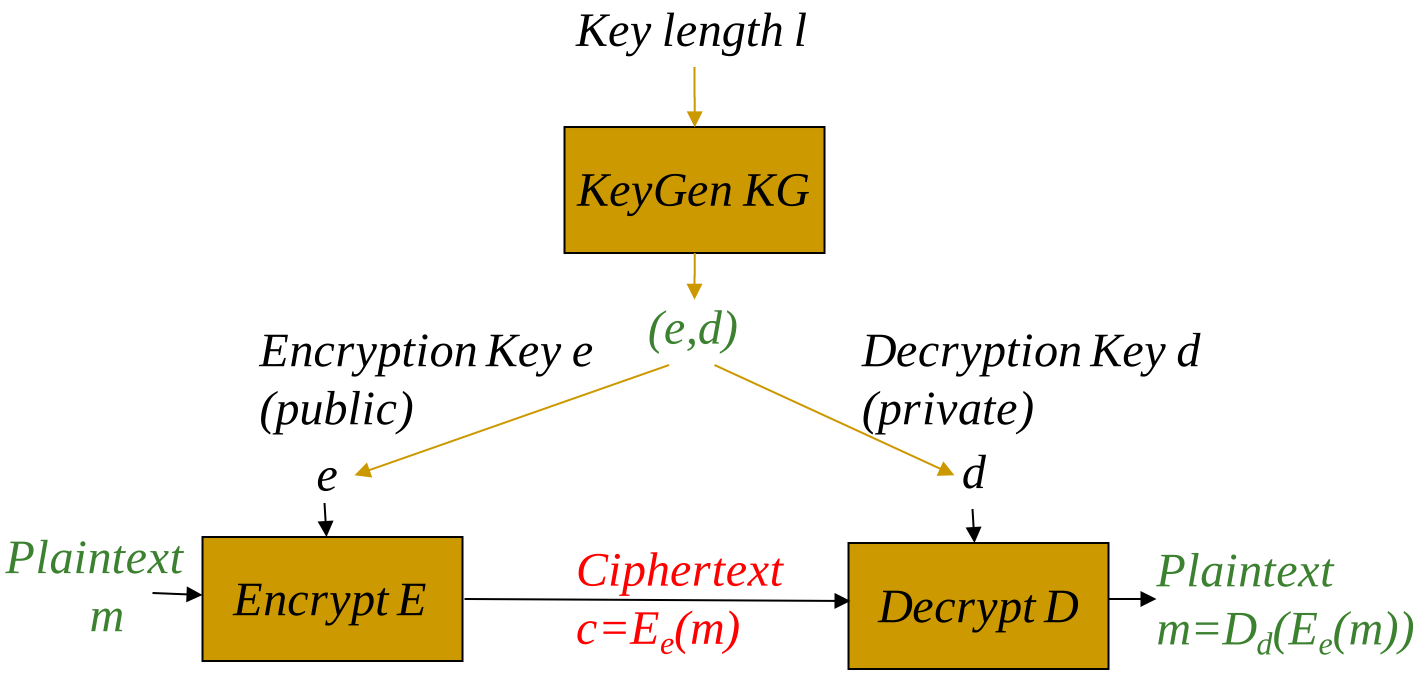
Public Key Cryptology, Part II: Public key cryptosystems and signature schemes

Last updated: Monday, March 23, 2020

Prof. Amir Herzberg
CSE Dept, Univ. of Connecticut

Public Key Cryptosystem



Using DH: for Encryption?

- Can we turn DH into… encryption?
- Bob **publishes** g^b as its public key
- Alice uses it (directly!) to encrypt messages for Bob
  - No interaction
- Let's see it gradually…

Turning [DH] to Public Key Cryptosystem

- Select random prime *p* and generator *g*
- Alice: secret key a, public key $P_A = g^a \mod p$
- Bob: secret key b, public key $P_B = g^b \mod p$

Turning [DH] to Public Key Cryptosystem

- Select random prime *p* and generator *g*
- Alice: secret key d_A public key ~~$P_A = g^a$ m~~ $e_A = g^{d_A} \; mod \; p$
- ~~Bob: secret key b, public key $P_B = g^b \; mod \; p$~~

  (Bob will encrypt – does not have keys)

Turning [DH] to Public Key Cryptosystem

- Select random prime *p* and generator *g*
- Alice: secret key d_A , public key $e_A = g^{d_A} \bmod p$
- ~~Bob: secret key b, public key $P_B = g^b \bmod p$~~
- To encrypt message m to Alice:
  - Bob selects random b
  - Sends: $g^b \bmod p$, $m \oplus h((e_A)^b) = m \oplus h(g^{b \cdot d_A} \bmod p)$
  - Secure if $h(g^{b \cdot d_A} \bmod p)$ is pseudo-random

Alice

$e_A = g^{d_A} \bmod p$ →

← $g^b \bmod p$, $m \oplus h(g^{b \cdot d_A} \bmod p)$

Bob

El-Gamal Public Key Cryptosystem

- Variant of [DH] PKC: Encrypt by multiplication, not XOR
- To encrypt message m to Alice, whose public key is $e_A = g^{d_A} \mod p$:
  - Bob selects random b
  - Sends: $g^b \mod p$, $m*(e_A)^b = m*g^{b \cdot d_A} \mod p$

