CSE 3400 - Introduction to Computer & Network Security
(aka: Introduction to Cybersecurity)

Lecture 11
Public Key Cryptography– Part II

Ghada Almashaqbeh
UConn

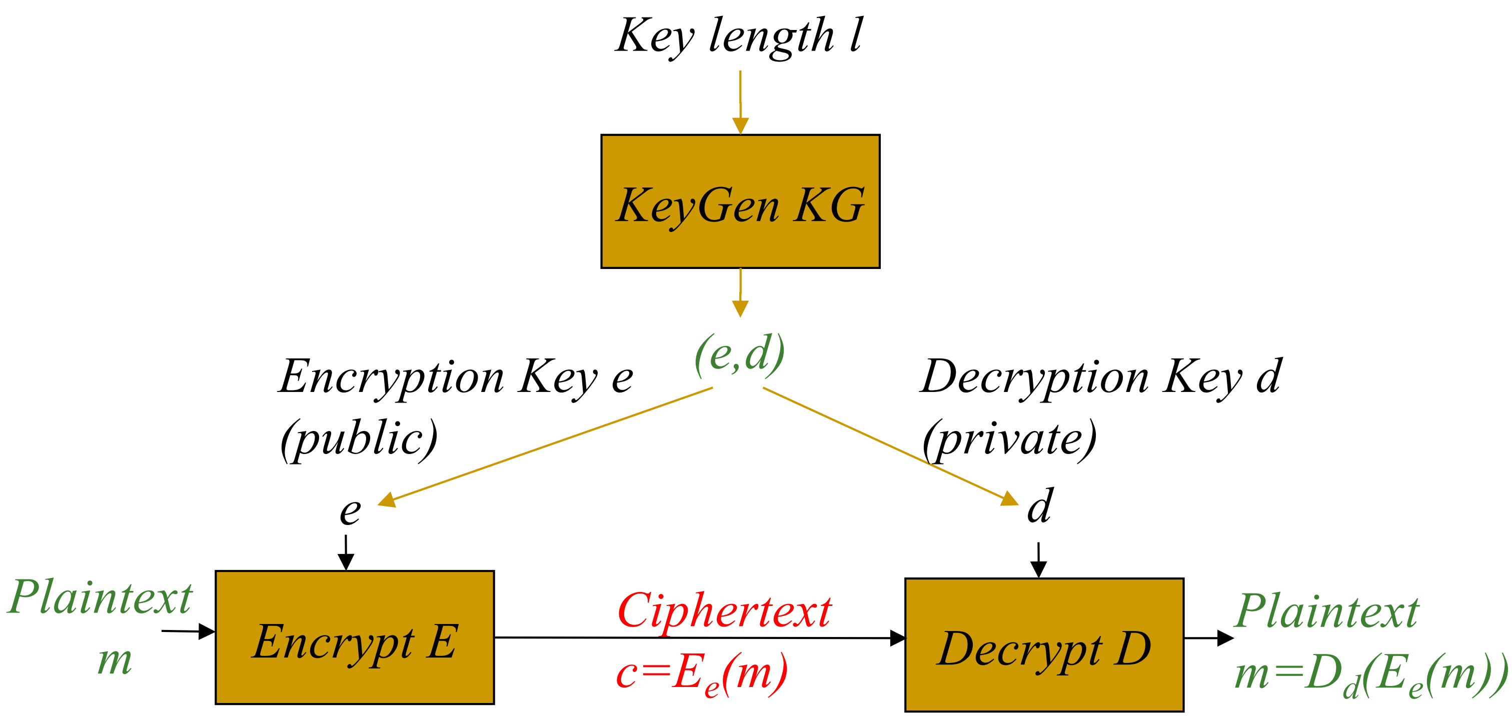
From Textbook Slides by Prof. Amir Herzberg
UConn

Outline

- Public key encryption.
- Digital signatures.

