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

Public Key Cryptography– Part II

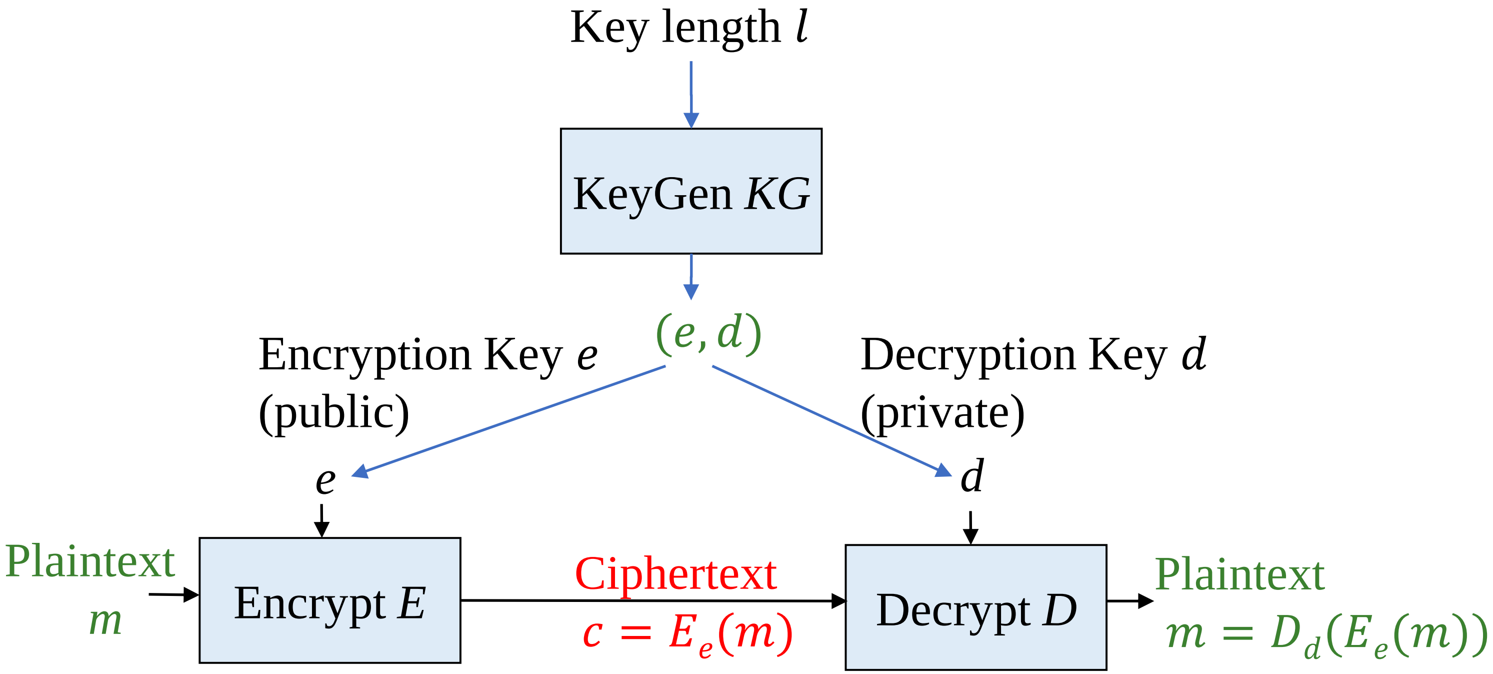
Z. Jerry Shi

From Textbook Slides by Prof. Amir Herzberg, revised by Prof. G Almashaqbeh

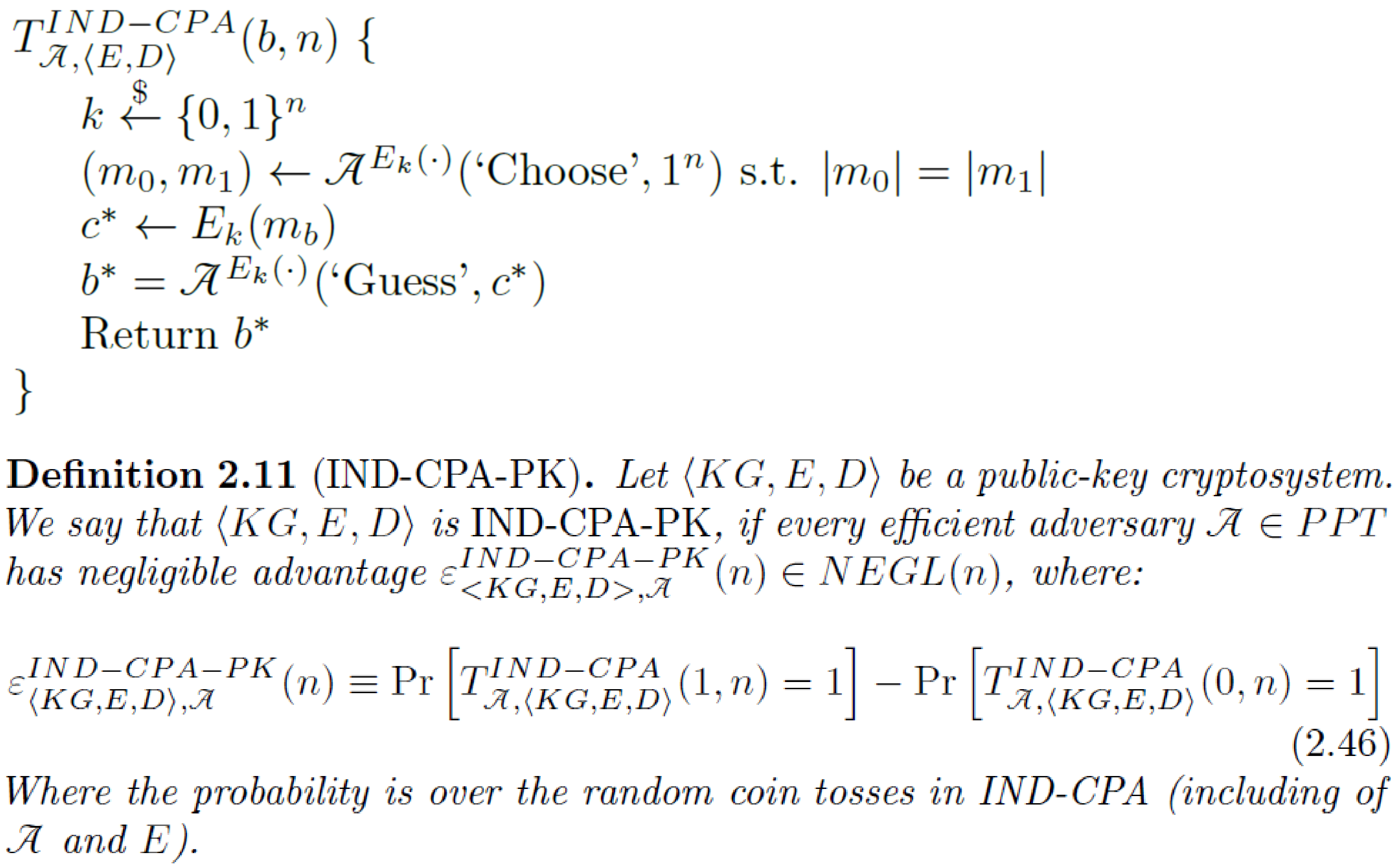
Outline

- Public key encryption
- Digital signatures
- PKI

Public Key Encryption



Public Key Encryption IND-CPA Security



```
$T^{IND-CPA}_{\mathcal{A},\langle E,D\rangle}(b,n)$ {
    $k \overset{\$}{\leftarrow} \{0,1\}^n$
    $(m_0,m_1) \leftarrow \mathcal{A}^{E_k(\cdot)}(\text{‘Choose’}, 1^n)$ s.t. $|m_0| = |m_1|$
    $c^* \leftarrow E_k(m_b)$
    $b^* = \mathcal{A}^{E_k(\cdot)}(\text{‘Guess’}, c^*)$
    Return $b^*$
}
```

