Asymmetric Primitives

(public key encryptions and digital signatures)

An informal, yet instructive account of asymmetric primitives …

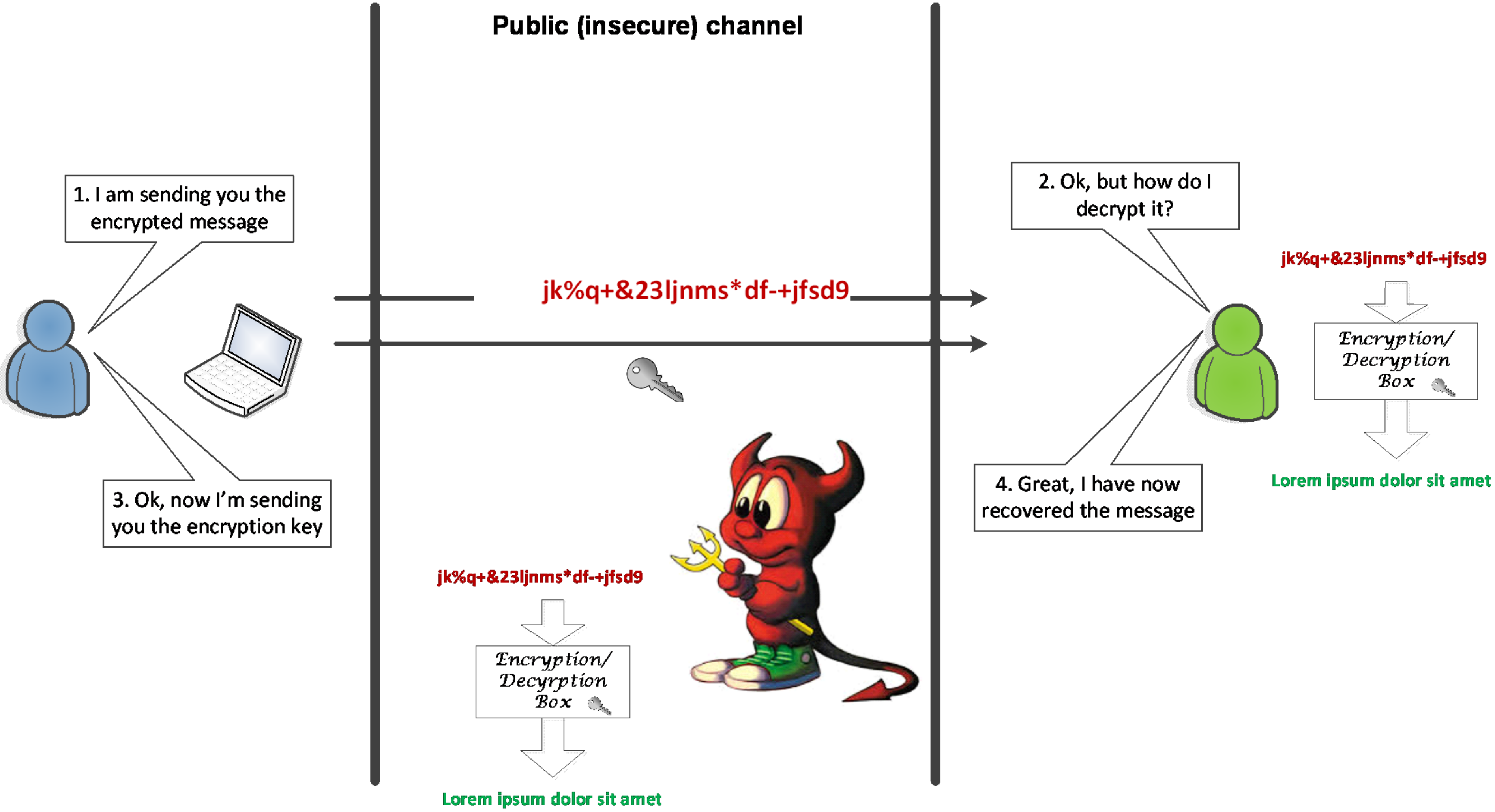
Timeline of the invention of public-key cryptography

- *1970-1974 British cryptographers James Ellis and Clifford Cocks from GCHQ invent the possibility of non-secret key encryption and the RSA*
- 1974 Ralph Merkle invented a public-key agreement that was published only in 1978
- 1976 Withfield Diffie and Martin Hellman, influenced by Ralph Merkle's work, published a method for public-key agreement (known as Diffie-Hellman key exchange, or Diffie-Hellman-Merkle key exchange)
- 1977 Ron Rivest, Adi Shamir and Leonard Adleman invent the RSA, published in 1978
- 1979 Michael O. Rabin publishes the Rabin cryptosystem, a public key cryptosystem with security equivalent to factoring
- 1985 Taher ElGamal published a method for encrypting and signing based on DHM key exchange
- 1985 Neal Koblitz and Victor Miller independently and simultaneously introduce elliptic curve cryptography

Why we need public-key cryptography?

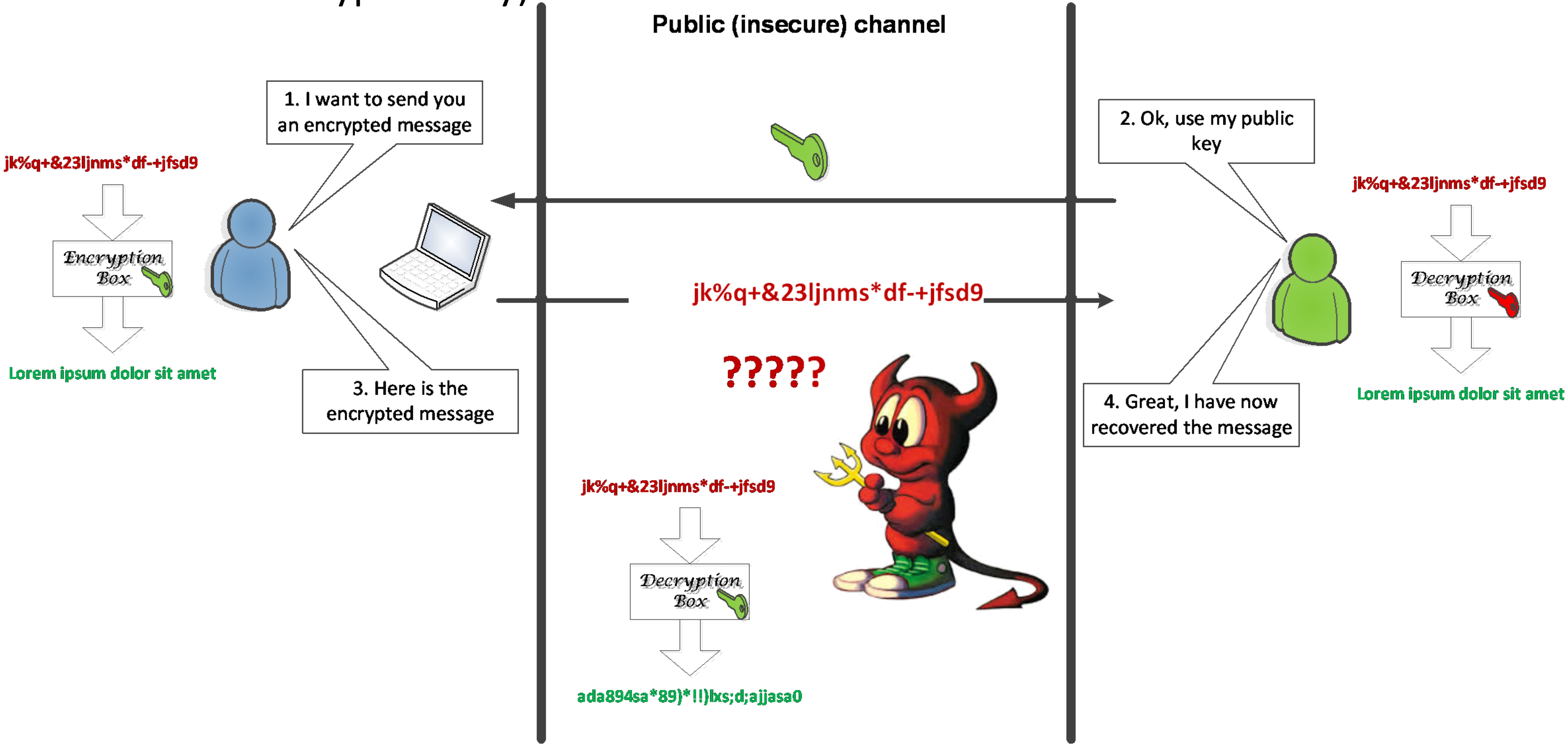
- Answer: Exchanging information securely over an insecure channel in the absence of a secretly shared key

- All symmetric key cryptosystems require a key to be shared between parties
- But in the real-world communication happens spontaneously between parties that did not interact before (i.e., previously shared secrets do not exist) and exchanging a secret key securely over a public channel (e.g., Internet) is not possible



How public-key encryption works (informal)