Public Key Encryption

Public Key Encryption



Public Key Encryption IND-CPA Security

$T^{IND-CPA}_{\mathcal{A},\langle KG,E,D\rangle}(b,n)$ {
    $(e,d) \overset{\$}{\leftarrow} KG(1^n)$
    $(m_0,m_1) \leftarrow \mathcal{A}(\text{`Choose'}, e)$ s.t. $|m_0| = |m_1|$
    $c^* \leftarrow E_e(m_b)$
    $b^* = \mathcal{A}(\text{`Guess'}, (c^*, e))$
    Return b^*
}

Definition 2.10 (PKC IND-CPA)**.** *Let $\langle KG, E, D\rangle$ be a public-key cryptosystem. We say that $\langle KG, E, D\rangle$ is* IND-CPA*, if every efficient adversary $\mathcal{A} \in PPT$ has negligible advantage $\varepsilon^{IND-CPA}_{<KG,E,D>,\mathcal{A}}(n) \in NEGL(n)$, where:*

$$\varepsilon^{IND-CPA}_{\langle KG,E,D\rangle,\mathcal{A}}(n) \equiv \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle KG,E,D\rangle}(1,n) = 1\right] - \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle KG,E,D\rangle}(0,n) = 1\right] \tag{2.35}$$

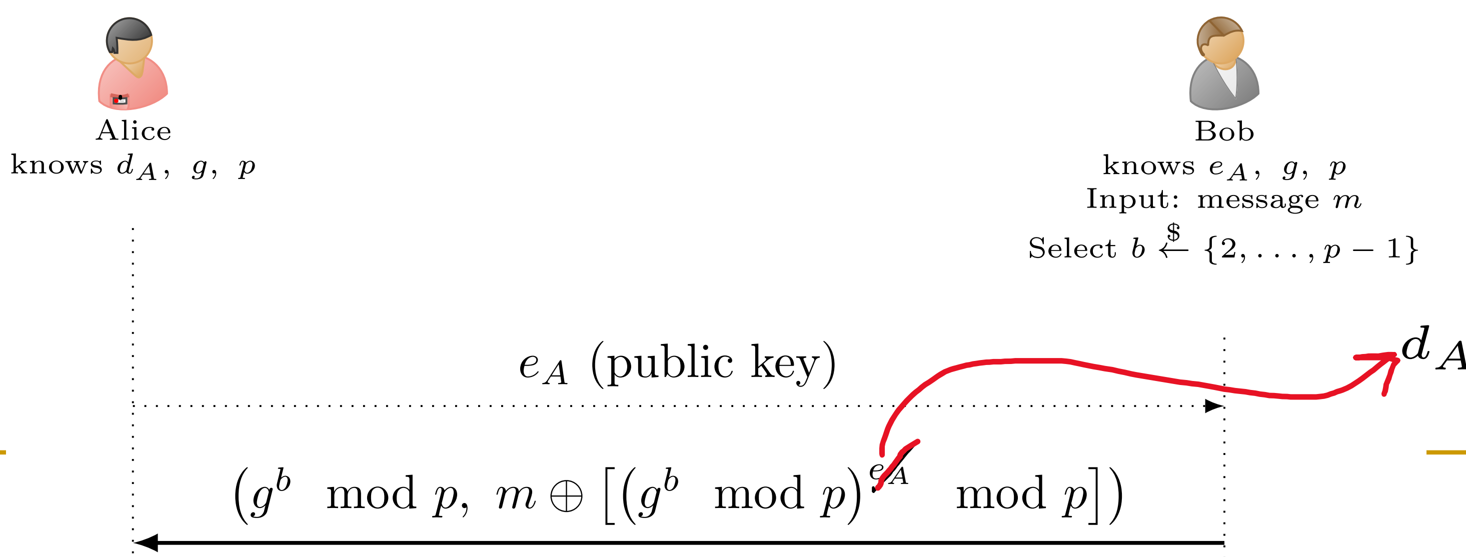
Where the probability is over the random coin tosses in IND-CPA (including of $\mathcal{A}$ and E).

Discrete Log-based Encryption

- We will explore two flavors:
  - An adaptation of DH key exchange protocol to perform encryption.
  - ElGamal encryption scheme.

