually have an inverse function)
- inverse function $f_k^{-1} : R_k \rightarrow D_k$, $(y,k) \rightarrow x$ where $y = f_k(x)$
  is computable in polynomial time
  if k (the private key) is used as an additional input

It is an outstanding *open problem* of computer science (closely related to the open problem of whether $P \neq NP$) whether such families actually exist.

RSA functions

- an *RSA function* $RSA^{n,\,e}_{p,\,q,\,d}$ is a number-theoretic function where
  - (p, q, d) is used as the *private key*
  - (n, e) as the *public key*

- the designated *secret holder* generates, *randomly* and *confidentially*, two different, sufficiently large *prime numbers* p and q

- $n := p \cdot q$
  is published as the modulus of the ring ($\mathbf{Z}_n$, +, ·, 0, 1):
  - all computations are performed in this ring
  - the multiplicative group is formed by those elements that are relatively prime to the modulus n, i.e.,

    $\mathbf{Z}_n^* = \{ x \mid 0 < x < n$ with $gcd(x, n) = 1\}$
  - this group has a cardinality $\varphi(n) = (p - 1)\cdot(q - 1)$
  - *Euler phi function* φ, is used for investigating properties of exponents for exponentiations

RSA functions

- the designated secret holder *randomly* selects
  the second component e of the *public key* such that
  $1 < e < \varphi(n)$ and $gcd\,(e, \varphi(n)) = 1$

- additionally, the designated secret holder *confidentially* computes
  the third component d of the *private key*
  as the multiplicative inverse of e modulo $\varphi(n)$:
  $1 < d < \varphi(n)$ and $e \cdot d \equiv 1 \bmod \varphi(n)$
  – in principle, multiplicative inverses can be efficiently computed
  - in this specific situation a knowledge of $\varphi(n)$ is needed,
    which requires one to know the secretly kept prime numbers p and q

- the *RSA function* for the selected parameters is defined by
  $RSA_{n,e,d} : \mathbf{Z}_n \to \mathbf{Z}_n$ with
  $RSA_{n,e,d}(x) = x^e \bmod n$
  - can be computed by whoever knows the public key (n, e)
  - the required properties of
    injective one-way functions with trapdoors (are conjectured to) hold

Injectivity and trapdoor: theorem

In the setting of the *RSA function* $RSA_{n,e,d}$

for all $x \in \mathbf{Z}_n$,

$(x^e)^d \equiv x \bmod n$

Injectivity and trapdoor: sketch of proof

the following congruences modulo n are valid for all $x \in \mathbf{Z}_n$:

$(x^e)^d$	$\equiv$	$x^{e \cdot d}$	exponentiation rules
	$\equiv$	$x^{k \cdot \varphi(n) + 1}$	$e \cdot d = k \cdot \varphi(n) + 1$, definition of d
	$\equiv$	$x \cdot (x^{\varphi(n)})^k$	exponentiation rules

Case 1, $x \in \mathbf{Z}_n^*$:

multiplicative group $\mathbf{Z}_n^*$ has order $\varphi(n)$: $(x^{\varphi(n)})^k \equiv 1^k \equiv 1 \bmod n$

thus: $(x^e)^d \equiv x \bmod n$

Case 2, $x \notin \mathbf{Z}_n^*$:

case assumption: $gcd\,(x, n) \neq 1$

n product of prime numbers p and q: $gcd\,(x, p) \neq 1$ or $gcd\,(x, q) \neq 1$

show for each subcase: $(x^e)^d \equiv x \bmod p$ and $(x^e)^d \equiv x \bmod q$

by the definitions of n, p and q

and *Chinese remainder theorem*: $(x^e)^d \equiv x \bmod n$

Subcase 2a

$gcd\ (x, p) \neq 1$:

p is prime: *p* divides *x* and
thus any multiple of *x* as well

hence: $(x^e)^d \equiv x \bmod p$

similarly:

$gcd\ (x, q) \neq 1$ implies

$(x^e)^d \equiv x \bmod q$

Subcase 2b

***gcd* (*x*, *p*) = 1:**

then $x \in \mathbf{Z}_p^*$ and, accordingly,

the following congruences modulo p are valid:

$x^{\varphi(n)}$	$\equiv$	$x^{(p-1)\cdot(q-1)}$	definition of $\varphi(n)$
	$\equiv$	$(x^{p-1})^{q-1}$	exponentiation rules
	$\equiv$	$1^{q-1} \equiv 1$	$x \in \mathbf{Z}_p^*$ has order $\varphi(p) = p-1$

as in Case 1, we then obtain the following congruences modulo p:

$(x^e)^d$	$\equiv$	$x^{e\cdot d}$	exponentiation rules
	$\equiv$	$x^{k\cdot\varphi(n)+1}$	$e\cdot d = k\cdot\varphi(n)+1$, definition of d
	$\equiv$	$x\cdot(x^{\varphi(n)})^k$	exponentiation rules
	$\equiv$	$x\cdot 1 \equiv x$	congruence shown above

similarly:

gcd (*x*, *q*) = 1 implies

$(x^e)^d \equiv x \bmod q$

Factorization conjecture of computational number theory

The factorization problem
restricted to products of two prime numbers, i.e.:

Given a number n of known form $n = p \cdot q$
where p and q are prime numbers,

to determine the actual factors p and q,
is computationally infeasible.

RSA conjecture

If the non-keyed inversion problem for RSA functions
was computationally feasible,

then the factorization problem
would be computationally feasible as well

Specialized RSA conjecture:

If the non-keyed inversion problem for RSA functions
by means of determining the private exponent *d* from an argument-value pair
was computationally feasible,

then the factorization problem
would be computationally feasible as well.

RSA conjecture and further conjectures

- RSA conjectures roughly says:
  "factorization" is *feasibly reducible* to "RSA inversion".

- The converse claim, namely:
  "RSA inversion" is feasibly reducible to "factorization",
  provably holds:

  If an "attacker" was able to feasibly factor the public modulus *n*
  into the prime numbers actually employed,
  then he could feasibly determine the full private key
  by just repeating the computations of the designated secret holder.