Alice

$e_A = g^{d_A} \mod p$ $\longrightarrow$

$\longleftarrow$ $(g^b \mod p$, $m* e_A^b \mod p)$

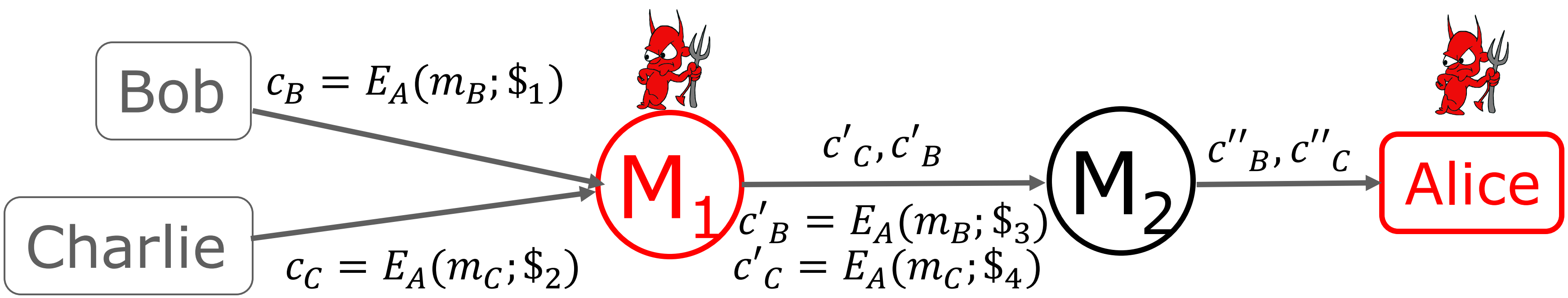
Bob

Select random b

- Problem: $g^{b \cdot d_A} \mod p$ may leak bit(s)…
- `Classical' DH solution: securely derive a key: $h\left(g^{a_i b_i} \mod p\right)$
- El-Gamal's solution: use a group where DDH believed to hold
  - Note: message must be encoded as member of the group!
  - So why use it? Some special properties…

Homomorphic Encryption

- Given: two ciphertexts $E_{e_A}(m_1), E_{e_A}(m_2)$
- Compute $E_{e_A}(m_1 \cdot m_2)$
- Applications, e.g.: re-encrypt for sender anonymity
  - Re-encrypt: $E_{e_A}(m_1;\$_1) = E_{e_A}(m_1 \cdot 1) = E_{e_A}(m_1;\$_2) \cdot E_{e_A}(1)$
    - Notation: $E_e(m;\$)$: encryption of message m with random string $\$ \in \{0,1\}^*$



- How? We show with El-Gamal

Note: does NOT work using $h\left(g^{a_i b_i} mod\ p\right)$

El-Gamal PKC: homomorphism

- Given two ciphertexts:
  - $E_{e_A}(m_1) = (x_1, y_1) = (g^{b_1} \, mod \, p, m_1 * g^{b_1 \cdot d_A} \, mod \, p)$
  - $E_{e_A}(m_2) = (x_2, y_2) = (g^{b_2} \, mod \, p, m_2 * g^{b_2 \cdot d_A} \, mod \, p)$
- $(x_1 x_2, y_1 y_2) = (g^{b_1 + b_2} \, mod \, p, m_1 \cdot m_2 * g^{(b_1 + b_2) \cdot d_A} \, mod \, p) =$
  $= E_{e_A}(m_1 \cdot m_2)$
- Decrypts to same message!
- Extension: **universal** re-encryption:
  same but without knowing public key g^a
  - Hint: send encryption of 1 with each ciphertext
  - Use: mix ciphertext for anonymous **recipient**, too

Fully-homomorphic encryption?

- We discussed multiplicative-homomorphism:
  - Given: two ciphertexts $E_{e_A}(m_1), E_{e_A}(m_2)$
  - Compute $E_{e_A}(m_1 \cdot m_2)$
- Alternative forms of homomorphism….
  - Additive-homomorphism: Compute $E_{e_A}(m_1 + m_2)$
  - Fully-homomorphic: both!
- Fully-homomorphic encryption:
  - Allows computing arbitrary function $E_{e_A}(f(m_1, m_2))$
    - Given only encrypted values: $E_{e_A}(m_1), E_{e_A}(m_2)$
    - Important… allows computing on encrypted data!!
    - Several designs, high overhead…

RSA Public Key Cryptosystem

2002 Turing Award

- First proposed – and still widely used
- Not really covered in this course – take crypto!
- Some basic details…
- Select two large primes p,q ; let $n=pq$
- Select prime e (public key: $<n,e>$)
  - Or co-prime with $\Phi(n) = (p-1)(q-1)$
- Let private key be $d=e^{-1} \bmod \Phi(n)$ (i.e., $ed=1 \bmod \Phi(n)$)
- Encryption: $RSA.E_{e,n}(m)=m^e \bmod n$
- Decryption: $RSA.D_{d,n}(c)=c^d \bmod n$
- Correctness: $D_{d,n}(E_{e,n}(m)) = (m^e)^d = m^{ed} = m \bmod n$
  - Intuitively: $ed=1 \bmod \Phi(n) \Rightarrow m^{ed} = m \bmod n$
- But why ?
  - A bit of number-theory `magic'…