- Use separate key for encryption and decryption (note that the decryption key must not be recoverable from the encryption key)



Public (insecure) channel

1. I want to send you an encrypted message

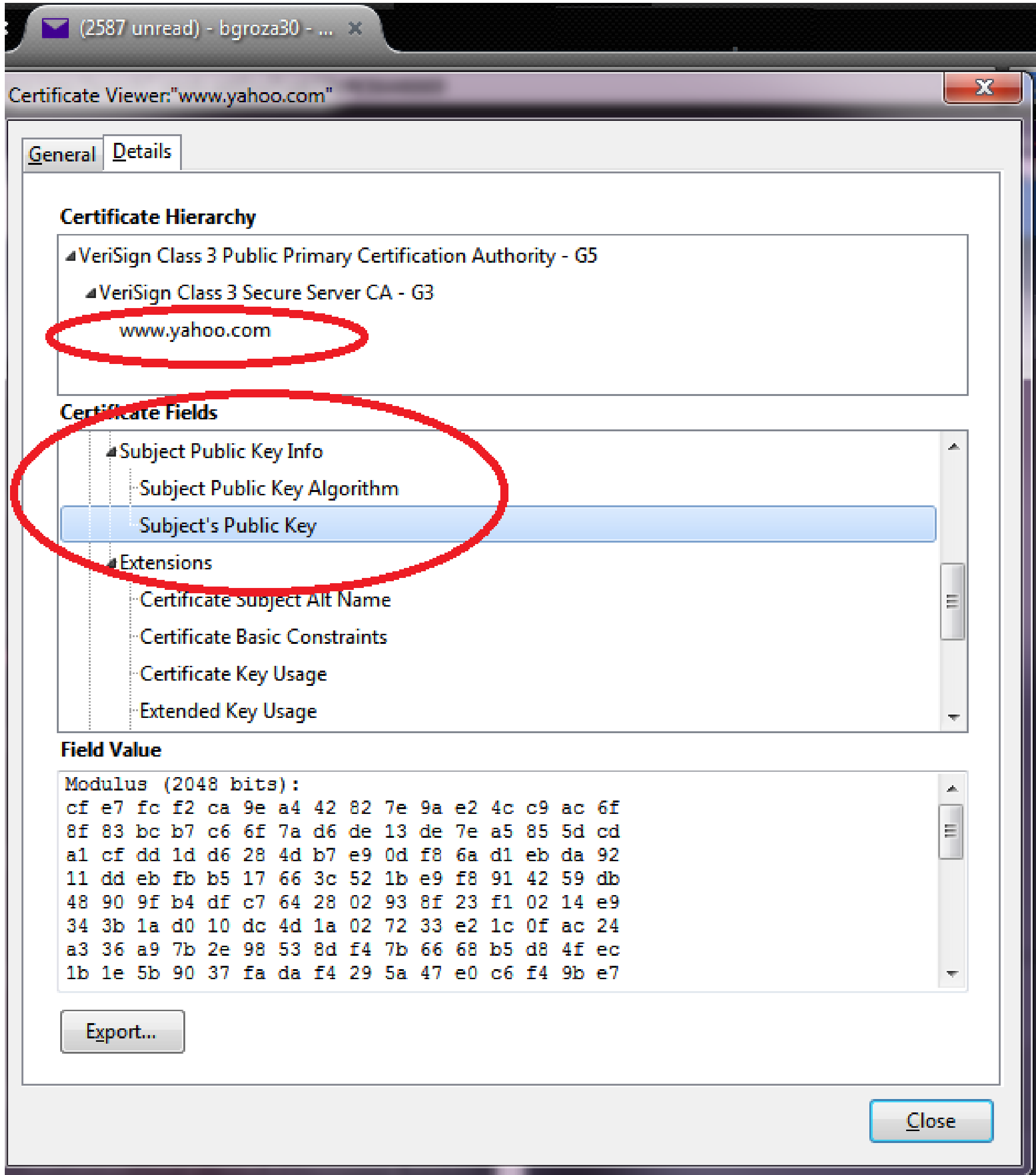
2. Ok, use my public key

3. Here is the encrypted message

4. Great, I have now recovered the message

Where is public-key encryption used?

- Used everywhere, examples:
  - In your browser: HTTPS, or HTTP over SSL/TLS, whenever you are using the Hypertext Transfer Protocol Secure (HTTPS) to privately read your e-mail, browse, chat or whatever …
  - Behind your routers: IPSEC
  - Etc.



```
Modulus (2048 bits):
cf e7 fc f2 ca 9e a4 42 82 7e 9a e2 4c c9 ac 6f
8f 83 bc b7 c6 6f 7a d6 de 13 de 7e a5 85 5d cd
a1 cf dd 1d d6 28 4d b7 e9 0d f8 6a d1 eb da 92
11 dd eb fb b5 17 66 3c 52 1b e9 f8 91 42 59 db
48 90 9f b4 df c7 64 28 02 93 8f 23 f1 02 14 e9
34 3b 1a d0 10 dc 4d 1a 02 72 33 e2 1c 0f ac 24
a3 36 a9 7b 2e 98 53 8d f4 7b 66 68 b5 d8 4f ec
1b 1e 5b 90 37 fa da f4 29 5a 47 e0 c6 f4 9b e7
```

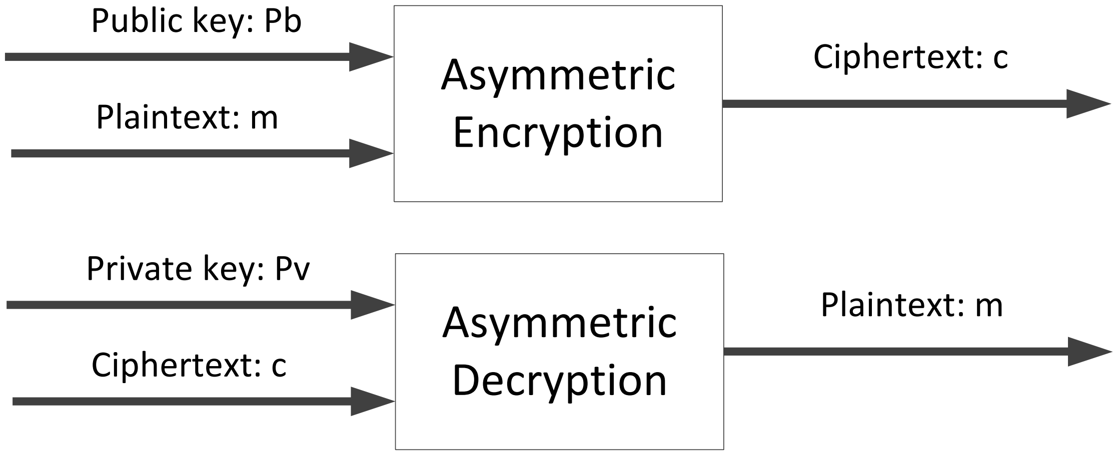
A more formal and constructive account of asymmetric primitives …

you should learn:

i. where is the primitive is used,
ii. what are the standards,
iii. how is it built,
iv. what are its properties

Type of functions (I) Asymmetric encryption schemes

- Description (informal): an algorithm that takes as input a public key (Pb) and message (m, called plaintext) and returns the encrypted message (c, called ciphertext), and a decryption algorithm that takes as input a private key (Pv) and ciphertext (c) and returns the message (m) (a key generation algorithm is also needed)



i.e., $c = e_{Pb}(m)$

i.e., $m = d_{Pv}(c)$

- Example of use: key-exchange for encrypted tunnels SSL/TLS, IPSEC, etc.
- Standards:
  - To use: RSA (2048 bit or above), Diffie-Hellman (with or without ECC)
  - Not to use: small key versions or unpadded (textbook) versions of the above
  - Future use: ECC to completely replace RSA (?)