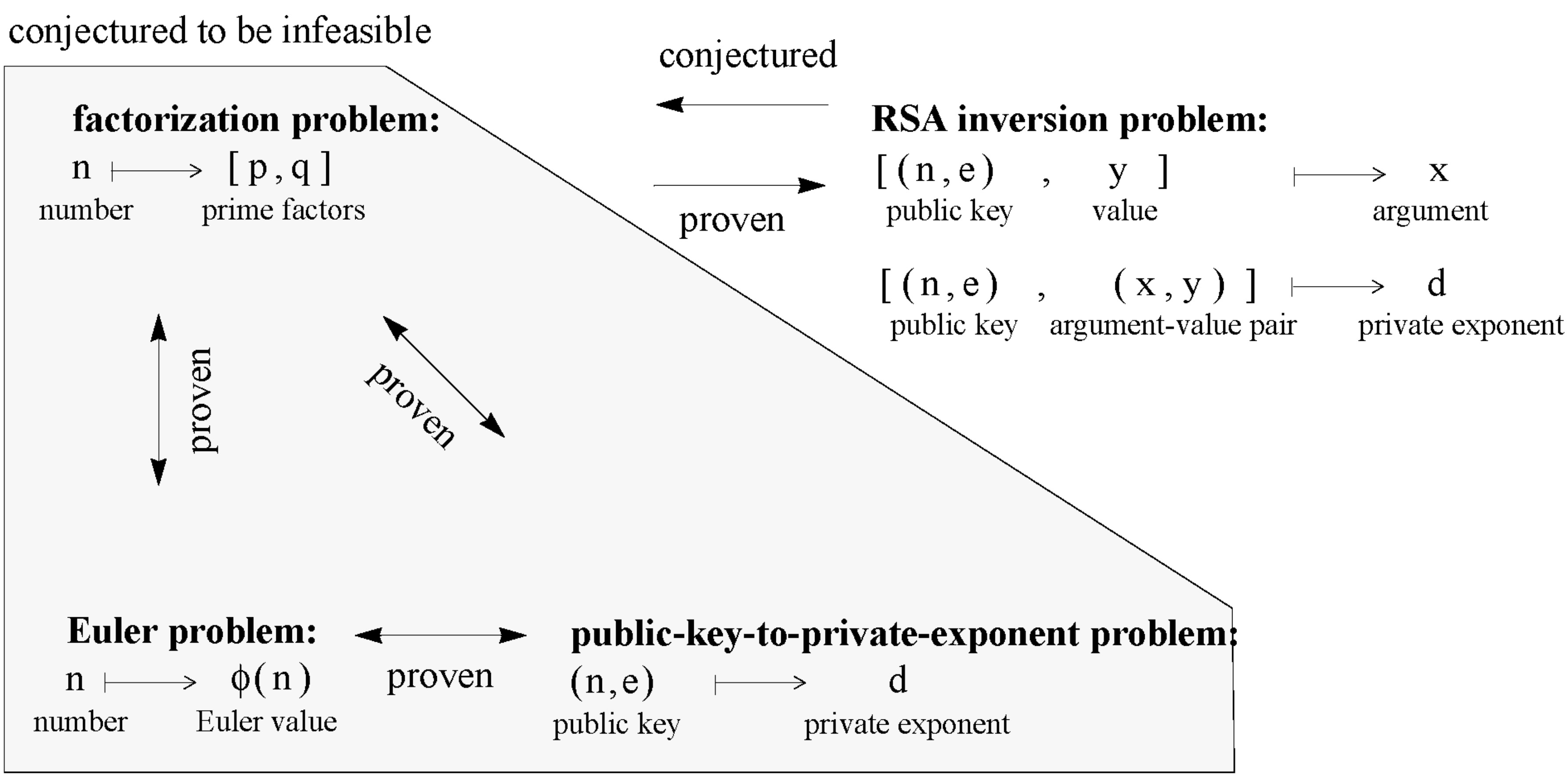
Some similar proven claims

"Factorization" is feasibly reducible to any of the following problems, and vice versa:

- ***Euler problem***:
  Given a number n of known form $n=p \cdot q$,
  where p and q are prime numbers,
  to determine the value $\varphi(n)$.

- ***Public-key-to-private-exponent problem***:
  given the public key (n, e),
  to determine the private exponent d.

Conjectures and proven claims about feasible reducibility



RSA asymmetric block cipher

- is an example of the *one-way function* approach

- is based on RSA functions and their properties

- is *asymmetric*, admitting *multiple* key usage

- operates *blockwise*, where the block length is determined by the parameters of the underlying RSA function

- achieves *complexity-theoretic security*, provided:
  - the *factorization conjecture* and the *RSA conjecture* hold
  - the key is properly generated and sufficiently long
  - some additional care is taken

What additional care ?

The RSA function $RSA_{n,e,d} : \mathbf{Z}_n \rightarrow \mathbf{Z}_n$, $RSA_{n,e,d}(x) = x^e \bmod n$
has the following properties:

• It is **deterministic**. Deterministic asymmetric encryption schemes are problematic, because they do not conceal message repetition. Also, given a sufficiently small set of possible plaintexts, the attacker can encrypt all possible plaintexts with the public encryption key and compare the result with the given ciphertext. One way to prevent this is to make the overall encryption process probabilistic, e.g. by encrypting not only the given plaintext, but the concatenation of the given plaintext with a freshly generated random number, to be used only once (“nonce”).

• It satisfies the **homomorphic** property f(x1::x2,k) = f(x1,k) :: f(x2,k) (where x1, x2 are plaintexts, k is a key and :: is concatenation of bitstrings). This is problematic because it allows the attacker to manipulate the plaintext in predictable ways by manipulating the ciphertext (without being able to decrypt it). One way to prevent is to provide message integrity, e.g. by encrypting not only the given plaintext, but the concatenation of the given plaintext with its hash (assuming as usual that the hash is not homomorphic).