Euler Theorem & Function $\phi_n = \Phi(n)$

- Euler's Theorem:
  if a, n are co-primes then $a^{\Phi(n)}=1 \bmod n$
  - Co-primes: no common divisor, i.e. $gcd(a, n)=1$
- Where $\Phi(n)$, called Euler function of n, is the number of positive integers less than n and co-prime to n.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\Phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

- $\Phi(p)=(p-1)$ for any prime p
- $\Phi(pq)=(p-1)(q-1)$ for any primes p,q
  - Why? pq has common divisor with $p, 2p,.. (q-1)p, q, 2q, ...(p-1)q$
  - So number of smaller co-primes: $(pq-1)-[(p-1)+(q-1)]=pq-p-q+1=(p-1)(q-1)$

Euler Theorem & Function $\phi_n = \Phi(n)$

- Euler's Theorem:
  if a, n are co-primes then $a^{\Phi(n)}=1 \bmod n$
  - Co-primes: no common divisor, i.e. $gcd(a, n)=1$
- Where $\Phi(n)$, called Euler function of n, is the number of positive integers less than n and co-prime to n.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\Phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

- For primes p, q holds $\Phi(pq)=(p-1)(q-1)$
- Exercises: (1) Fermats' little Theorem:
  p prime, $a \bmod p \neq 0 \Rightarrow a^{p-1} = 1 \bmod p$
- (2) $(\forall b)\ a^b = a^{b \bmod (p-1)} \bmod p$, $a^b = a^{b \bmod \Phi(n)} \bmod n$

RSA Public Key Cryptosystem

- Select two large primes p,q and let $n=pq$
  - ➔ $\Phi(n)=\Phi(pq)=(p-1)(q-1)$
- Select prime e; public key: $<n,e>$
  - Private key: $d=e^{-1} \bmod \Phi(n)$ ➔ $ed=1+l\,\Phi(n)$ for some l
- Encryption: $RSA.E_{e,n}(m)=m^e \bmod n$
- Decryption: $RSA.D_{d,n}(c)=c^d \bmod n$
- Correctness: $D_{d,n}(E_{e,n}(m))= m^{ed} \bmod n$
- $m^{ed}=m^{ed}= m^{1+l\,\Phi(n)} =m\, m^{l\,\Phi(n)} =m\,(m^{\Phi(n)})^l$
- $m^{ed} \bmod n =m\,(m^{\Phi(n)} \bmod n)^l \bmod n$
- Eulers'Theorem: $m^{\Phi(n)} \bmod n=1 \bmod n$
- ➔ $D_{d,n}(E_{e,n}(m))= m^{ed} \bmod n=m\,1^l \bmod n =m$

RSA Public Key Cryptosystem

- Correctness: $D_{d,n}(E_{e,n}(m)) = m^{ed} \bmod n$
- $m^{ed} = m^{ed} = m^{1+l\,\Phi(n)} = m\, m^{l\,\Phi(n)} = m\,(m^{\Phi(n)})^l$
- $m^{ed} \bmod n = m\,(m^{\Phi(n)} \bmod n)^l \bmod n$
- Eulers'Theorem: $m^{\Phi(n)} \bmod n = 1 \bmod n$
- $\rightarrow D_{d,n}(E_{e,n}(m)) = m^{ed} \bmod n = m\, 1^l \bmod n = m$
- Comments:
  - $m<n \rightarrow m = m \bmod n$
  - Eulers'Theorem holds (only) if m, n are co-primes
  - If not co-primes? Use Chinese Reminder Theorem
    - A nice, not very complex argument
    - But: beyond our scope – take Crypto!

The RSA Problem and Assumption

- RSA problem: Find m , given (n,e) and 'ciphertext' value $c = m^e \bmod n$
- RSA assumption: if (n,e) are chosen `correctly', then the RSA problem is `hard'
  - I.e., no efficient alg can find m with high probability
  - For `large' n and $m \overset{\$}{\leftarrow} \{1, \ldots, n\}$