Definition 2.11 (IND-CPA-PK). *Let $\langle KG, E, D\rangle$ be a public-key cryptosystem. We say that $\langle KG, E, D\rangle$ is* IND-CPA-PK, *if every efficient adversary $\mathcal{A} \in PPT$ has negligible advantage $\varepsilon^{IND-CPA-PK}_{<KG,E,D>,\mathcal{A}}(n) \in NEGL(n)$, where:*

$$\varepsilon^{IND-CPA-PK}_{\langle KG,E,D\rangle,\mathcal{A}}(n) \equiv \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle KG,E,D\rangle}(1,n) = 1\right] - \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle KG,E,D\rangle}(0,n) = 1\right] \tag{2.46}$$

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

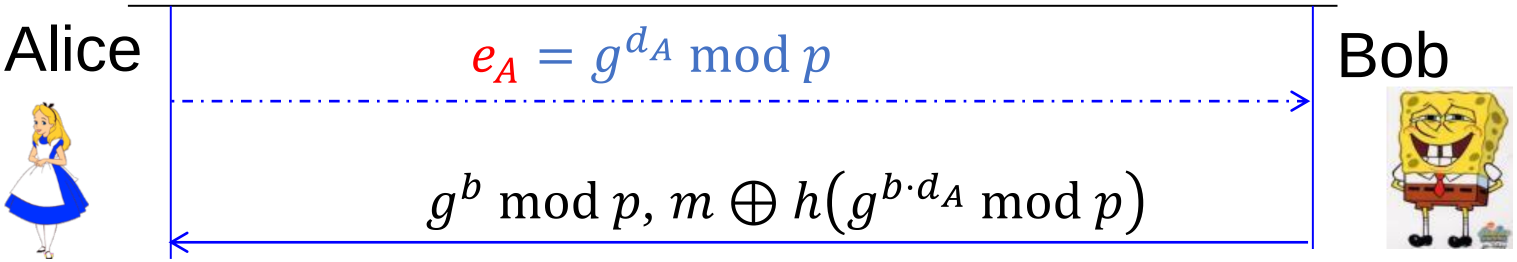
A cannot tell whether c is encrypted m0 or m1

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

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



Alice

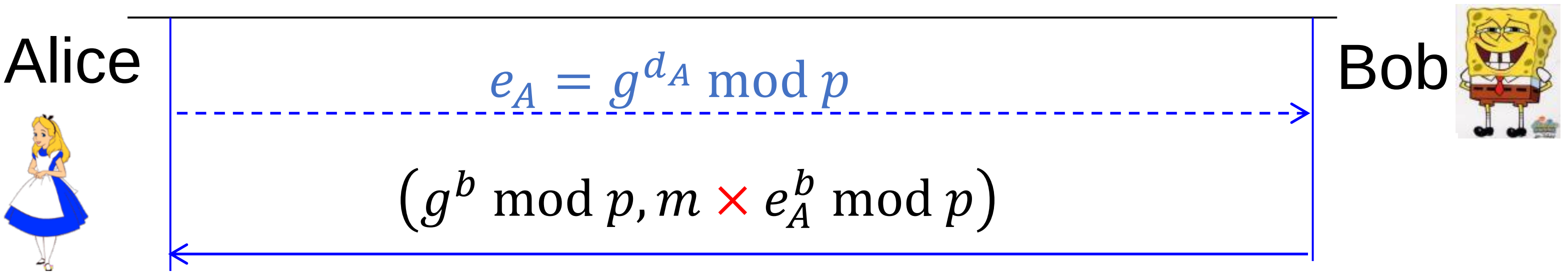
$e_A = g^{d_A} \bmod p$

Bob

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

ElGamal Public Key Encryption

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



Alice

Bob

$e_A = g^{d_A} \bmod p$

$\left(g^b \bmod p, m \times e_A^b \bmod p\right)$

ElGamal Public Key Encryption

- Encryption:

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

- 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: $g^{b \cdot d_A} \bmod p$ may leak bit(s)…
- `Classical' DH solution: securely derive a key, e.g., $h(\cdot)$