RSA encryption: protocol outline

- **key generation:**
  selecting a *private key* (p, q, d) and a *public key* (n, e) for $RSA_{n,e,d}$
- **preprocessing of** a message m, using an agreed *hash function*:
  - adding a nonce *non* (for *probabilistic encryption*)
  - adding the hash value h (m , *non*) (for *authenticated encryption*)
- **encryption:** computing $y = x^e \bmod n$
  for x = (m, *non*, h (m, *non*)) ,
  if interpretable as a positive number less than n
- **decryption**: computing $y^d \bmod n$
  for received message y
- **postprocessing** of the decryption result:
  - extracting the three components
  - recomputing the hash value of the first two components
  - comparing this hash value with the third component (received hash value):
    if the received hash value is verified,
    the first component is returned as the (presumably) correct message

RSA encryption: underlying sets

for each fixed setting of an RSA function $RSA_{n,e,d}$:

- **plaintexts:**
  bit strings over the set { 0 , 1}
  of some fixed length $l_{mes} \leq \text{ld } n$ *(where ld = logarithm of base 2)*
- **ciphertexts:**
  bit strings over the set { 0 , 1},
  basically of length ld n
  (binary representation of a positive number less than n (residue modulo n))
- **keys:**
  given the public key (n , e),
  in principle there is a unique residue modulo n
  that can be used as the private decryption exponent d ,
  whose binary representation is a bit string,
  basically of length ld n or less
  (from the point of view of the nondistinguished participants,
  this decryption exponent cannot be "determined")

RSA: key generation Gen

- selects a *security parameter l* that basically determines the length of the key
- generates randomly two large prime numbers *p* and *q*
of the length required by the security parameter (note that both numbers need to be sufficiently large; it is not sufficient if only their product is large).
- computes the modulus $n := p \cdot q$
- selects randomly an encryption exponent *e* that is relatively prime to $\varphi(n) = (p-1) \cdot (q-1)$
- computes the decryption exponent *d* as the solution of $e \cdot d \equiv 1 \bmod \varphi(n)$

RSA: encryption algorithm Enc

- takes a possibly padded message m of length l_{mes} as a plaintext
- generates a random bit string non as a nonce of length l_{non}
- computes a hash value $h(m, non)$ of length l_{hash}
- concatenates these values with appropriate separators:
  the resulting bit string x must, basically, have length ld n
  ($l_{mes} + l_{non} + l_{hash} \leq \text{ld } n$,
  binary representation of a positive number less than n (residue modulo n))
- taking the public key (n, e),
  computes and returns the ciphertext
  $y = x^e \bmod n$

RSA: decryption algorithm Dec

- taking the first component n of the public key (n, e) and the third component d of the private key (p, q, d), inverts the given ciphertext y by computing

  $x = y^d \bmod n$

- decomposes the result x into
  - message part m
  - nonce part non
  - hash value part $hash$

  according to the separators employed

- inspects the received hash value:
  - if $h(m, non) = hash$,
    then m is returned as the (supposedly) correct message
  - otherwise, an error is reported

RSA: fundamental properties

- to be considered: *correctness*, *secrecy* and *efficiency*
- the *modulus n* should have a length of at least 1024;
  even a larger length might be worthwhile to resist dedicated attacks
  (note that both factors p, q need to be sufficiently large as well)

- there is a trade-off between secrecy and efficiency, roughly estimated:
  - *key generation* consumes time $O\,((\mathrm{ld}\ n)^4)$
  - operations of *modular arithmetic*, needed for *encryption* and *decryption*, consume time at most $O\,((\mathrm{ld}\ n)^3)$

- high performance can be achieved in practice
  by employing specialized algorithms for both software and hardware

- there are some known weaknesses of specific choices of the parameters

- preprocessing and postprocessing are necessary:
  - *probabilistic encryption* demanded for sophisticated secrecy property
  - *added nonce* needed for several purposes

Brute-forcing RSA ?