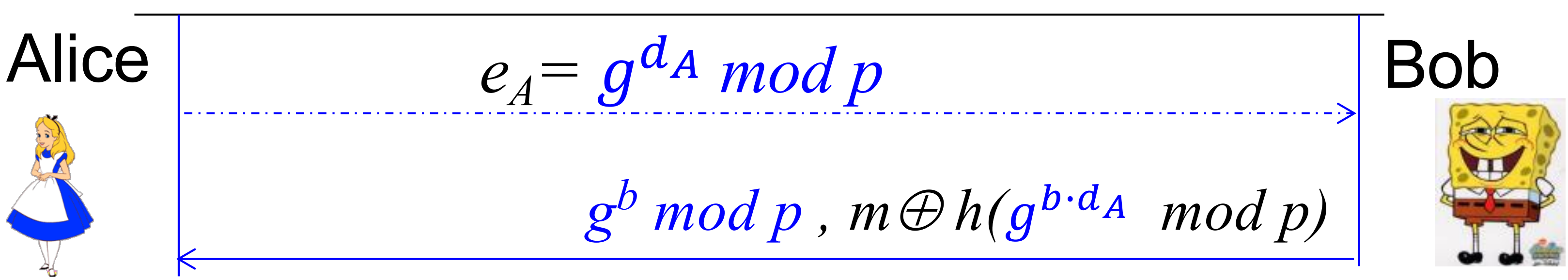
Turning [DH] to Public Key Cryptosystem

- 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: $g^b \bmod p$, $m \oplus (e_A)^b = m \oplus g^{b \cdot d_A} \bmod p$
  - Secure assuming DDH: if the attacker can distinguish $g^{b \cdot d_A}$ from a random string, IND-CPA may not hold.



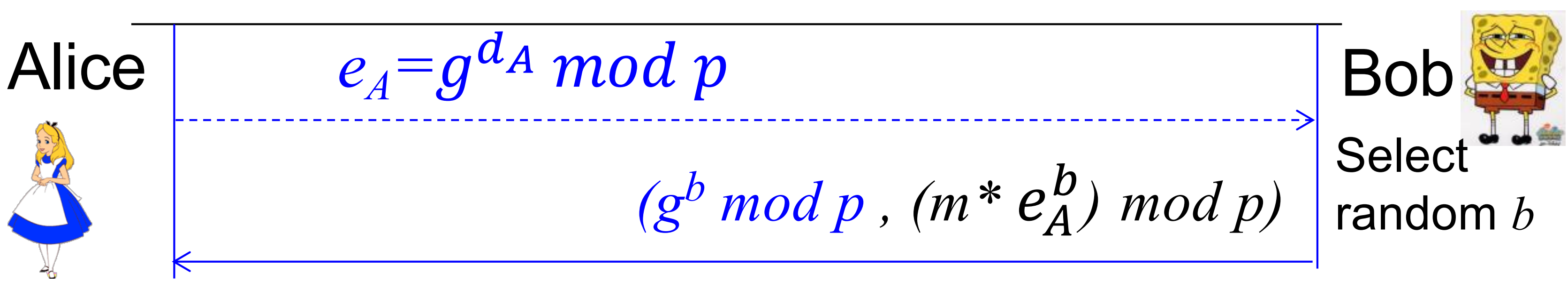
Turning [DH] to Public Key Cryptosystem

- Solves dependency on DDH assumption; secure under the (weaker) CDH assumption.
- To encrypt message m to Alice:
  - Bob selects random b
  - Sends: $g^b \; mod \; p \; , \; m \oplus h((e_A)^b) = m \oplus h(g^{b \cdot d_A} \; mod \; p)$
  - Secure if $h(g^{b \cdot d_A} \; mod \; p)$ is pseudo-random



ElGamal Public Key Encyption

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



ElGamal Public Key Encryption

- Encryption:

$$E^{EG}_{e_A}(m) \leftarrow \left\{ \left(g^b \mod p \;,\; m \cdot e_A^b \mod p\right) | b \xleftarrow{\$} [2, p-1] \right\}$$

- Decryption:

$$D_{d_A}(x, y) = x^{-d_A} \cdot y \mod p$$

- Correctness:

$$\begin{aligned} D_{d_A}(g^b \mod p \;,\; m \cdot \; e_A^b \mod p) \;=& \\ =& \left[\left(g^b \mod p\right)^{-d_A} \cdot \left(m \cdot \left(g^{d_A}\right)^b \mod p\right)\right] \mod p \\ =& \left[g^{-b \cdot d_A} \cdot m \cdot g^{b \cdot d_A}\right] \mod p \\ =& \; m \end{aligned}$$