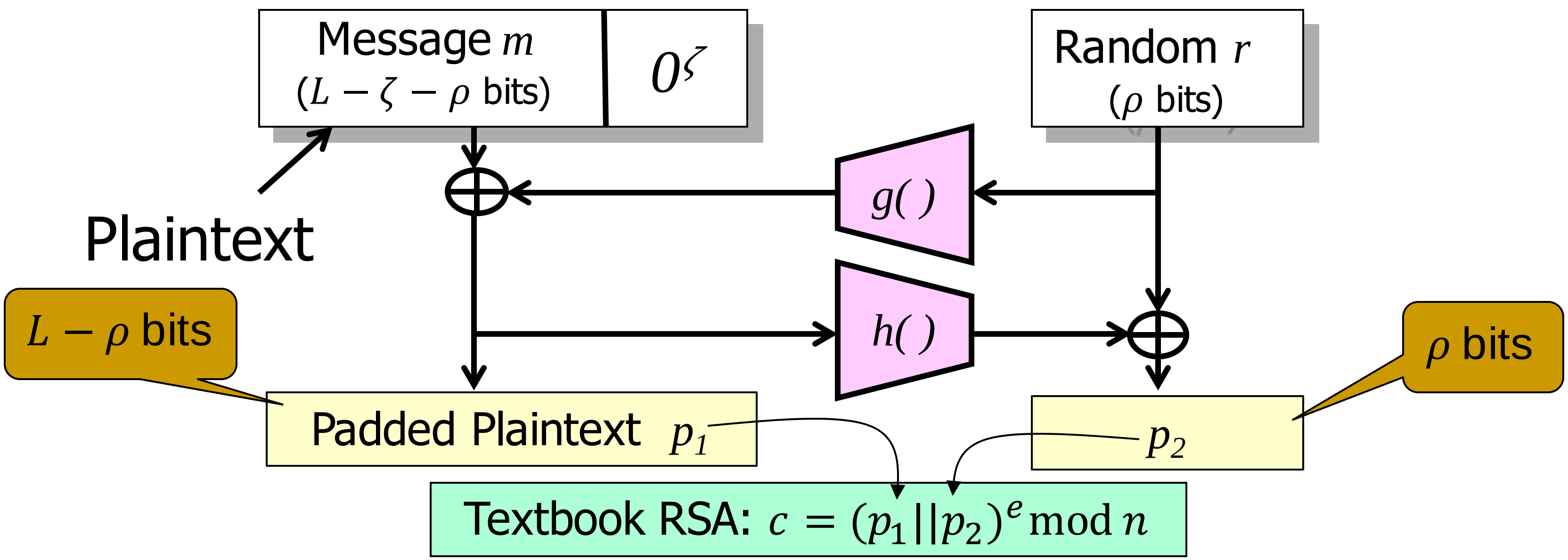
- Does not prevent exposure of partial information
- May not be secure for a non-random message
- Does not ensure randomization (indistinguishablity)

Padding RSA

- Pad and Unpad functions: $m = Unpad(Pad(m;r))$
  - Encryption with padding: $c = [Pad(m,r)]^e \bmod n,$
  - Decryption with unpad: $m = Unpad(c^d \bmod n)$
- Required to…
  - Add randomization
    - Prevent detection of repeating plaintext
  - Prevent 'related message' attack (to allow use of tiny e)
  - Detect, prevent (some) chosen-ciphertext attacks
- Early paddings schemes subject to CCA attacks
  - Even 'Feedback-only CCA' (aware of unpad failure)

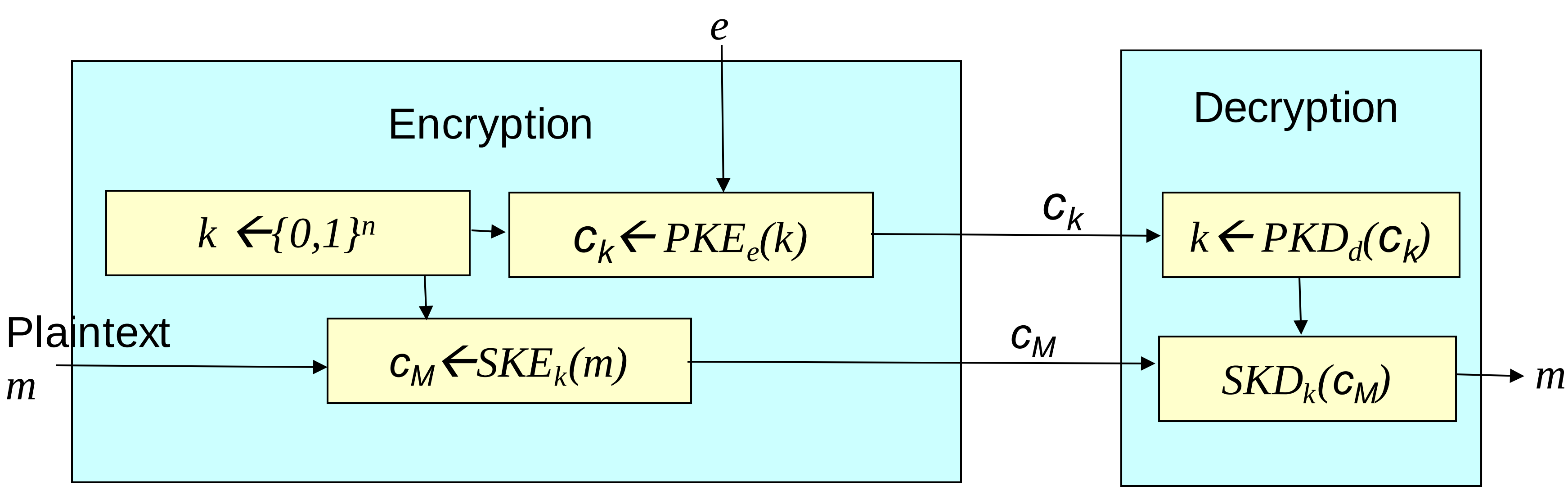
Optimal Asymmetric Encryption Padding (OAEP)