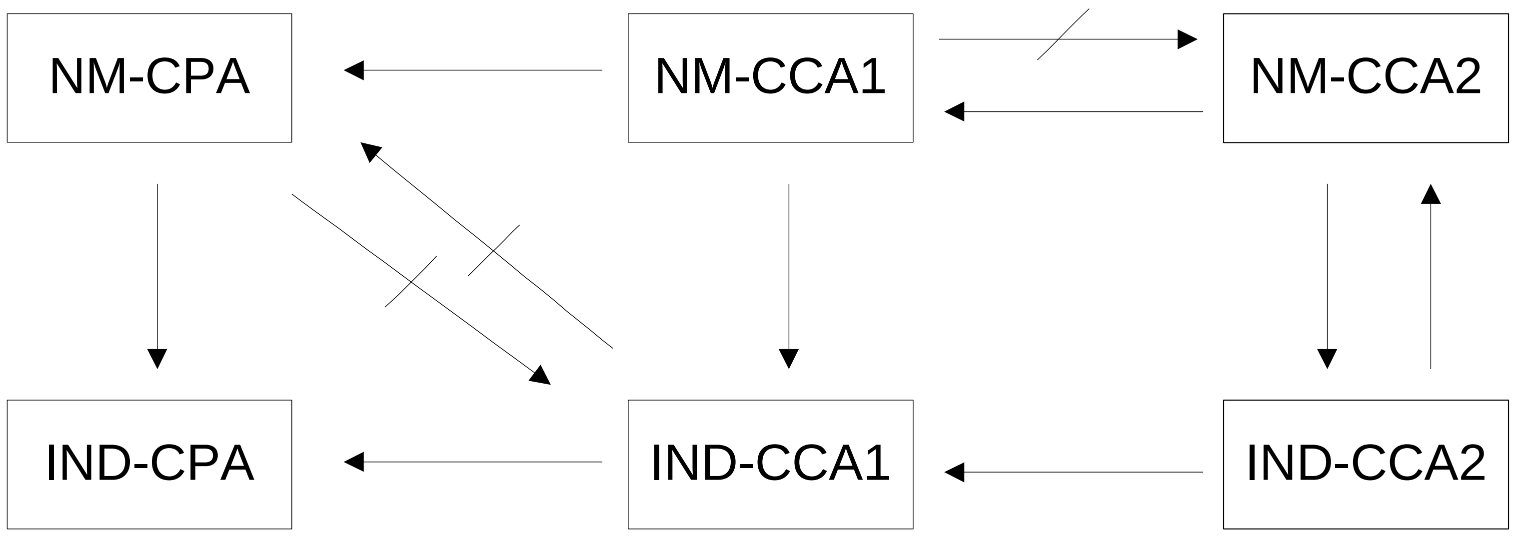
Asymmetric encryption: formal definition

- A symmetric encryption scheme is a triple of algorithms:
  - *Gen is the key generation algorithm that takes the security parameter l, random coins and outputs the public and private key* $(Pb, Pv) \leftarrow Gen(1^l)$
  - *Enc is the encryption algorithm that takes as input the public key and the message, then outputs the ciphertext* $c \leftarrow Enc(Pb, m)$
  - *Dec is the decryption algorithm that takes as input the ciphertext and the private key and outputs the message* $m \leftarrow Dec(Pv, c)$

- A correctness condition enforces that $Dec\big(Pv, Enc(Pb, m)\big) = m$
- A security condition enforces that given the public key *Pb* it is infeasible to compute the private key *Pv,* but this is not enough (remember SS/IND/NM security properties)

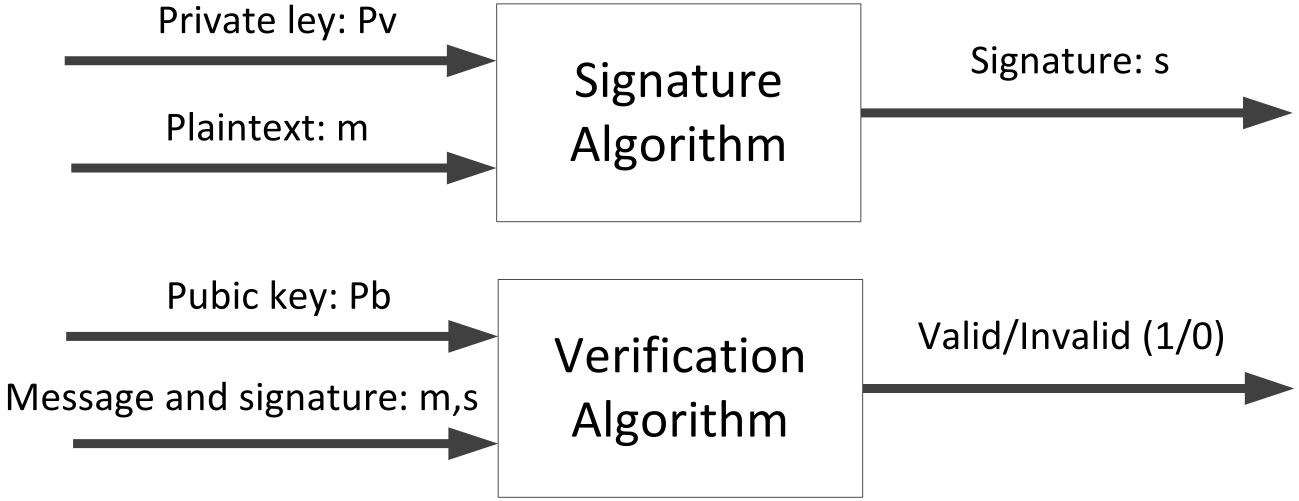
What are the desired security properties for PKC?

- Similar to what we defined in case of symmetric encryptions: active adversaries (CPA/CCA) and IND/NM:
  - IND – indistinguishability of ciphertexts – what you already know from symmetric cryptosystems
  - NM – non-malleability of ciphertexts – the adversary cannot modify a given challenge ciphertext such that it decrypts to a valid plaintext
- Pictured below are relations among security notions for PKC as proved by Bellare, Desai, Pointcheval & Rogaway ‘1998



Type of functions (II) Digital signatures

- Description (informal): the electronic "equivalent" of a handwritten signature, the signing algorithm takes the private key and message and returns a signature, the verification algorithm takes the public key, message and signature and checks if the input is genuine. (a key generation algorithm is also needed)



i.e., $s = Sig_{Pv}(m)$

- Example of use: document signing, driver signing, public-key certificate signing, SSL/TLS, etc.
- Standards:
  - To use: RSA-PSS, RSA-FDH, RSA-PKCS
  - Not to use: small key versions of the above or unpadded (textbook) versions
  - Future use: N/A

Digital signatures: formal definition

- A symmetric encryption scheme is a triple of algorithms:
  - *Gen is the key generation algorithm that takes random coins, the security parameter l and outputs the public and private key* $(Pb, Pv) \leftarrow Gen(1^l)$
  - *Sig is the signing algorithm that takes as input the private key and the message, then outputs the signature* $s \leftarrow Sig(Pv, m)$
  - *Ver is the verification algorithm that takes as input the signature and the public key and outputs the 1 if the signature is valid or 0 otherwise* $\{0,1\} \leftarrow Ver(Pb, s, m)$

- A correctness condition enforces that $Ver\big(Pb, Sig(Pv, m)\big) = 1$
- A security condition enforces that given the public key *Pb* it is infeasible to compute the private key *Pv*, but this is not enough (see security properties)

What do we mean by breaking a signature?

- *Existential forgery* – find a valid message-signature without controlling the message
- *Selective forgery* – forge signature over messages that have a particular structure
- *Universal forgery* – forge signatures over any kind of messages (without knowing the private key)
- *Total break* – recover the private key (sign anything)

What are the adversary capabilities?

- *Key-only* – adversary knows only the public key
- *Known-messages* – adversary has valid messages-signature pairs but not at his choice
- *Chosen message* – adversary has messages-signature pairs at his choice (adaptive chosen-message is a flavor of this notion where the adversary is allowed to chose messages after fixing the target to be forged)

To sum up: **unforgeability under chosen-message attacks** is the desired property (adversary cannot forge signatures, even if he has full access tot the signing oracle)

Fundamentals - Number Theory (in 1 slide)

- Definition: A set A together with some operation × forms an abelian group if the operation × is:
  - i. associative, i.e., $(\mathrm{a} \times \mathrm{b}) \times \mathrm{c} = \mathrm{a} \times (\mathrm{b} \times \mathrm{c})$,
  - ii. comutative, i.e., $\mathrm{a} \times \mathrm{b} = \mathrm{b} \times \mathrm{a}$,
  - iii. there exists an identity element e such that $\mathrm{e} \times \mathrm{a} = \mathrm{a} \times \mathrm{e} = \mathrm{a}$,
  - iv. each element a has an inverse b such that $\mathrm{a} \times \mathrm{b} = \mathrm{b} \times \mathrm{a} = \mathrm{e}$.
- $Z_n = \{0,1,2,\dots,n-1\}$ is called the set of integers modulo n, i.e., remainders mod n, then $(Z_n, +)$ forms an abelian group
- $Z_n^* = \{x \in Z_n \,|\, \gcd(x,n) = 1\}$ is the set of integers modulo n that are relatively primes to n, then $(Z_n, *)$ forms an abelian group
- The Euler's totient function function is defined as $\varphi(n) = |Z_n^*|$, that is $\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \dots \left(1 - \frac{1}{p_r}\right)$ where $p_1, \dots, p_r$ are the prime factors of n
- Euler's Theorem – strong result that builds the RSA trapdoor

$$\forall x \in Z_n^*, x^{\varphi(n)} \equiv 1 \bmod n$$

Tools: Computational Number Theory (in 1 slide)

- The following computational problems make public key trapdoors possible, to build public key trapdoors we need both problems that can be efficiently solved (encryption and decryption, i.e., the cryptosystem is efficient) and problems that cannot be efficiently solved (finding the private key from the public key, i.e., breaking the cryptosystem is hard)