ElGamal Public Key Cryptosystem

- 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…

ElGamal PKC: homomorphism

- Multiplying two ciphertexts produces a ciphertext of the multiplication of the two plaintexts.
- Given two ciphertexts:
  - $E_{e_A}(m_1) = (x_1, y_1) = (g^{b_1} \bmod p, m_1 * g^{b_1 \cdot d_A} \bmod p)$
  - $E_{e_A}(m_2) = (x_2, y_2) = (g^{b_2} \bmod p, m_2 * g^{b_2 \cdot d_A} \bmod p)$
- $Mult\big((x_1, y_1), (x_2, y_2)\big) \equiv (x_1 x_2, y_1 y_2)$
- Homomorphism:
- $= (g^{b_1+b_2} \bmod p, m_1 \cdot m_2 * g^{(b_1+b_2)\cdot d_A} \bmod p) =$
  $= E_{e_A}(m_1 \cdot m_2)$
- ➔ compute $E_{e_A}(m_1 \cdot m_2)$ from $E_{e_A}(m_1)$, $E_{e_A}(m_1)$

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; huge research effort to reduce this overhead.

RSA Public Key Encryption

2002 Turing Award

- First proposed – and still widely used
- Not really covered in this course – take crypto!
- 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? Remember Euler's theorem.

RSA Public Key Cryptosystem

- Correctness: $D_{d,n}(E_{e,n}(m)) = m^{ed}\ mod\ n$
- $m^{ed} = m^{ed} = m^{1+l\,\Phi(n)} = m\, m^{l\,\Phi(n)} = m\,(m^{\Phi(n)})^{l}$
- $m^{ed}\ mod\ n = m\,(m^{\Phi(n)}\ mod\ n)^{l}\ mod\ n$
- Eulers'Theorem: $m^{\Phi(n)}\ mod\ n = 1\ mod\ n$
- $\rightarrow D_{d,n}(E_{e,n}(m)) = m^{ed}\ mod\ n = m\ 1^{l}\ mod\ n = m$
- Comments:
  - $m<n \rightarrow m = m\ mod\ n$
  - 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 algorithm can find m with non-negligible probability
  - For `large' n and $m \overset{\$}{\leftarrow} \{1, \ldots, n\}$
- RSA and factoring
  - Factoring alg ➔ alg to 'break' RSA
  - Algorithm to find RSA private key ➔ factoring alg
  - But: RSA-breaking may *not* allow factoring

RSA PKC Security

- It is a deterministic encryption scheme → cannot IND-CPA secure.
- RSA assumption does not rule out exposure of partial information about the plaintext.
- It is not CCA secure.

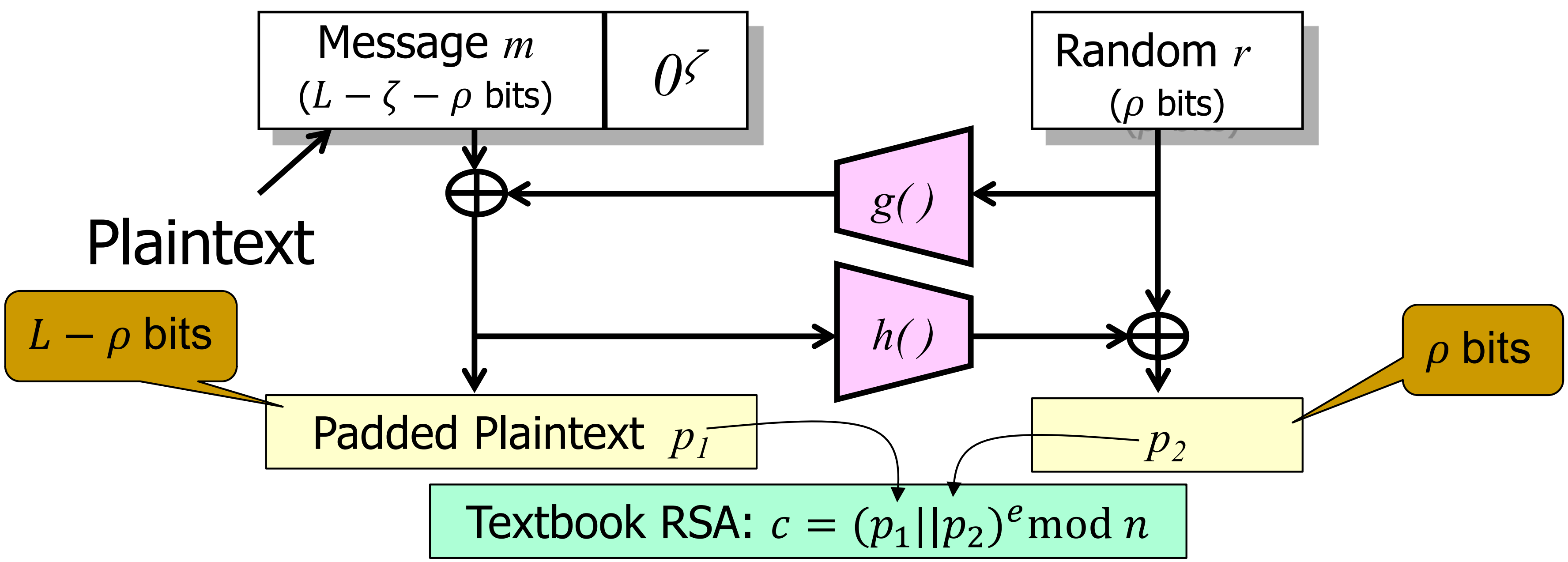
A solution: apply a random padding to the plaintext then encryption using RSA.