- No chosen-ciphertext attacks: ciphertext 'proves' *knowledge of plaintext*
- Feistel-like; use two crypto-hash functions g, h (assume 'random')
  - Let L be length of input to RSA, $\zeta, \rho \ll \mathrm{L}$ be 'security parameters' (say 80 bits)
  - g: 'random function' from ρ bits to L-ρ bits, h: 'random function' from L-ρ bits to ρ bits
  - If p_1 wasn't used as input to h → $h(p_1)$ is 'random' → $h(p_1)\ r$ is 'random' → $g(h(p_1) \oplus r)$ is 'random' → highly unlikely that ζ LSbits of $p_1 \oplus g(h(p_1) \oplus r)$ are zero
  - This kind of argument is called *random oracle methodology (ROM)*



Hybrid Encryption (`enveloping`)

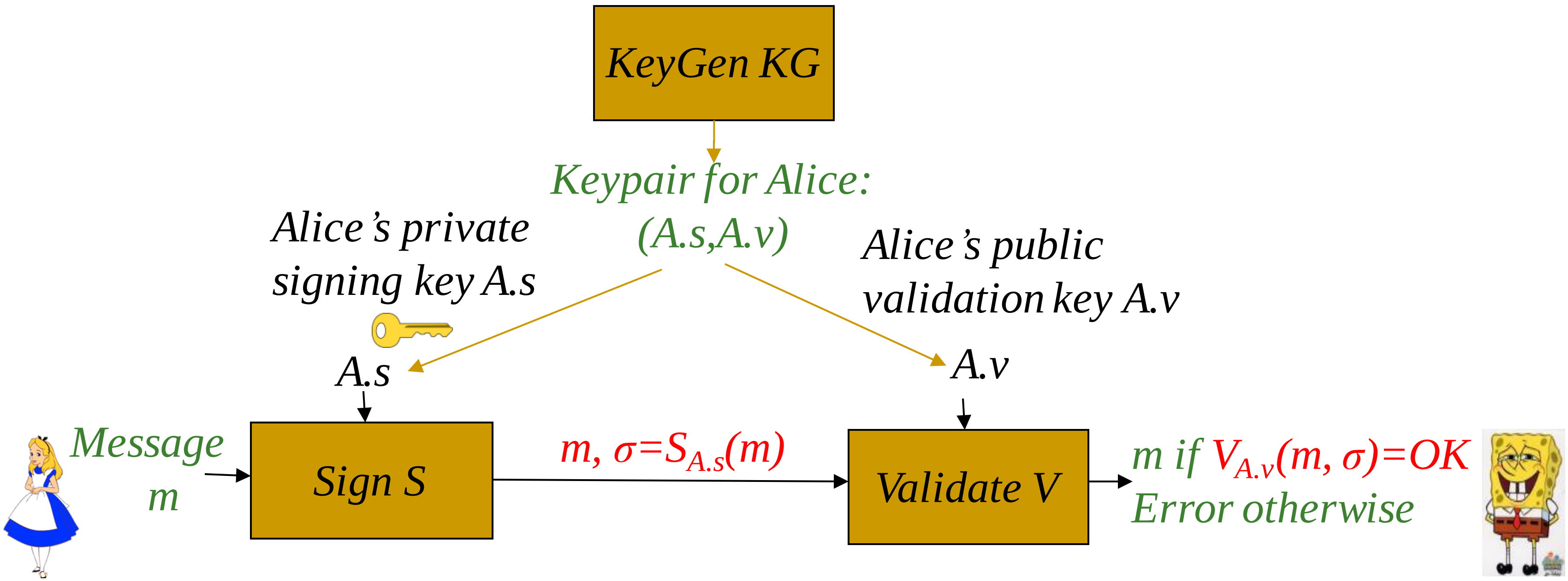
- Challenge: public key cryptosystems are slow
- Hybrid encryption:
  - Use VIL secret key cryptosystem (SKE, SKD)
  - Encrypt shared key k and use k to encrypt plaintext
  - Send ciphertext c_M (encrypted message) with encrypted key c_k



How does Bob know Alice’s public key?