Cf discussion on: http://crypto.stackexchange.com/questions/3043/how-much-computing-resource-is-required-to-brute-force-rsa/3044#3044

The number of primes smaller than x is approximately $\frac{x}{\ln x}$. Therefore the number of 512bit primes (approximately the length you need for 1024 bit modulus) is approximately
$$\frac{2^{513}}{\ln 2^{513}} - \frac{2^{512}}{\ln 2^{512}} \approx 2.76 \times 10^{151}.$$

The number of RSA moduli (i.e. pair of two distinct primes) is therefore
$$\frac{(2.76\times10^{151})^2}{2} - 2.76 \times 10^{151} = 1.88 \times 10^{302}.$$

Now consider that the observable universe contains about 10^{80} atoms. Assume that you could use each of those atoms as a CPU, and each of those CPUs could enumerate one modulus per millisecond. To enumerate all 1024bit RSA moduli you would need:

$1.88 \times 10^{302}\, ms/10^{80} = 1.88 \times 10^{222}\, ms = 1.88 \times 10^{219}\, s = 5.22 \times 10^{215}\, h = 1.43$ Just $\times\, 10^{213}$ years

as a comparison: The universe is about 13.75×10^{9} years old.

RSA: added nonce

- Enlarges the search space for the straightforward *inversion algorithm* that an attacker could use given a ciphertext and the public key.

- Prevents a known *ciphertext / plaintext* vulnerability, by ensuring that a given plaintext *m* will produce different ciphertexts when being sent multiple times.

RSA: authenticated encryption

- needed to prevent active attacks enabled by the *multiplicativity property* (*homomorphism property*) of exponentiation:

  for all x, y and w: $(x \cdot y)^w = x^w \cdot y^w$,

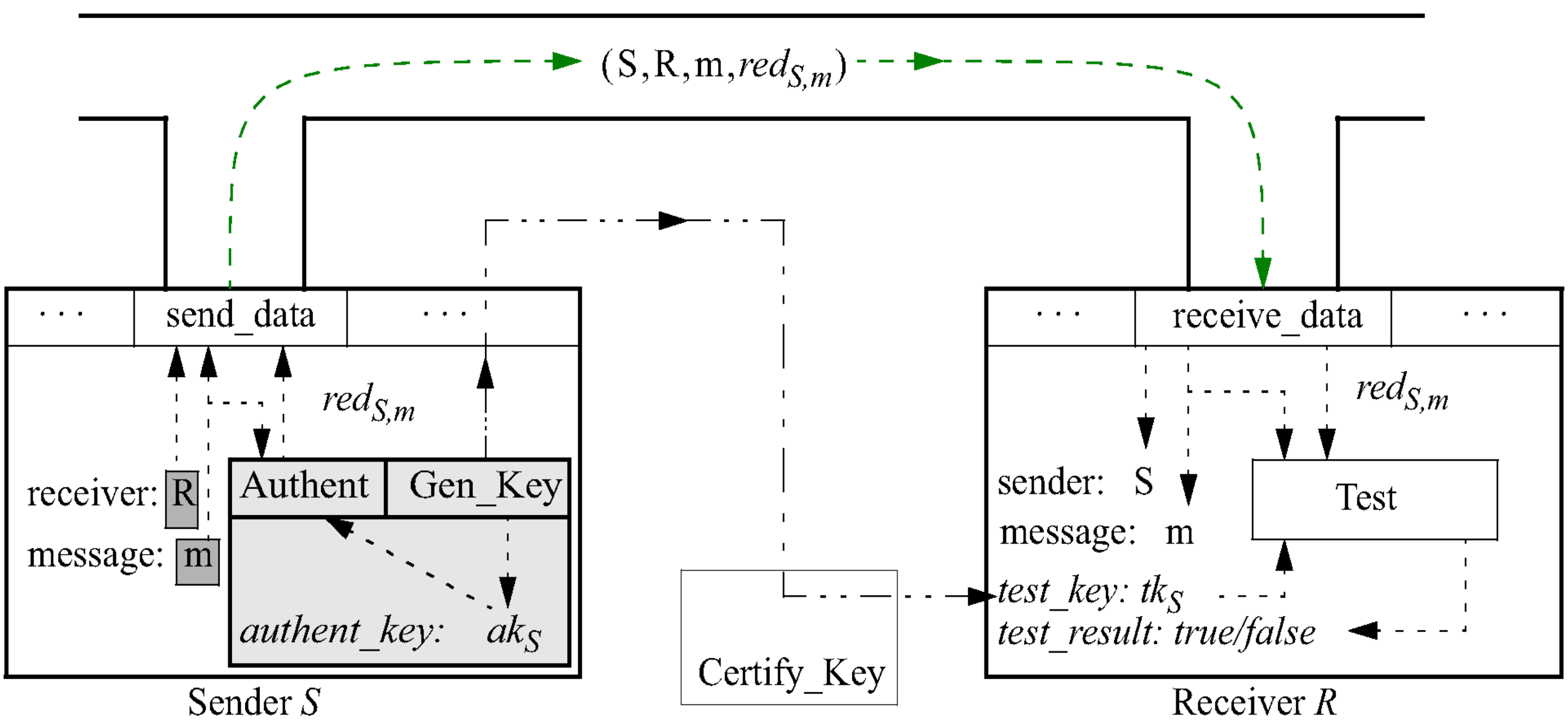
  which is inherited by any RSA function
- example of an *attack to decrypt* an observed ciphertext y:
  - select a multiplicatively invertible element $u \in \mathbf{Z}_n{}^*$
  - compute $t := y \cdot u^{\,e}$ mod n, by employing the public key (n, e)
  - somehow succeed in presenting t as a (harmless-looking) ciphertext to the holder of the private key and obtain