- 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

- Homomorphism: multiplying two ciphertexts produces a ciphertext of the multiplication of the two plaintexts.
- Given two ciphertexts:

$$E_{e_A}(m_1) = (\, x_1, y_1) = (g^{b_1} \bmod p,\ m_1 \ \cdot\ g^{b_1 \cdot d_A} \bmod p)$$

$$E_{e_A}(m_2) = (\, x_2, y_2) = (g^{b_2} \bmod p,\ m_2 \cdot g^{b_2 \cdot d_A} \bmod p)$$

We can compute $E_{e_A}(m_1 \cdot m_2)$ from $E_{e_A}(m_1)$, $E_{e_A}(m_1)$

$$E_{e_A}(m_1 \cdot m_2)$$
$$= (\, x_1 \cdot x_2 \bmod p, y_1 \cdot y_2 \bmod p)$$
$$= \ (g^{b_1+b_2} \bmod p, m_1 \cdot m_2 \cdot\ g^{(b_1+b_2)\cdot d_A} \bmod p)$$

RSA Public Key Encryption

2002 Turing Award

- First proposed – and still widely used
  - Not really studied in this course

- Select two large primes p, q and compute $n = pq$
- Select e s.t. e is co-prime with $\Phi(n) \; = (p-1)(q-1)$
  - The public key is (n, e)
- Let private key be $d = e^{-1} \bmod \Phi(n)$
  - $ed = 1 \bmod \Phi(n)$

For message $m < n$

Encryption: $E_{e,n}(m) = m^e \bmod n$

Decryption: $D_{d,n}(c) = c^d \bmod n$

RSA Public Key Cryptosystem

- Correctness

$$D_{d,n}(c) = c^d = \left(E_{e,n}(m)\right)^{\mathrm{d}} = (m^e)^d = m^{ed} = m \bmod n$$

If m and n are coprime

$$m^{ed} = m^{1+k\cdot\phi(n)} = m^1\, m^{k\cdot\phi(n)} = m\left(m^{\phi(n)}\right)^k = m \bmod n$$

Because $m^{\phi(n)} = 1 \bmod n$ (Euler’s Theorem)

If m and n are not coprime, use Chinese Reminder Theorem

The RSA Problem and Assumption

- RSA problem: Find m , given (n, e) and $c = m^e \bmod n$
- 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 $m \overset{\$}{\leftarrow} \{1, \ldots, n-1\}$

- RSA and factoring
  - Factoring algorithm → algorithm to 'break' RSA
  - Algorithm to find RSA private key → factoring algorithm
    - Knowing d → Factoring n
  - But: RSA-breaking may not allow factoring

RSA PKC Security

- It is a deterministic encryption scheme → cannot IND-CPA secure

$$E_{e,n}(m) = m^e \bmod n$$

- Textbook RSA is also multiplicative-homomorphic. It is not IND-CCA secure

$$m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e$$

- RSA assumption does not rule out exposure of partial information about the plaintext

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

Padding RSA

- Pad and Unpad functions:
  - Encryption with padding, and decryption with unpad:

$$c = [\mathrm{Pad}(m, r)]^e \bmod n$$
$$m = \mathrm{Unpad}(c^d \bmod n)$$

- 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

PKCS#1 padding

- PKCS#1 v1.5 padding (RFC 2313)

$$M = Pad(m) = 0\mathrm{x}00 \parallel 0\mathrm{x}02 \parallel r \parallel 0\mathrm{x}00 \parallel m$$

m: the original message

r : a random string. At least 8 non-zero random bytes

Not semantically secure

- Optimal Asymmetric Encryption Padding (OAEP)
  - Adopted in PKCS#1 v2.0

Small encryption key

- Since e is public, we can choose small e and make encryption fast, for example, $e = 3$
  - There are attacks proposed, but can be prevented padding
  - In practice, it is advised to use a larger one, e.g., $e = 65537$

- Decryption key d must be large
  - The length of d should be longer than $0.292|n|$

Common modulus

Suppose you are an admin of company C.

You generate keys for each employee in the company as the follows.

1. Select two large primes p, q and compute $n = pq$
2. Select a different encryption key for each employee:

   $e_0, e_1, e_2, \ldots$
3. Generate corresponding decryption keys for each employee:

   $d_0, d_1, d_2, \ldots$
4. Distribute n and d_i to each employee securely

Do you see a problem?