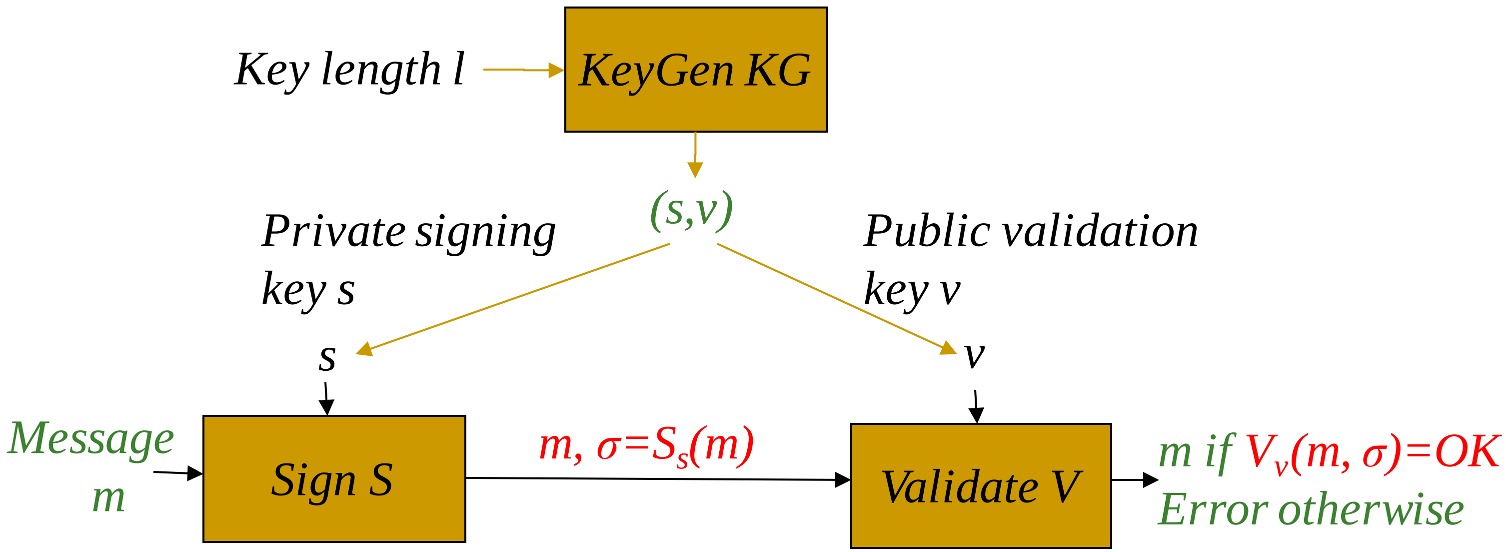
- Depends on threat model…
  - Passive (`eavesdropping`) adversary: just send it
  - Off-path (`blind`) adversary: use nonce
  - Man-in-the-Middle (MITM): **authenticate**
- Authenticate – how?
  - MAC: requires shared secret key
  - **Public key signature scheme**: authenticate using public key
  - Certificate: public key of entity – **signed** by **certificate authority (CA)**

Public Key Digital Signatures



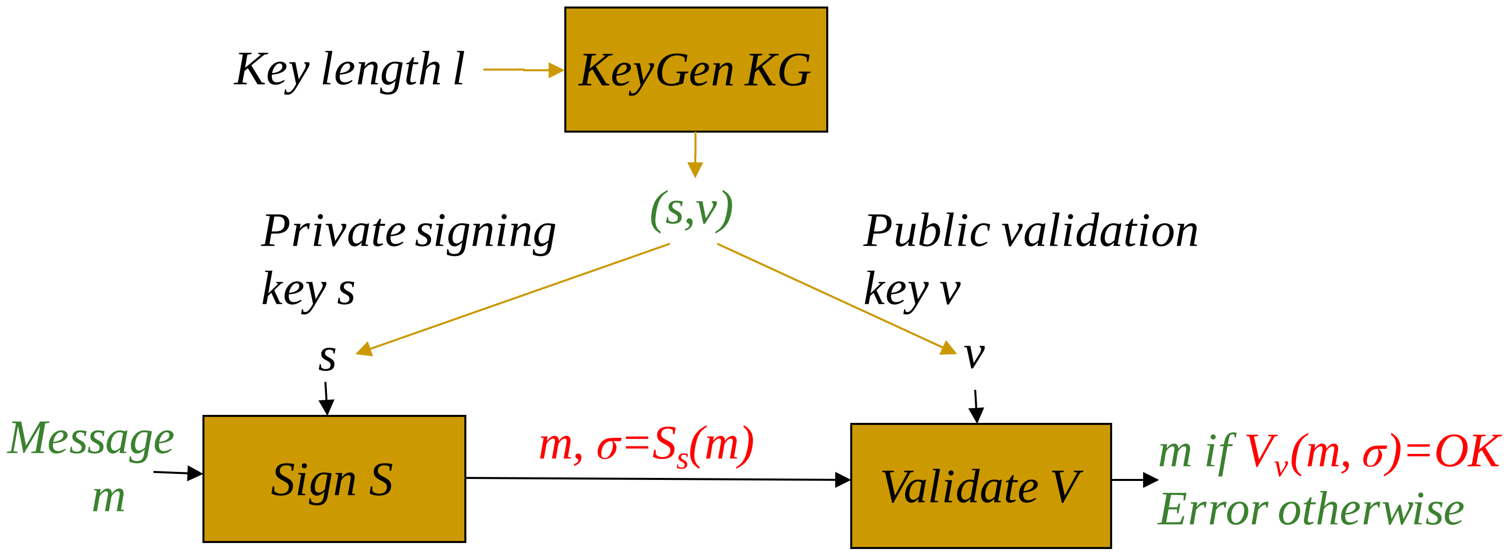
- Sign using a private, secret signature key (*A.s* for Alice)
- Validate using a <u>public</u> key (*A.v* for Alice)
- Everybody can validate signatures at any time
  - Provides authentication, integrity <u>**and**</u> evidence / non-repudiation
  - MAC: 'just' authentication+integrity, no evidence, can repudiate