Efficiently Computable	Prerequires
Elementary operations in Z_n^*: -, +, *, /, a^x	-
Greatest common divisor (GCD) and multiplicative inverse, i.e., x^{-1}	-
Primality testing	-
Square root in Z_n^*, i.e., $\sqrt[2]{x} mod n$	**If and only if** factorization known
e-th root in Z_n^*, i.e., $\sqrt[e]{x} mod n$	**If** factorization known
Systems of simultaneous congruences over co-primes (Chinese Remaindering Theorem)	-

Not Efficiently Computable	Prerequires
Logarithms, i.e., $log_a(a^x) mod$ p	Order of the group sufficiently large
Factorization of an integer	Large integers with non-trivial factors
Square root in Z_n^*, i.e., $\sqrt[2]{x} mod n$	If factorization is not known
e-th root in Z_n^*, i.e., $\sqrt[e]{x} mod n$	If factorization is not known

RSA public key cryptosystem

- **Key generation**

1. Generate two random primes p,q
2. Compute $n=pq$, $\varphi(n)=(p-1)(q-1)$
3. Choose e relatively prime to $\varphi(n)$
4. Compute d such that $ed\equiv 1 \bmod \varphi(n)$
5. Public key is $Pb=(n,e)$ and private key $Pv=(n,d)$

- **Encryption**

1. Obtain the public key $Pb=(e,n)$
2. Compute $c=m^e \bmod n$, (note that the message must be represented as integer mod n)

- **Decryption**

1. Receive the encrypted message c
2. Compute $m=c^d \bmod n$ by using the private key Pv

- Example (with artificially small numbers)

- **Key generation**

$$p=11,\ q=13,$$

$$n=p\cdot q=143, \phi(n)=(p-1)\cdot(q-1)=120$$

$$e=7,\ d=103,$$

$$Pb=(7,143), Pv=(103,143)$$

- **Encryption**

$$m=5$$

$$c=m^e \bmod n=5^7 \bmod 143=47$$

- **Decryption**

$$c=47$$

$$m=c^d \bmod n=47^{103} \bmod 143=5$$

Real World RSA Keys

- 2048 bit RSA key from RSA factoring challenge (offered 200.000$ for its factors)

25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357

- Question: Consider to factor by exhaustive search? What is the expected number of steps?
- You should take a look at the following:
  - *electron**s** in universe:*
    $8{,}37*10^{77}\approx 83700000000000000000000000000000000000000000000000000000000000000000000000000000$
  - age of solar system: $1{,}89*10^{17}\approx 189000000000000000$

- Need more motivation? The following rewards were withdrawn by RSA, but still …

RSA-768	**$50,000 USD**	**(factored December 12, 2009)**
RSA-896	**$75,000 USD**	
RSA-1024	**$100,000 USD**	
RSA-1536	**$150,000 USD**	
RSA-2048	**$200,000 USD**	

RSA Computational requirements in brief

- Generating keys is the most intensive computational step as generation of two random primes requires: generating a random integer + testing for primality (there are ~x/ln(x) prime numbers up to x, so probability of success is ~1/ln(x))
- Encryption is usually the most efficient step since one can choose special form exponents: 3, 5, 65537 (note that primes of the form 1000…0001 are preferred)
- Decryption is always more computationally intensive than encryption because the decryption exponent is in the order of the modulus n
- Questions: why are exponents of the form 100…001 preferred? Why is the decryption exponent in the order of n?

RSA CRT speed-up

- For faster computations, RSA decryption is usually performed with Chinese-Remaindering-Theorem
- This allows performing decryptions modulo p and q then combines them to get the result

$$\begin{cases} m_1 = c^{d_1} mod\ p \\ m_2 = c^{d_2} mod\ q \end{cases} \quad \Longrightarrow m = m_1 q(q^{-1} mod\ p) + m_2 p(p^{-1} mod\ q)$$

- where $d_1 = d\ mop\ (p-1)$ and $d_2 = d\ mop\ (q-1)$
- Questions: why is the decryption exponent reduced mod p-1 and q-1? Why this works faster than standard decryption?
- Note: there are alternative ways for doing the same, e.g., see in .NET implementation

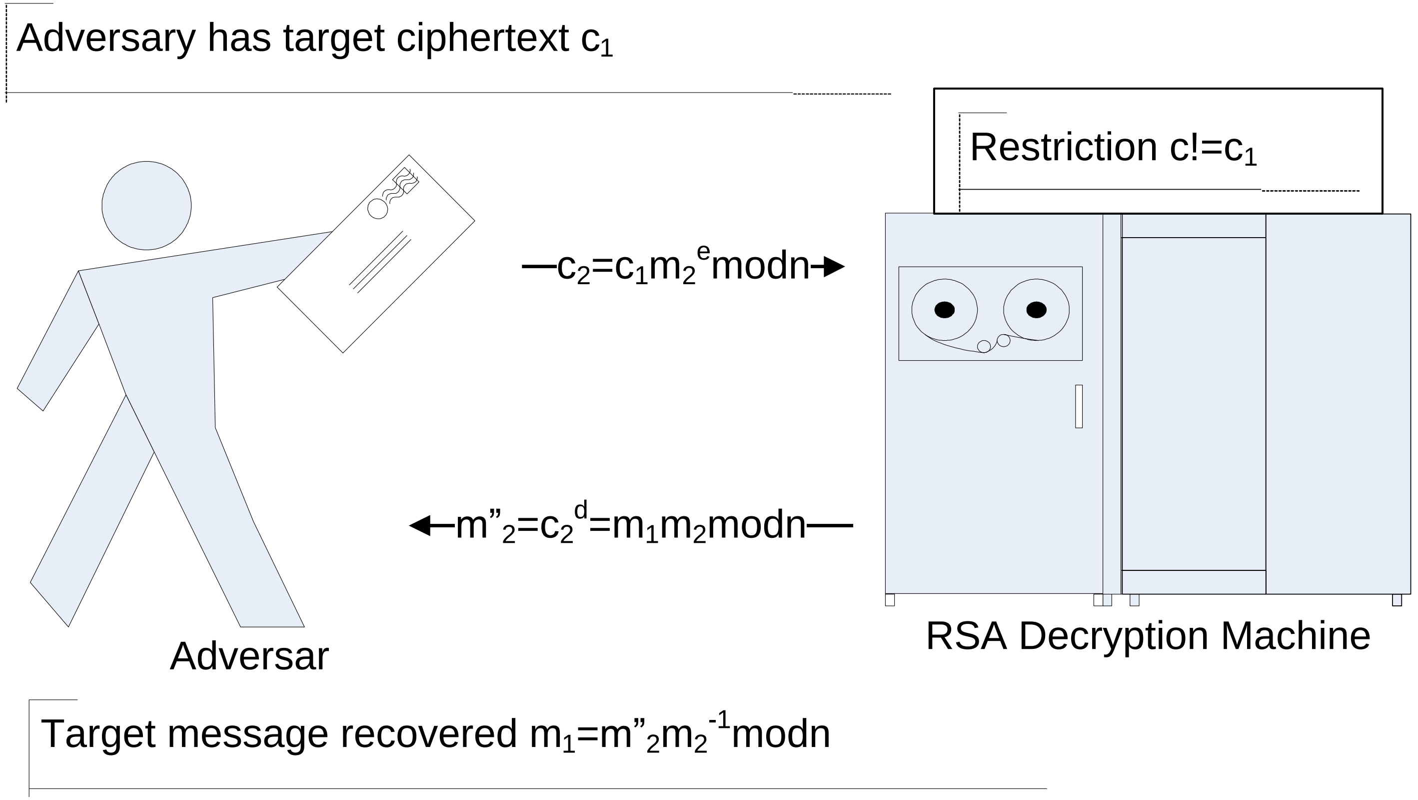
Mathematical security & properties (or vulnerabilities?)

- Relation between RSA and Factoring: no proof of equivalence between breaking RSA and factoring exists so far, some facts:
    - ❖Factoring obviously leads to breaking the RSA
    - ❖Computing a private-public RSA key pair also leads to factoring (discussed in laboratory exercises)
    - ❖Proving that RSA decryption leads to factoring seems to be hard (or maybe this equivalence is not true after all)

- Many interesting properties behind the text-book RSA trapdoor, some of them opening door for attacks (all these will be discussed in laboratory exercises):
    - ❖Small messages
    - ❖Small encryption exponents
    - ❖Small decryption exponents
    - ❖Messages that do not encrypt

Why text-book RSA fails in front of active adversaries?