Common modulus attack - 2

Suppose Alice and Bob generate their keys using the same n.

Alice's keys are (n, e_a), Bob's keys are (n, e_b) and

$$\gcd(e_a, e_b) \ = \ 1$$

One sends the same message m to both Alice and Bob.

Eve can find out m from the ciphertext: m^{e_a} and m^{e_b}.

Because $\gcd(e_a, e_b) = \ 1$, Eve finds s_a and s_b such that, $e_a s_a + e_b s_b = 1$, as in Extended Euclidean Algorithm

Then she computes

$$(m^{e_a})^{s_a}(m^{e_b})^{s_b} = m^{e_a s_a + e_b s_b} = m$$

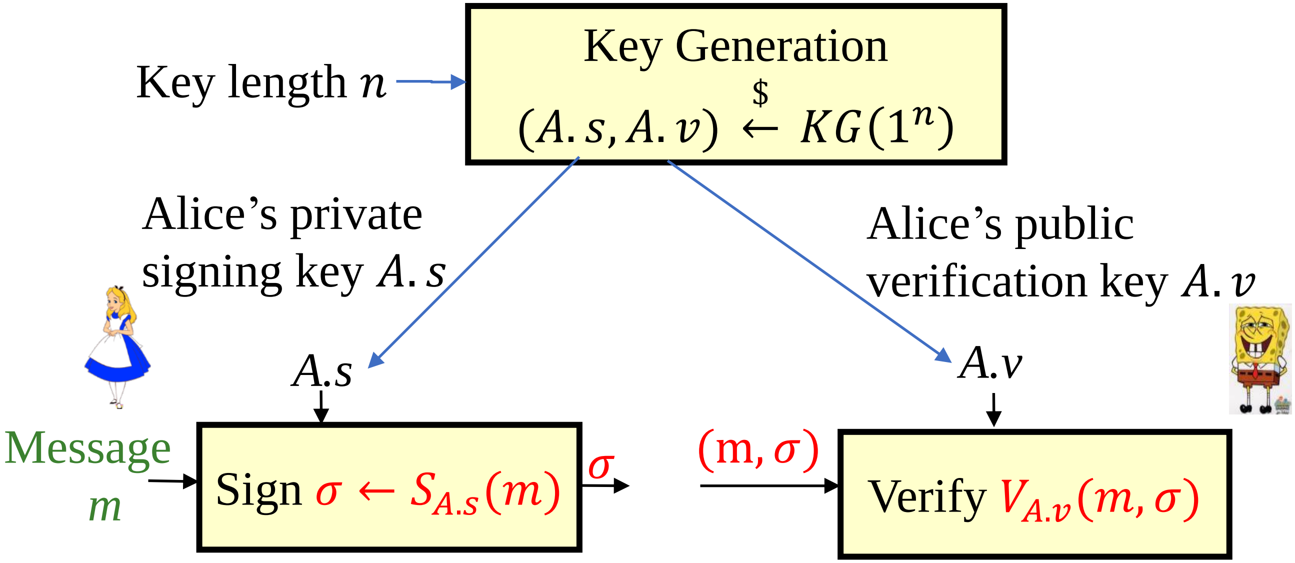
How does Bob know Alice's public key?

- 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 a trusted party's public key

Digital Signature

Public Key Digital Signatures



Key length n

Key Generation

$(A.s, A.v) \overset{\$}{\leftarrow} KG(1^n)$

Alice's private signing key $A.s$

Alice's public verification key $A.v$

$A.s$

$A.v$

Message m

Sign $\sigma \leftarrow S_{A.s}(m)$

σ

(m, σ)

Verify $V_{A.v}(m, \sigma)$

- Alice signs m, using $A.s$, a private, secret signature key
- Everyone can validate her signatures with her public key $A.v$

Correctness:

For every message m and valid key pair (s, v), $V_v\big(m, S_s(m)\big) = OK$

Digital Signatures Security: Unforgeability

- Given v, attacker cannot find any ‘valid’ (m, σ), i.e.,

$$V_v(m, \sigma) = OK$$

Even when attacker can select m' and receive $\sigma' = S_s(m')$

Note that $m' \neq m$

Digital Signature provides authentication, integrity **and** evidence/non-repudiation