PK Signatures: Unforgeability Requirement



- Unforgeability: given v, attacker should be unable to find **any** 'valid' (m, σ), i.e., $V_v(m, \sigma)=OK$
  - Even when attacker can select messages m', receive $\sigma'=S_s(m')$
  - For any message except chosen m

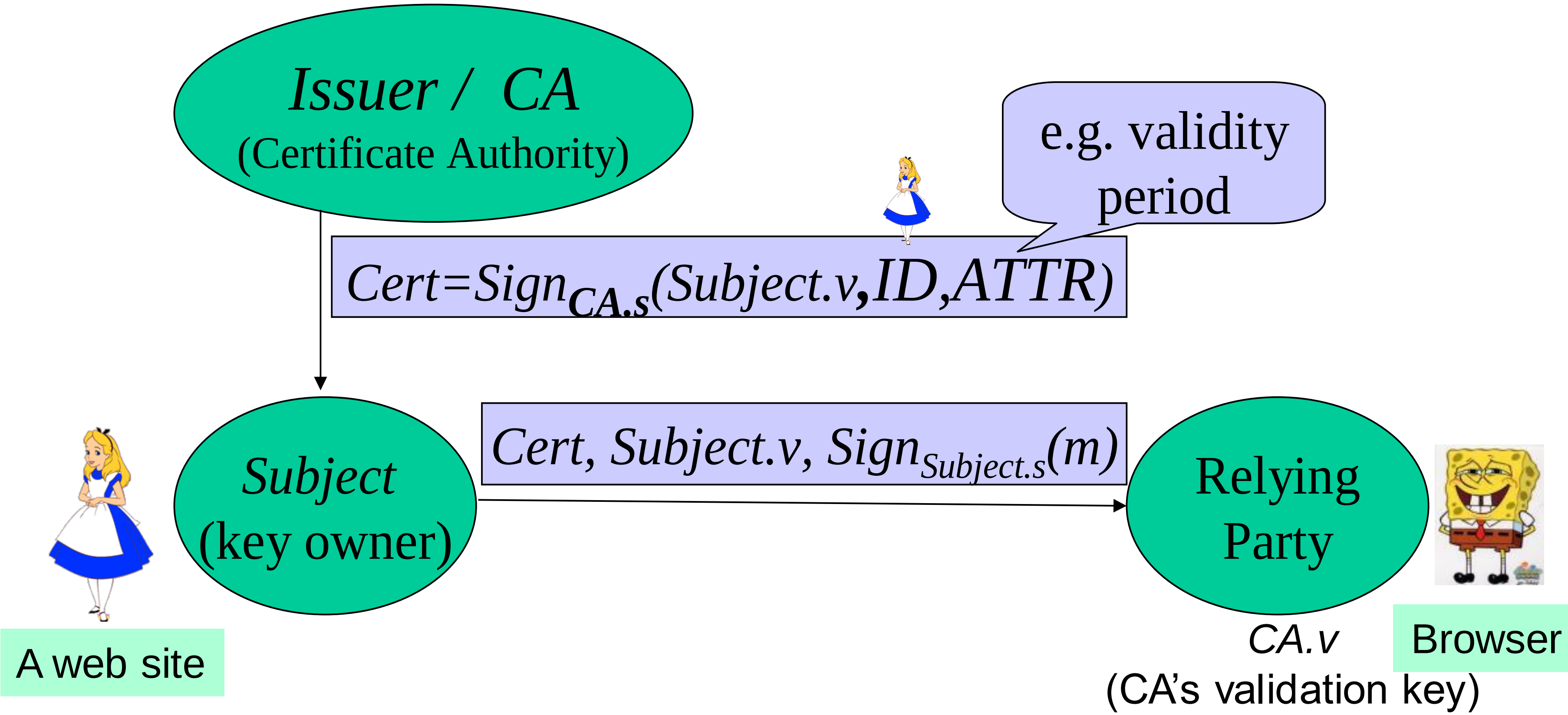
PK Signatures: Unforgeability Requirement



- (Existential) Unforgeability Experiment:
  - Generate (s, v), give v to attacker
  - Attacker can select messages m', receive $\sigma'=S_s(m')$
  - Attacker outputs claimed-forgery: (m, σ)
  - Attacker wins if v(m,)=Ok , and attacker never selected m

Public Key Certificate

- *Certificate:* signature by Certificate Authority (CA) over subject's public key and attributes (e.g., domain name)