- Question: Consider IND (indistinguishability) as security property, is textbook RSA secure under this property?
- Answer: No, in fact no deterministic public key cryptosystem is.
- Question: Consider an CCA adversary, can the adversary recover the full plaintext in case of textbook RSA?
- Answer: Yes, textbook RSA is completely insecure under CCA adversaries



Adversary has target ciphertext c_1

Restriction $c!=c_1$

—$c_2=c_1m_2^e$modn→

←$m''_2=c_2^d=m_1m_2$modn—

Adversar

RSA Decryption Machine

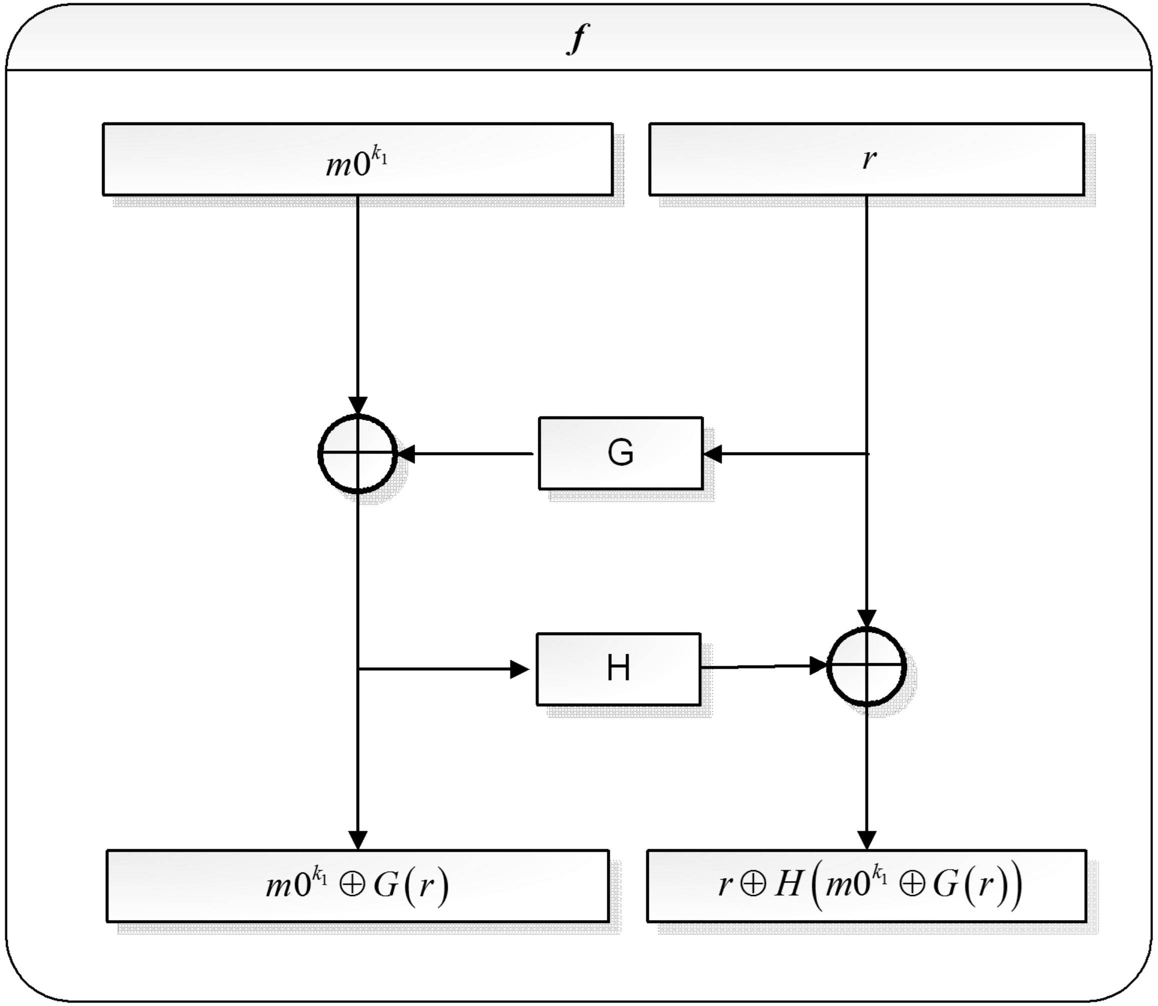
Target message recovered $m_1=m''_2m_2^{-1}$modn

Secure versions of RSA: RSA-OAEP

- Bellare & Rogaway 1991
- Main idea: embed a Feistel network under RSA:

$$E(x) = f\left(x \oplus G(r) \,\|\, r \oplus H\left(x \oplus G(r)\right)\right)$$

- OAEP has provable NM/IND security under CCA adversaries
- Some historical turnarounds for OAEP:
  - Bellare & Rogaway proved that OAEP gives security on any trapdoor
  - Shoup proved they were wrong
  - Fujisaki & Okamoto proved that security holds for RSA
  - All proofs are in the Random Oracle Model but hash functions in practice are not random oracles



Introducing RSA-PKCS#1

- RSA encryption according to PKCS#1 (Public-Key Cryptography Standards)
- Before encryption, message is padded as:

$$\left(00\ldots00 \,\|\, 00\ldots10 \,\|\, random \,\|\, 00\ldots00 \,\|\, m\right)^{e} \bmod n$$

- Note: the random number below has k-3-|m| bytes (at least 8) where k is the byte length of the modulus

- Good news: previous CCA attacks does not work, can be (somewhat) securely used in practice
- Bad news: there are some attacks for special cases (small exponents, special messages, etc.), and more, there is no proof that RSA-PKCS#1 is secure
- Good news: newer versions of PKCS#1 include RSA-OAEP as improved encryption/decryption method

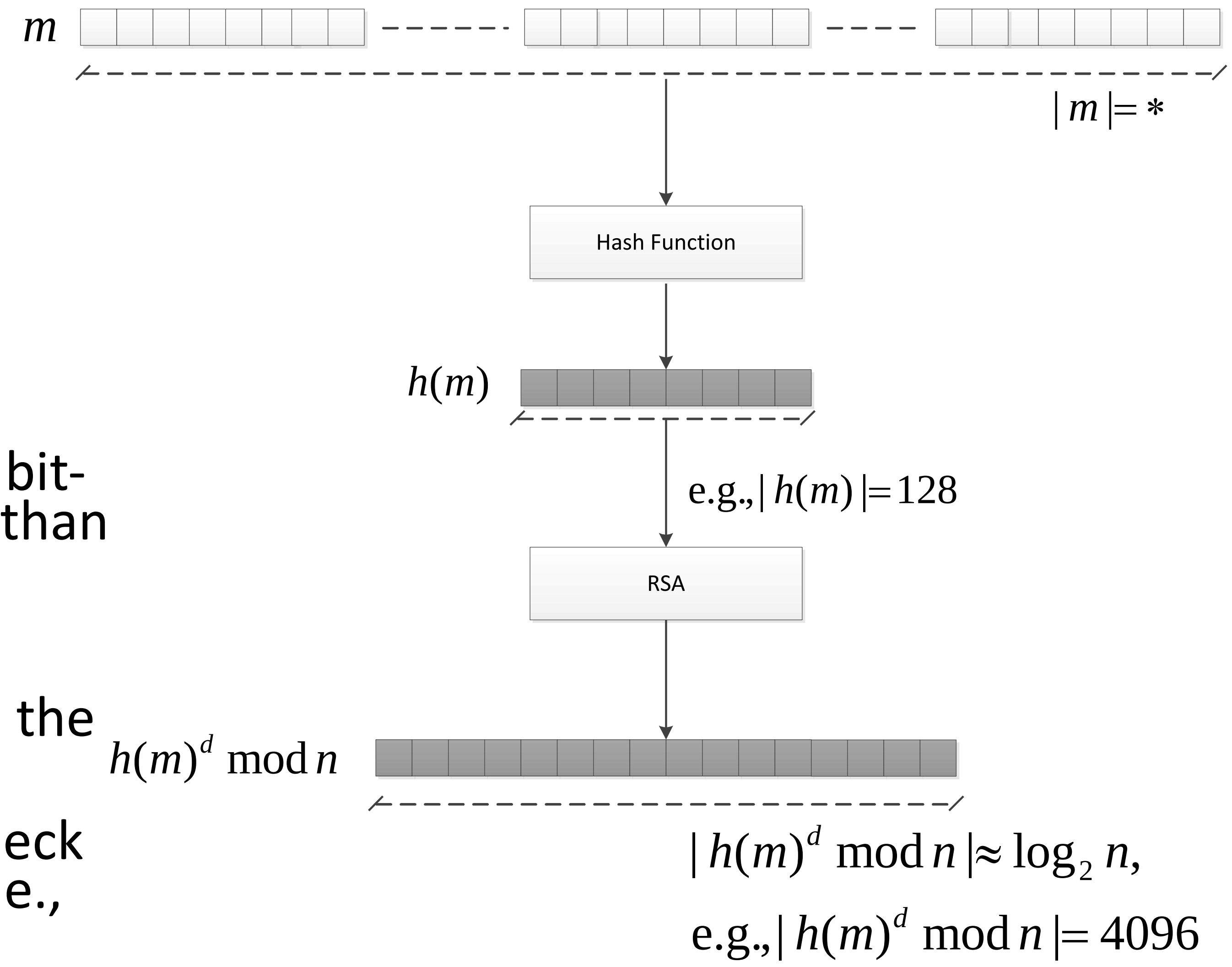
The textbook RSA signature (hash then sign)

- Principle:
  - ➢To sign: hash the message then use the private key to sign the hash
  - ➢To verify: use the public key to recover the hash then compare it to the hash of the original message
- **Sign**