MAC only provides authentication and integrity. No evidence, can repudiate

Digital Signature Scheme Security

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

```
$(s, v) \overset{\$}{\leftarrow} \mathcal{S}.\mathcal{KG}(1^l)$ ;
$(m, \sigma) \overset{\$}{\leftarrow} \mathcal{A}^{\mathcal{S}.\mathcal{Sign}_s(\cdot)}(v, 1^l)$;
return $(\mathcal{S}.\mathcal{Verify}_v(m, \sigma) \wedge (\mathcal{A}$ 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:*

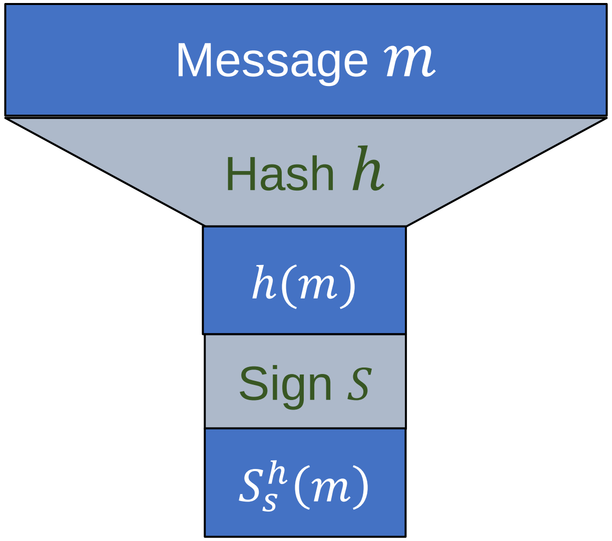
$$\varepsilon^{EUF-Sign}_{\mathcal{S},\mathcal{A}}(1^l) \equiv \Pr\left(EUF^{Sign}_{\mathcal{A},\mathcal{S}}(1^l)(1^l) = \textsc{True}\right) \tag{1.32}$$

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
- Hash-then-sign
  - Use collision resistant hash function (CRHF)
  - Handle messages of arbitrary lengths
- Sign: $\sigma = \text{RSA}.S_s(m) = h(m)^s \bmod n$
- Verify:

```
$\text{RSA}.V_v(m,\sigma)$
{
    if $h(m) = \sigma^v \bmod n$
        return OK
    else
        return FAIL
}
```



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

Pizza

- Motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

Trust model

- Web of trust OpenPGP (Pretty Good Privacy)
  - Decentralized model (no central authority)
  - Multiple trust levels (don’t trust, don’t know, marginal, full)
  - The user decide if a key is valid
    - Do you trust a friend of your friends? How about their friends?

- Public Key Infrastructure (PKI)
  - Centralized
  - Issued by a trusted third party (CA)
    - Certificate authority
  - Must trust issuer
    - If CA is compromised …

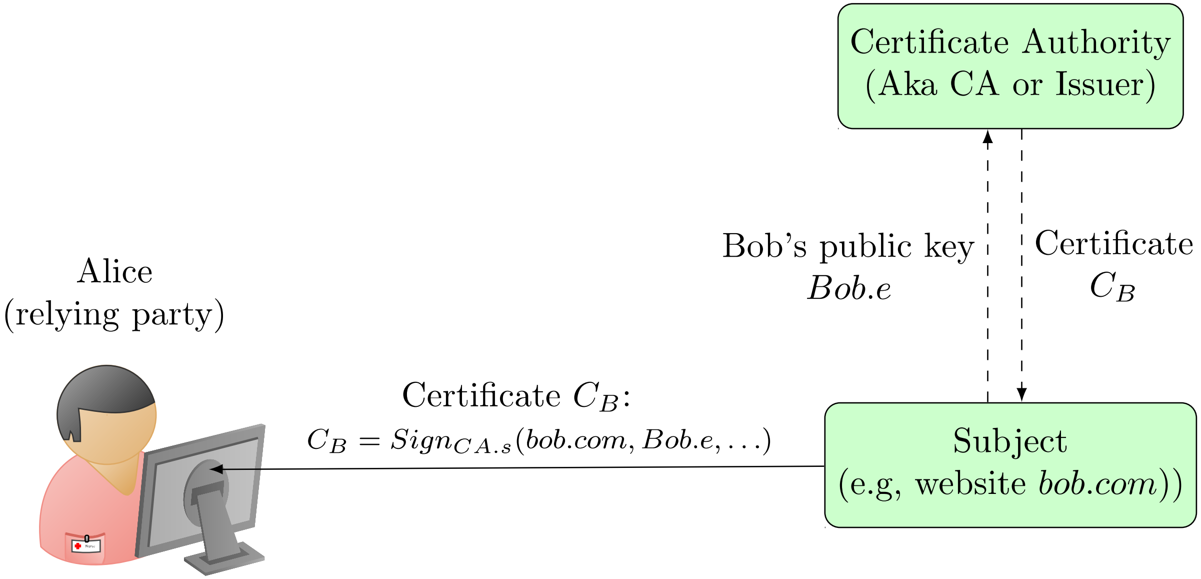
Public Key Infrastructure
PKI

Public keys are very useful…

- Secure web connections
- Software signing (against malware)
- Secure messaging, email
- Cryptocurrency and blockchains.
- But …
    - How do we know the PK of an entity?
        - Mainly: signed by a trusted Certificate Authority
        - E.g., in TLS, browsers maintain list of ‘root CAs’

Public Key Certificates & Authorities

- *Certificate:* signature by Issuer / Certificate Authority (CA) over subject's public key and attributes
- Attributes: identity (ID) and others…
  - Validated by CA (liability?)
  - Used by **relying party** for decisions (e.g., use this website?)



Certificate Authority (Aka CA or Issuer)
Bob's public key $Bob.e$
Certificate C_B
Alice (relying party)
Certificate C_B:
$C_B = Sign_{CA.s}(bob.com, Bob.e, \ldots)$
Subject (e.g, website $bob.com$))

Certificates are all about **Trust**

- Certificate
  - CA attests that Bob's public key is $Bob.e$

$$C_{Bob} = Sign_{CA.s}(Bob.com, Bob.e, \dots)$$

- Do we **trust** this attestation to be true?

Both Windows and Linux have a list of Trusted Root CAs

X.509: An ITU-T Standard for PKI

Certificate

- Version, Serial Number, Algorithm ID
- Issuer
- Validity (Not Before, Not After)
- Subject
- Subject Public Key Info
  - Public Key Algorithm, Subject Public Key, etc.
- Issuer Unique Identifier (optional)
- Subject Unique Identifier (optional)
- Extensions (optional)

Certificate Signature Algorithm

Certificate Signature

Files: .pem, .cer (crt), .p7b, .p12, .pfx

Rogue Certificates

- Rogue cert: equivocating or misleading (domain) name
- Attacker goals:
  - Impersonate: web-site, phishing email, signed malware..
  - Equivocating (same name): circumvent name-based security mechanisms, such as *Same-Origin-Policy (SOP), blacklists, whitelists*, *access-control* …
  - Name may be misleading even if not equivocating
- Types of misleading names (‘cybersquatting’):
  - Combo names: bank.com vs. accts-bank.com, bank.accts.com, …
  - Domain-name hacking: accts.bank.com vs. accts-bank.com, … or accts-bank.co
  - Homographic: paypal.com [l is L] vs. paypaI.com [i is I]
  - Typo-squatting: bank.com vs. banc.com, baank.com, banl.com,…

PKI Failures

- Certificates are valid until they expire
- There could be PKI failures and certificates must be revoked
  - Subject key exposure
  - CA failure
  - Cryptanalysis certificate forgery
    - Find collisions in the hash function used in the HtS paradigm,
    - or exploit some vulnerability in the digital signature scheme used for signing
- There should be revocation mechanisms

Some Infamous PKI Failures

2001	VeriSign: attacker gets code-signing certs