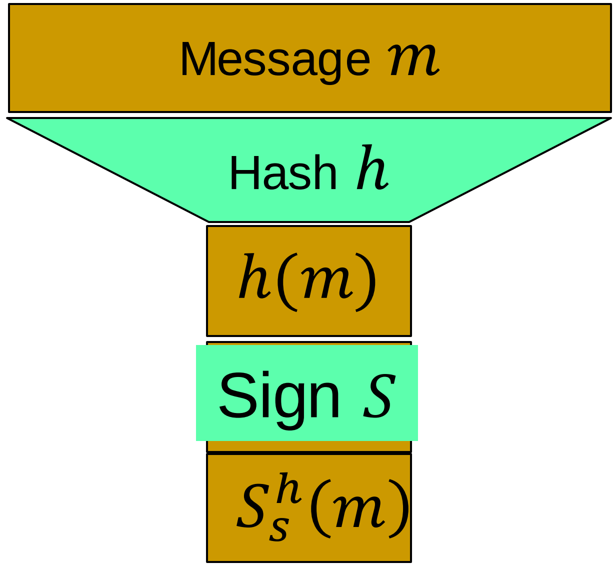
More: in TLS/PKI lectures…

RSA Signatures

- Secret signing key s, public verification key v
- Short (<n) messages: RSA signing with message recovery
- First attempt:
  - RSA.$S_s(m)= m^s \bmod n$,
    RSA.$V_v(m,x)=\{$ OK if $m=x^v \bmod n$; else, FAIL $\}$
  - Hmm… for any x, let $m=x^v \bmod n$; then RSA.$V_v(m,x)=$ OK
  - Unforgeability requirement fails: attacker has a forgery !
- Preventing `random signatures' ?
  - RSA.$S_s(m)= pad(m)^s \bmod n$,
    RSA.$V_v(m,x)=\{$OK if $m=unpad(x^v \bmod n)$; else, FAIL$\}$
  - *Pad, unpaid:* redundancy added (pad) and verified (unpad)
- Long messages: ??
  - Hint: use collision resistant hash function (CRHF)

The Hash-then-Sign Paradigm

- Challenge: messages are long, PKC is slow
- How to sign long messages – efficiently?
  - Using Collision-Resistant Hash h :
    ➔ infeasible to find pair (x, x') s.t. $x' \neq x$ yet $h(x)=h(x')$
  - And signature scheme (S, V)
- Solution: $S_s^h(m) = S_s\big(h(m)\big)$
  - Cf.: hybrid encryption



RSA Signatures

- Secret signing key s, public verification key v
- Short (<n) messages: RSA signing with message recovery
  - RSA.$S_s(m) = R(m)^s \bmod n$,
    RSA.$V_v(x) = \{R^{-1}(x^v \bmod n)$; error if undefined$\}$
  - $R(.)$: redundancy function ; make random string unlikely to be valid signature
- Long messages: hash-then-sign, RSA_h, $h:\{0,1\}^* \rightarrow \{0.1\}^L$
  - Aka signature with appendix
  - RSA_$h.S_s(m) = m||[h(m)]^s \bmod n$
  - RSA_$h.V_v(m \,||\, x) = m$ iff $h(m) = x^v \bmod n$ (else: error)
  - m, m' s.t. $h(m)=h(m') \rightarrow sign(h(m))=sign(h(m'))$
- h is (keyless) collision resistant hash function (CRHF)
  → infeasible to find <u>pair</u> (x, x') s.t. $x' \neq x$ yet $h(x)=h(x')$

Discrete-Log Digital Signature?

- RSA allowed encryption and signing… based on assuming factoring is hard
- Can we sign based on assuming discrete log is hard?
- Most well-known, popular scheme: DSA
  - Digital Signature Algorithm, by NSA/NIST
  - Details: crypto course
- We'll discuss simpler, less efficient <u>El-Gamal Signatures</u>

El-Gamal signatures

- Parameters: $p \leftarrow primes[n\ bit], g \leftarrow Generator(p)$
- Key generation: $s \overset{\$}{\leftarrow} \{2,..,p-2\}, v \leftarrow g^s mod\ p$
- Sign: $k \overset{\$}{\leftarrow} \{2,..,p-2 | \gcd(k,p-1)=1\}$
  - $r \leftarrow g^k mod\ p, \quad t \leftarrow (h(m) - sr) \cdot k^{-1} mod\ (p-1)$
  - If $t = 0$ then select new k
  - Signature is (r,t)
- Verify: $g^{h(m)} = v^r r^t\ mod\ p; 0 < r < p; 0 < t < p-1$
- Correctness:

$$g^{h(m)} = g^{sr+kt} = (g^s)^r \left(g^k\right)^t = v^r r^t\ mod\ p$$

- Using Fermat law: $g^b = g^{b\ mod\ (p\text{-}1)}\ mod\ p$
- Efficient off-line sign: precompute $r \leftarrow g^k mod\ p$

Summary

- Public key crypto allows:
  - Easier key management, distribution
    - Key agreement (DH): only need authenticated channel
    - Encryption: easier distribution, maintenance – **public** key
  - Resiliency to key exposure (PFS and PRS)
  - Signatures
    - Certificate: public key and a signature authenticating it
    - Evidences
    - Handling VIL messages: hash-then-sign
- Next: Public Key Infrastructure (PKI) and TLS