1. Compute $s=H(m)^d \bmod n$, (note that the bit-length of the hash must be less or equal than that of the modulus n)

- **Verify**

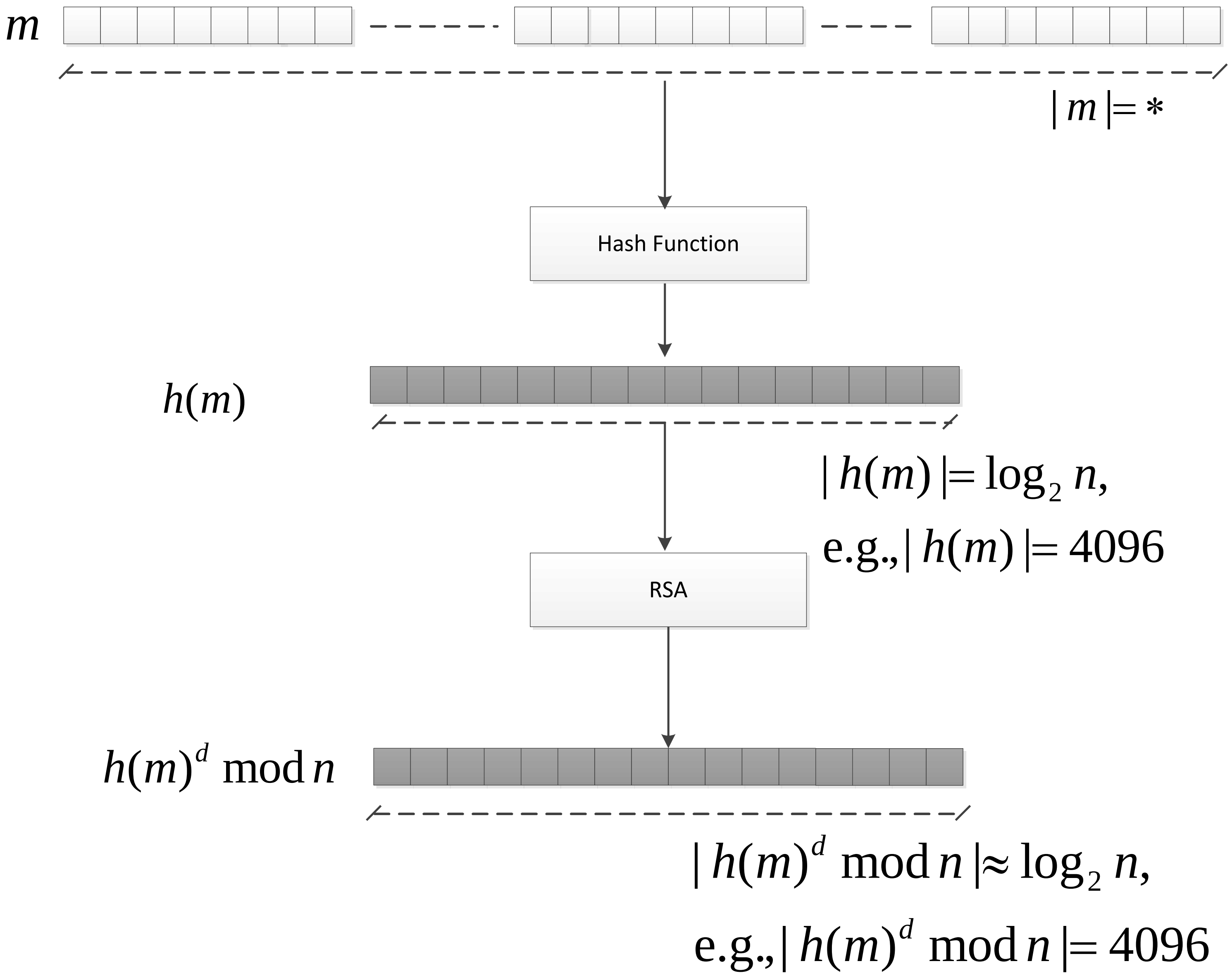
1. Recover the hash from the signature with the help of the public key $h'=s^e \bmod n$
2. Compute the hash of the message and check that it is equal with the recovered hash, i.e., $h'=H(m)$



- Note: in case of RSA the signing algorithm is the reverse of encryption algorithm, this leaves the impression that in general signing is the reverse of encryption, but turns out not to be the case for many other public key cryptosystems, e.g., ElGamal

RSA – Full Domain Hash (FDH)

- **Principle**: use a hash function that spans over the entire domain of the modulus
- **Security:** RSA-FDH is provable secure in the Random-Oracle-Model


m

$|m| = *$

$h(m)$

$|h(m)| = \log_2 n,$

$\text{e.g.}, |h(m)| = 4096$

$h(m)^d \bmod n$

$|h(m)^d \bmod n| \approx \log_2 n,$

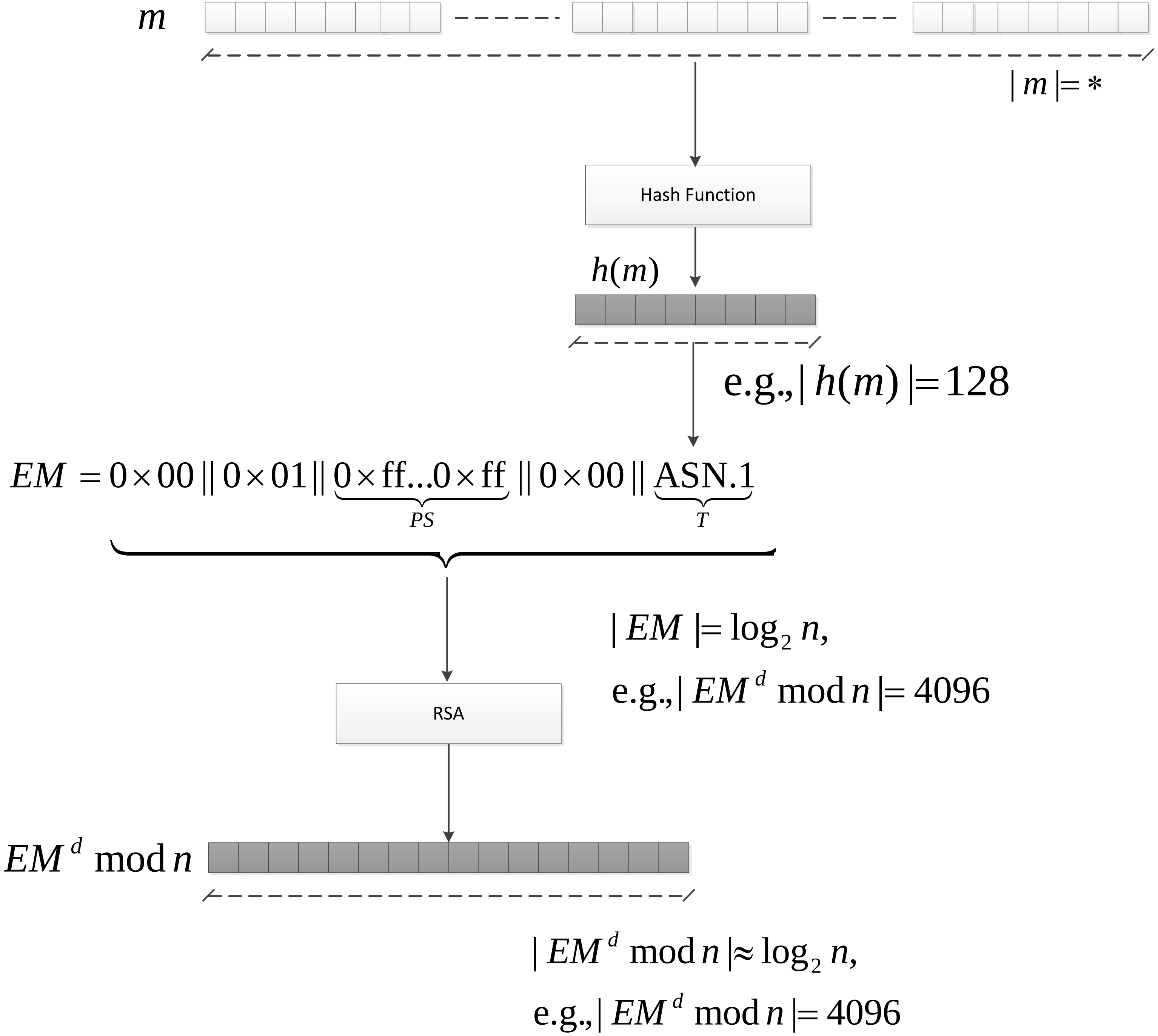
$\text{e.g.}, |h(m)^d \bmod n| = 4096$

Proving RSA FDH security

- To be done as lecture and/or laboratory excercise

RSA – PKCS v.1.5

- Standard published by RSA laboratories as of 1991, current version is from 2012



e.g., $|h(m)| = 128$

$$EM = 0\times 00 \,\|\, 0\times 01 \,\|\, \underbrace{0\times \mathrm{ff} \ldots 0\times \mathrm{ff}}_{PS} \,\|\, 0\times 00 \,\|\, \underbrace{\mathrm{ASN.1}}_{T}$$