    the corresponding plaintext $t^{\,d}$ with property

    $t^{\,d} \equiv (y \cdot u^{\,e})^d \equiv y^{\,d} \cdot u^{e \cdot d} \equiv y^d \cdot u \mod n$
  - solve the congruence for the wanted value y^d by computing

    $y^{\,d} = t^{\,d} \cdot u^{\,-1} \mod n$
- this attack will not succeed with the employment of a hash function, provided this hash function does not suffer from the same multiplicativity property

Asymmetric authentication (digital signing)



RSA asymmetric digital signatures

- is an example of the *one-way function* approach
- is based on RSA functions and their properties
- is *asymmetric*, admitting *multiple* key usage
- achieves *complexity-theoretic security*, provided:
  - the *factorization conjecture* and the *RSA conjecture* hold
  - the key is properly generated and sufficiently long
  - some additional care is taken
- is obtained by exchanging the roles of encryption and decryption,

  given a suitable *RSA function* $RSA_{n,e,d}$ with
  - *private key* (p, q, d)
  - *public key* (n, e)

RSA digital signatures: protocol outline

- **preprocessing** of a message *m* using an agreed *one-way hash function*: computing a hash value $h(m)$

- **authentication:**
  computing the "RSA decryption" of the hash value

  $red = h(m)^d \bmod n$

- **verification:**
  - computing the "RSA-encryption" of the cryptographic exhibit

    $red^{\,e} \bmod n$

    to recover the presumable hash value
  - comparing the result

    with the freshly recomputed hash value of the received message *m*

RSA digital signatures: underlying sets

- **messages:**
  bit strings over the set {0, 1}
  that can be mapped by the agreed one-way hash function *h*
  to bit strings basically of length ld *n*
  (positive numbers less than *n* (residues modulo *n*))

- **cryptographic exhibits:**
  bit strings over the set {0, 1},
  basically of length ld *n*
  (positive numbers less than *n* (residues modulo *n*))