2008	Thawte: email-validation (attackers' mailbox)
2008,11	Comodo not performing domain validation
2011	DigiNotar compromised, 531 rogue certs (discovered); a rogue cert for _*.google.com_ used for MitM against 300,000 Iranian users.
2011	TurkTrust issued intermediate-CA certs to users
2012	Trustwave issued intermediate-CA certificate for eavesdropping
2013	ANSSI, the French Network and Information Security Agency, issued intermediate-CA certificate to MitM traffic management device
2014	India CCA / NIC compromised (and issued rogue certs)
2015	CNNIC (China) issued CA-cert to MCS (Egypt), who issued rogue certs. Google and Mozilla removed CNNIC from their root programs.
2013-17	Audio driver of Savitech install root CA in Windows
2015,17	Symantec issued unauthorized certs for over 176 domains, causing removal from all root programs.
2019	Mozilla, Google *browsers* block *customer-installed* Kazakhstan root CA (Qaznet)
2019	Mozilla, Google revoke intermediate-CA of DarkMatter, and refuse to add them to root program

PKI Goals/Requirements

Trustworthy issuers: Trust anchor/root CAs and Intermediary CAs; Limitations on Intermediary CAs (e.g., restricted domain names)

Accountability: identify issuer of given certificate

Timeliness: limited validity period, timely **revocation**

Transparency: public log of all certificate; no ‘hidden’ certs!

Non-Equivocation: one entity – one certificate

Privacy: why should CA know which site I use?

Covered Material From the Textbook

- Chapter 1: Section: 1.4
- Chapter 6: Sections 6.4, 6.5 (except 6.5.6 and 6.5.7), and 6.6 (except RSA with message recovery)
- Chapter 8: Section 8.1

RSA examples

$p = 19,\ q = 31,\ n = p \cdot q = 589$

$\varphi(n) = 18 * 30 = 540$ (factors in 540: 2, 3, 5)

$e = 13 \qquad d = e^{-1} = 457 \bmod 540$ (not mod 589)

$m = 387$

Encryption: $c = m^e = 387^{13} = 368 \bmod 589$

Decryption: $m' = c^d = 368^{457} = 387 \bmod 589$

$m = 323$

Encryption: $c = m^e = 323^{13} = 228 \bmod 589$

Decryption: $m' = c^d = 228^{457} = 323 \bmod 589$

Factoring n vs exposing d

Theorem: Factoring n is equivalent to exposing d

If one can factor n, d can be computed from e

If d is known, one can factor n

$$\begin{aligned} e \cdot d &= 1 \bmod \varphi(n) \\ e \cdot d - 1 &= k\,\varphi(n) \end{aligned}$$

$a^{ed-1} = 1 \bmod n$ for all a that is coprime to n

Let $(e \cdot d - 1) = t \cdot 2^s$ and t is odd

Select a that is coprime to n, compute

$$r_1 = a^{\frac{ed-1}{2}}, r_2 = a^{\frac{ed-1}{4}}, r_3 = a^{\frac{ed-1}{8}}, \quad \ldots \quad , r_s = a^t \quad (\mathrm{mod}\ n)$$

Note $r_0 = 1$ and $r_{i-1} = r_i^2$ for $i = 1,2,\ldots,s$

With 50% chance, $\exists i, r_i \neq \pm 1$ for $i = 1,2,\ldots,s$.

Find the smallest i such that $r_i \neq \pm 1$. $\gcd(r_i - 1, n)$ is a non-trivial factor of n.

Example: factoring n if d is known

$p = 19, \ q = 31, \ n = p \cdot q = 589$

$\varphi(n) = 18 * 30 = 540$ (factors in 540: 2, 3, 5)

$e = 13 \qquad d = e^{-1} = 457 \bmod 540$ (not mod 589)

$(d \cdot e - 1) = 5940 = 2^2 * 1485$ so $(s = 2, \ t = 1485)$

The exponents to be used in test are $2970 = \frac{5940}{2}$ and $1485 = \frac{5940}{4}$.

Suppose randomly pick 90. 90 is coprime to 589.

$$r_1 = 90^{2970} = 1, r_2 = 90^{1485} = 94 \bmod 589$$

94 is a non-trivial square-root of 1 mod n

93 and 95 have common factors with 589.

gcd(93, 589) = 31 gcd(95, 589) = 19