$|EM| = \log_2 n,$

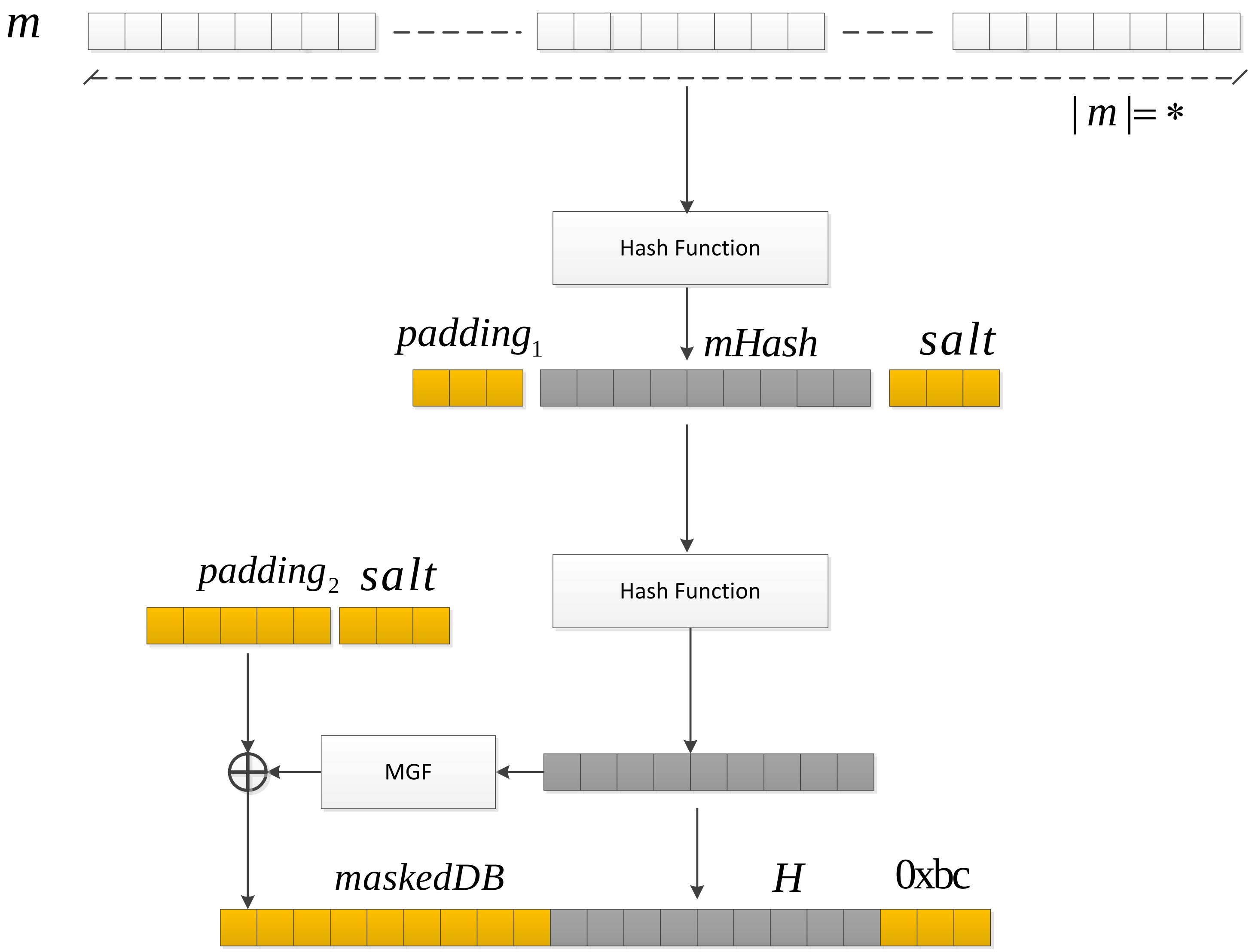
e.g., $|EM^d \bmod n| = 4096$

$|EM^d \bmod n| \approx \log_2 n,$

e.g., $|EM^d \bmod n| = 4096$

RSA – Probabilistic Standard Signature (PSS)

- Designed by Bellare & Rogaway, also included in newer versions of PKCS



The Rabin cryptosystem

- Published in ‘79 by M.O. Rabin

- Key generation

1. Generate two random primes p,q
2. Fix e = 2
3. Public key is $Pb=(2,n)$ and private key $Pv=(p,q)$

- Encryption

1. Obtain the public key $Pb=(2,n)$
2. Compute $c=m^2 \bmod n$

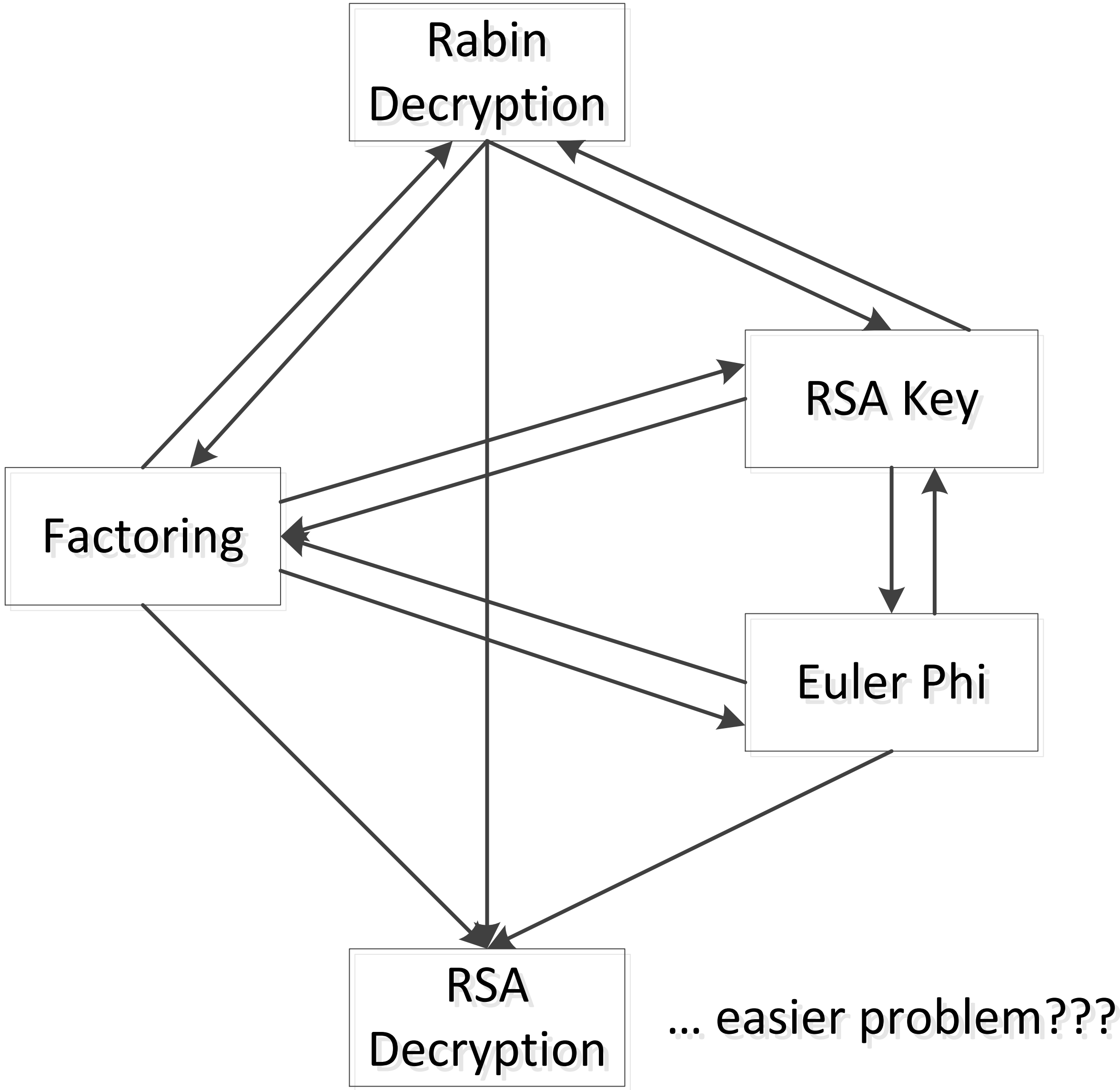
- Decryption

1. Compute m as the square root of c

- Notes:
  - Rabin is not a particular case of RSA, 2 cannot be an RSA encryption exponent
  - Requires padding similar to the RSA to be secure
  - If the modulus is the product of two primes then there are 4 square roots (need redundancy/padding to decide which of them was the message)
- Question: why 2 cannot be an RSA exponent? Why are there 4 roots?