- **keys:**
  given the public key (*n*, *e*),
  in principle there is a unique residue modulo *n*
  that can be used as the private decryption exponent *d* ,
  whose binary representation is a bit string, basically of length ld *n* or less;
  (from the point of view of the nondistinguished participants,
  this decryption exponent cannot be "determined")

RSA digital signatures: three algorithms

- ***key generation* algorithm *Gen*:**
  same as for RSA encryption
- ***authentication* (*signature*) algorithm *Aut*:**
  - takes a message *m* of an appropriate length
  - computes $h(m)$, where *h* is an agreed *one-way hash function*
  - returns $red = h(m)^d \bmod n$
- ***verification* algorithm *Test*:**
  - takes the received cryptographic exhibit *red*
  - computes $hash := red^{\,e} \bmod n$
  - takes the received message *m*
  - determines its hash value $h(m)$
  - checks whether this (correct) hash value equals the (received) value *hash*:
    $Test\,((n, e), m, red)$ returns *true* iff $h(m) = red^{\,e} \bmod n$

RSA digital signatures: fundamental properties

- to be considered: *correctness*, *unforgeability* and *efficiency*
- basic aspects of these properties can be derived like for RSA encryption
- regarding *correctness*:
  the commutativity of multiplication and exponentiation, i.e.,
  for all b, e_1, e_2:

  $$(b^{e1})^{e2} = b^{e1 \cdot e2} = b^{e2 \cdot e1} = (b^{e2})^{e1},$$

  is inherited by
  - encryption function $x^e \bmod n$
  - decryption function $y^d \bmod n$
- these functions are mutually inverse, independent of the application order

RSA encryption and digital signatures

- any *commutative* (asymmetric) *encryption* mechanism with encryption algorithms *Enc* and *Dec* that satisfy,

  for all plaintexts or ciphertexts x and for all keys (ek, dk)

  $$Dec\ (dk, Enc\ (ek, x)) = Enc\ (ek, Dec\ (dk, x))$$

  can be converted into an *authentication* (*signature*) *mechanism*

- **authentication:** $Aut\ (dk, x) = Dec\ (dk, x)$,

  using the private decryption key *dk* as the authentication key

- **verification:** $Test\ (ek, x, red) = true$ iff $x = Enc\ (ek, red)$, using the public encryption key *ek* as the test key

- *correctness* of the authentication

  is implied by the encryption correctness:

  $Enc\ (ek, Aut\ (dk, x)) = Enc\ (ek, Dec\ (dk, x)) = Dec\ (dk, Enc\ (ek, x)) = x$

- *unforgeability* is implied by the secrecy of the encryption

ElGamal asymmetric block cipher

- is another well-known example of the *one-way function* approach
- is based on ElGamal functions and their properties
- is *asymmetric*, admitting *multiple* key usage
- operates *blockwise*, where the block length is determined by the parameters of the underlying ElGamal function
- achieves *complexity-theoretic security*, provided:
  - the *discrete logarithm conjecture* and the *ElGamal conjecture* hold
  - the key is properly generated and sufficiently long
  - some additional care is taken

Asymmetric block ciphers based on elliptic curves

- are increasingly important examples of the *one-way function* approach
- are based on generalized ElGamal functions that
  are defined over appropriately constructed finite cyclic groups
  derived from elliptic curves based on a finite field
- are *asymmetric*, admitting *multiple* key usage
- operate *blockwise*, where the
  block length is determined by the parameters of the underlying elliptic curve
- achieve *complexity-theoretic security*, provided:
  - the pertinent *discrete logarithm conjecture* and related conjectures hold
  - the key is properly generated and sufficiently long
  - some additional care is taken
- offer a large variety of alternatives to the still predominant RSA approach, and thus diminish the dependence on the special unproven conjectures
- promise to achieve the wanted degree of secrecy
  with improved efficiency in comparison with the RSA approach

Asymmetric authentication by ElGamal and elliptic curves

- similar to encryption