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, $\zeta, \rho \ll L$ 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
  - Secure in the *random oracle model (ROM)*

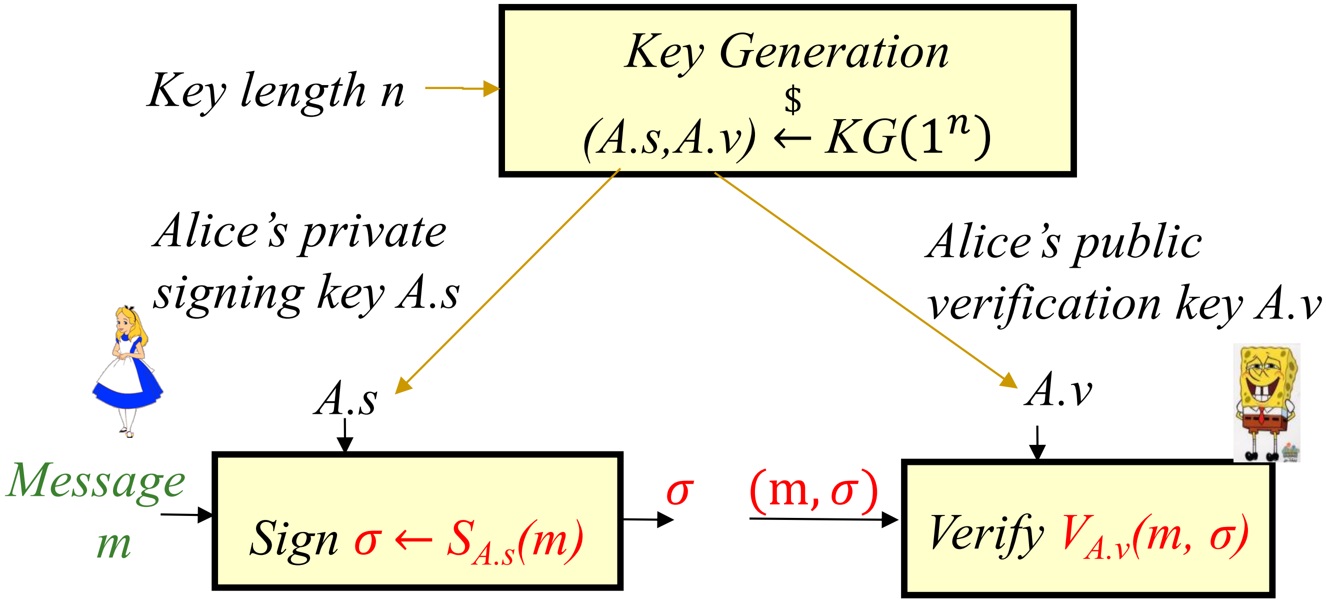


How does Bob know Alice's public key?

- Depends on threat model…
  - Passive (`eavesdropping`) adversary: just send it
  - 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)**