Recap: computational problems behind factoring based schemes

- All problems seem to nicely reduce one to another: Factoring, Rabin Decryption, RSA Key Generation and Euler Phi computation
- Is just RSA Decryption for which there is no proof that it will allow solving the others
- Note: arrow from P1 to P2 means that if you could solve P1, you can solve P2



The Diffie-Hellman-Merkle Key exchange – The Discrete Logarithm Terrain

- Method for securely exchanging a key over an insecure channel between two parties

- Key setup

1. Fix a prime p
2. Choose a generator g of Z_p

- Exchange

1. $A \rightarrow B$: $g^a mod\ p\ n$ (a is a fresh secret random value)
2. $B \rightarrow A$: $g^b mod\ p$ (b is a fresh secret random value)

Where

- Compute

1. A computes $(g^b)^a mod\ p = g^{ba} = g^{ab}$
2. B computes $(g^a)^b mod\ p = g^{ab}$

- Notes:
  - The protocol above is vulnerable to a man-in-the-middle attack (but it's trivial to derive secure versions of it)
  - The order of the group Z_p must have a large prime factor, usually one works with $p = 2q + 1$(this is usually called a safe prime)

ElGamal encryption

- Key generation

1. Generate a random prime p
2. Choose a generator g
3. Choose a random value $a \in (1, p-2)$
4. Compute $g^a mod\ p$
5. Public key is $\mathrm{Pb} = (\mathrm{p}, \mathrm{g}, g^a)$ and private key is $\mathrm{Pv} = (\mathrm{p}, \mathrm{g}, a)$

- Encryption

1. Obtain the public key $\mathrm{Pb} = (\mathrm{p}, \mathrm{g}, g^a)$
2. Choose a random value $\mathrm{k} \in (1, p-2)$
3. Compute $c_1 = g^k mod\ p,\ c_2 = m(g^a)^k\ mod\ p$
4. Send $c = (c_1, c_2)$

- Decryption

1. Receive the encrypted message c
2. Recover the message as $\mathrm{m} = {c_1}^{-a} c_2$

- Remark:
  - Same remark for the order of the group as in the case of Diffie-Hellman
  - When computing $c_2 = m(g^a)^k\ mod\ p$ multiplication is used to conceal the message, but you can use other operations as well (XOR, symmetric encryption, etc., with the Diffie-Hellman key)

ElGamal Signature

- Published by Taher ElGamal in ‘84 (dlogs were used in crypto since the ’76 work of Diffie&Hellman, but a dlog signing scheme eluded for many years)

- Key generation

1. Generate a random prime p
2. Generate a random integer a $\in (1, p-2)$
3. Compute $y = g^a mod\ p$
4. Public key is $Pb = (g, y, p)$ private key is $Pv = (g, a, p)$

- Sign

1. Generate random $\mathrm{k} \in (0, p-1)$
2. Having h the hash of the messge, compute $r = g^k\ mod\ p$ and $\mathrm{s} = k^{-1}(h - ar) mod\ (p-1)$
3. Output the pair (r, s) as the signature

- Verify

1. Compute the hash of the message h
2. Verify that $r \in (0, p)$ and $s \in (0, p-1)$ return 0 if not
3. Verify that $g^h = y^r r^s$ return 1 if so or 0 otherwise

- Remarks:
  - Key generation is cheaper than for RSA (only one prime needed), more, the prime field can be a global parameter, i.e., more entities can use the same fixed p
  - Signing requires more computations but these are done over a prime p that is usually smaller than the RSA modulus, therefore its faster
  - Verification is slower than for RSA (if special public exponents are used, i.e., 65537, etc.)

ElGamal – notes on security

- So far there exist no security reductions (proofs) for ElGamal signatures, nor for DSA (next), Schnorr signature is the simplest dlog based signature that has a security reduction to the dlog problem but is quite absent in practice
- Selecting a random k is mandatory for the security of the ElGamal signature, if k is not random then the secret key is trivial to recover:

Let the first signature be

$$\{r_1 = g^k \, mod \, p, s_1 = k^{-1}(h_1 - ar) mod \, (p-1)\}$$

and the second

$$\{r_2 = g^k \, mod \, p, \; s_2 = k^{-1}(h_2 - ar) mod \, (p-1)\}$$

then

$$\mathrm{k} = (s_1 - s_2)/(h_1 - h_2)$$

now a can be recovered from any of s_1 , s_2

The Digital Signature Algorithm - DSA

- Also known as DSS – Digital Signature Standard, standardized by NIST
- It is a variation of the ElGamal signature, all previous remarks apply here as well
- It differs from ElGamal mostly at key generation and verification, resulting in smaller signatures (a small but true practical advantage)

- Key generation

1. Generate a random prime p such that another prime q of 160 bits divides $p-1$
2. Select a generator g of order q
3. Generate random a $\in (0, q-1)$
4. Compute $y = g^a mod\ p$
5. Public key is $Pb = (g, y, p)$ private key is $Pv = (g, a, p)$

- Sign

1. Generate random $\mathrm{k} \in (0, q-1)$
2. Having h the hash of the messge, compute $r = g^k\ mod\ p\ mod\ q$ and $\mathrm{s} = k^{-1}(h + ar) mod\ q$
3. Output the pair (r, s) as the signature

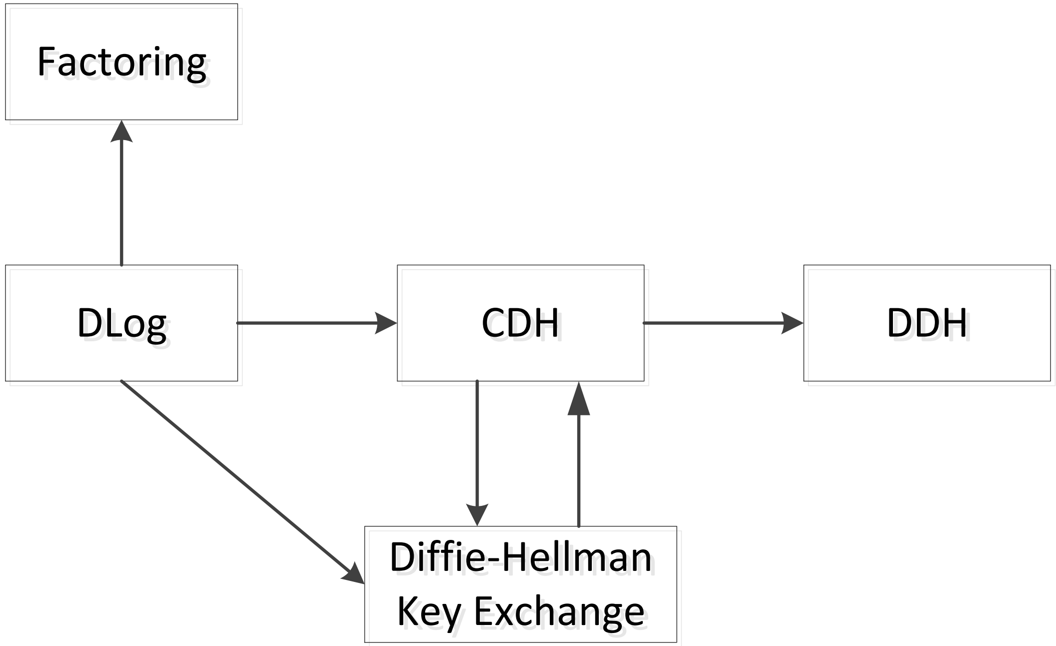
- Verify

1. Compute the hash of the message h
2. Verify that $r \in (0, q)$ and $\mathrm{s} \in (0, q)$ return 0 if not
3. Verify that $v = r$ and return 1 if so or 0 otherwise, where $v = (\ g^{u_1} y^{u_2} mod\ p) mod\ q$, $u_1 = wh\ mod\ q, u_1 = rw\ mod\ q$, $\mathrm{w} = s^{-1}\ mod\ q$

- Remark: parameter q here is fixed at 160 bits according to the output size of SHA1, it can be set to 224 and 256 for SHA2 (see FIPS 186-3)

Computational problems behind DLog based schemes

- All of the previous are apparently based on the difficulty of computing discrete logarithms, but there are three flavors of this problem:
  - Decisional Diffie-Hellman problem (DDH) – let $y_0 = g^{ab}, y_1 = r$, and β a random bit, given g^a, g^b, y_β find β (that is, distinguish between a complete random value and a DH key)
  - Computational Diffie-Hellman problem (CDH) – given g^a, g^b compute g^{ab}
  - Discrete Logarithms (DLog) – given g^a compute a
- The security of the Diffie-Hellman key exchange is equivalent to CDH (and at most as hard as DLog)
- If DLog can be computed Factoring is easy



More on digital signatures: message recovery

- All of the previous signatures worked with the hash of the message, these are usually called signatures with appendix
- Signatures with message recovery also exist, for example with RSA if the message is smaller than the modulus one can sign directly on the message, then recover it from the signature
    - Sign: compute $s = m^d \bmod n$, (note that the message must be smaller than the modulus n)
    - Verify: recover the message from the signature with the help of the public key $m = s^e \bmod n$
- Question: show an existential forgery on the above RSA signing scheme (to avoid such forgeries padding must be used).