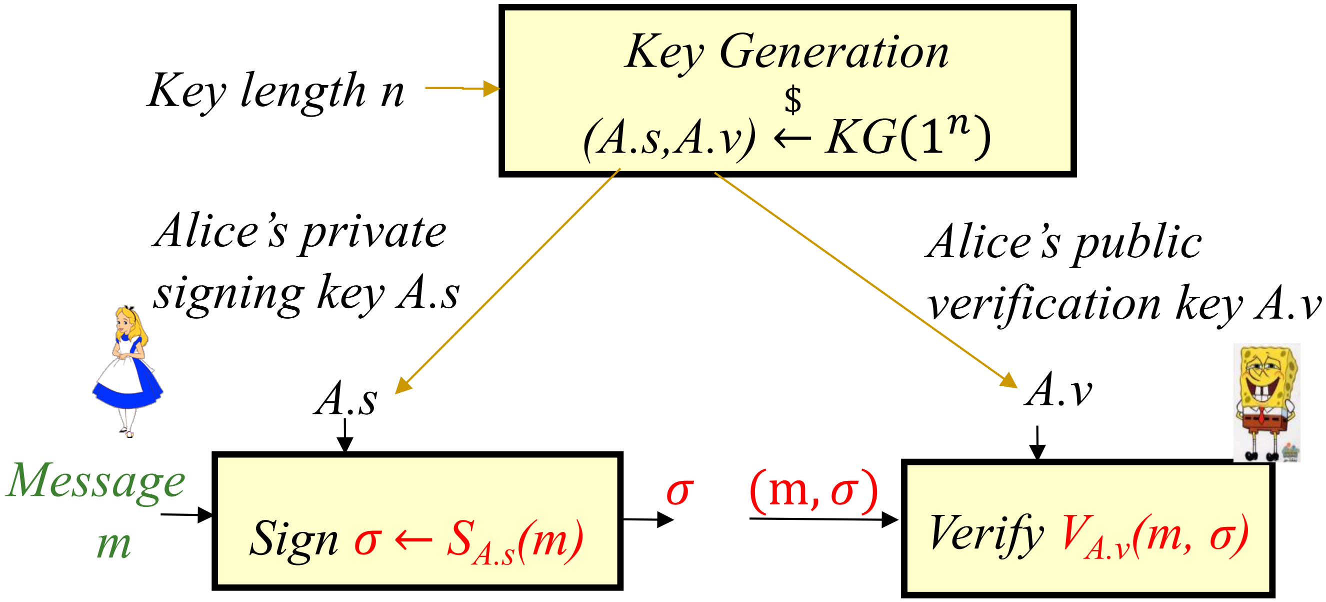
Digital Signature

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

Digital Signatures Security: Unforgeability



- 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

Digital Signature Scheme Definition

Definition 1.4 (Signature scheme and its correctness)**.** *A* signature scheme *is defined by a tuple of three efficient (PPT) algorithms,* $\mathcal{S} = (\mathcal{KG}, \mathcal{Sign}, \mathcal{Verify})$, *and a set* M *of messages, such that:*

$\mathcal{KG}$ *is a randomized algorithm that maps a unary string (security parameter* 1^l*) to a pair of binary strings* $(\mathcal{KG}.s(1^l), \mathcal{KG}.v(1^l))$.

$\mathcal{Sign}$ *is an algorithm*[8] *that receives two binary strings as input, a signing key* $s \in \{0,1\}^*$ *and a message* $m \in M$*, and outputs another binary string* $\sigma \in \{0,1\}^*$. *We call* σ *the* signature *of* m *using signing key* s.

$\mathcal{Verify}$ *is a predicate that receives three binary strings as input: a verification key* v*, a message* m*, and* σ*, a purported signature over* m. $\mathcal{Verify}$ *should output* TRUE *if* σ *is the signature of* m *using* s*, where* s *is the signature key corresponding to* v *(generated with* v*).*

Usually, M *is a set of binary strings of some length. If* M *is not defined, then this means that any binary string may be input, i.e., the same as* $M = \{0,1\}^*$.

We say that a signature scheme $(\mathcal{KG}, \mathcal{Sign}, \mathcal{Verify})$ *is* correct*, if for every security parameter* 1^l *holds:*

$$\left(\forall (s,v) \xleftarrow{\$} \mathcal{KG}(1^l),\ m \in M\right) \mathcal{Verify}_v(m, \mathcal{Sign}_s(m)) = \text{'}Ok\text{'} \qquad (1.31)$$

Digital Signature Scheme Security

Algorithm 1 The existential unforgeability game $EUF^{Sign}_{\mathcal{A},\mathcal{S}}(1^l)(1^l)$ between signature scheme $\mathcal{S} = (\mathcal{KG}, \mathcal{S}ign, \mathcal{V}erify)$ and adversary $\mathcal{A}$.

$(s, v) \xleftarrow{\$} \mathcal{S}.\mathcal{KG}(1^l)$;
$(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{S}.\mathcal{S}ign_s(\cdot)}(v, 1^l)$;
return $(\mathcal{S}.\mathcal{V}erify_v(m, \sigma) \wedge (\mathcal{A}$ didn't request $S_s(m)))$;

Definition 1.6. *The* existential unforgeability advantage function *of adversary $\mathcal{A}$ against signature scheme $\mathcal{S}$ is defined as:*

$$\varepsilon^{EUF-Sign}_{\mathcal{S},\mathcal{A}}(1^l) \equiv \Pr\left(EUF^{Sign}_{\mathcal{A},\mathcal{S}}(1^l)(1^l) = \textsc{True}\right) \tag{1.32}$$

Where the probability is taken over the random coin tosses of $\mathcal{A}$ and of $\mathcal{S}$ during the run of $EUF^{Sign}_{\mathcal{A},\mathcal{S}}(1^l)$ with input (security parameter) 1^l, and $EUF^{Sign}_{\mathcal{A},\mathcal{S}}(1^l)$ is the game defined in Algorithm 1.

RSA Signatures

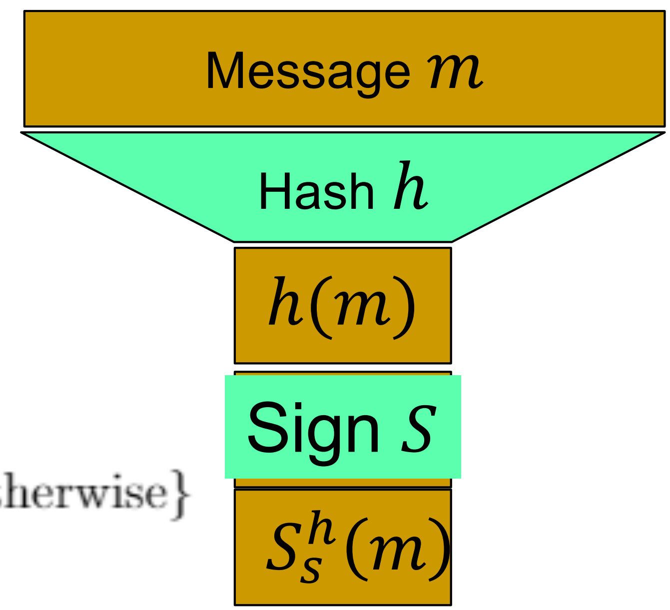
- Secret signing key s, public verification key v
- Short (<n) messages: RSA signing with message recovery
- First attempt:
  - RSA.$S_s(m) = m^s \bmod n$,
    RSA.$V_v(m,x) = \{ OK \text{ if } m = x^v \bmod n; \text{ else, } FAIL \}$
  - Hmm… for any x, let $m = x^v \bmod n$; then RSA.$V_v(m,x) = OK$
  - Unforgeability requirement fails: attacker has a forgery !
- Preventing `random signatures' ?
  - RSA.$S_s(m) = pad(m)^s \bmod n$,
    RSA.$V_v(m,x) = \{OK \text{ if } m = unpad(x^v \bmod n); \text{ else, } FAIL\}$
  - *Pad, unpad:* 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 <u>pair</u> (x, x') s.t. $x' \neq x$ yet $h(x)=h(x')$
  - And signature scheme (S, V)
- Solution: $S_s^h(m) = S_s\big(h(m)\big)$

$$S_s^{RSA,h}(m) = ([h(m)]^s \mod n,\ m)$$
$$V_v^{RSA,h}(\sigma, m) = \{m \text{ if } h(m) = \sigma^v \mod n, \text{ error otherwise}\}$$



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>ElGamal Signature</u>s

ElGamal signatures

- Parameters: $p \leftarrow prime[n\ bit], g \leftarrow Generator(p)$
- Key generation: $s \xleftarrow{\$} \{2,..,p-2\}, v \leftarrow g^s mod\ p$
- Sign: $k \xleftarrow{\$} \{2,..,p-2 | \gcd(k,p-1) = 1\}$
  - $r \leftarrow g^k mod\ p, \quad t \leftarrow (h(m) - sr) \cdot k^{-1} mod\ (p-1)$
  - 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's Theorem: $g^b = g^{b\ mod\ (p-1)}\ mod\ p$
- Efficient off-line sign: precompute $r \leftarrow g^k mod\ p$

Covered Material From the Textbook

- Chapter 1, Section: 1.3
- Chapter 2, Sections 2.7.2
- Chapter 6, Sections 6.5, 6.6, and 6.7

Thank You!