



CSE 127: Computer Security

Public Key Infrastructure

Kirill Levchenko

October 26, 2017

Symmetric Primitives

$$H = \text{H}(M)$$

$$A = \text{MAC}_k(M)$$

$$C = \text{Enc}_k(M)$$

$$M = \text{Dec}_k(C)$$

Hash Functions

- ❖ A (cryptographic) hash function maps arbitrary length input into a fixed-size string

$$H = H(M)$$

$$H \in \{0, 1\}^m$$

$$M \in \{0, 1\}^*$$

- ❖ **Property:** *Collision resistance*

- Computationally hard to find two inputs with same hash value

Message Authentication Code

$$A = \text{MAC}_k(M) \quad \begin{array}{l} A \in \{0, 1\}^m \\ M \in \{0, 1\}^* \end{array}$$

❖ **Property:** unforgeability

- Computationally hard to create A for M without knowing key

❖ **Non-property:** No secrecy guarantees!

- A not guaranteed to reveal nomething about M
 - In practice, usually A does not leak any information about M

Symmetric Encryption

$$C = E_k(M)$$

$$C \in \{0, 1\}^{iB}$$

$$M = D_k(C)$$

$$M \in \{0, 1\}^{iB}$$

$$k \in \{0, 1\}^n$$

❖ **Property:** C reveals nothing about M

- Computationally hard to distinguish two inputs $M \neq M'$ based on ciphertext only

❖ **Non-property:** No integrity guarantees!

- May be possible to change M in known way by changing C
 - Stream ciphers particularly vulnerable to tampering

Asymmetric Primitives

$$(K, k) \leftarrow \text{Keygen}(r) \quad (K, k) \leftarrow \text{Keygen}(r)$$

$$C = E_K(M) \quad S = S_k(M)$$

$$M = D_k(C) \quad V_K(M, S)$$

Asymmetric Encryption

$$C = E_K(M)$$

$$C \in \{0, 1\}^m$$

$$M = D_k(C)$$

$$M \in \{0, 1\}^m$$

❖ **Property:** C reveals nothing about M

- Computationally hard to distinguish two inputs $M \neq M'$ based on ciphertext only

❖ **Non-property:** No integrity guarantees!

- May be possible to change M in known way by changing C

Digital Signatures

$$S = S_k(M) \quad S \in \{0, 1\}^\ell$$

$V_K(M, S)$ ← Returns
“yes” if valid,
“no” otherwise

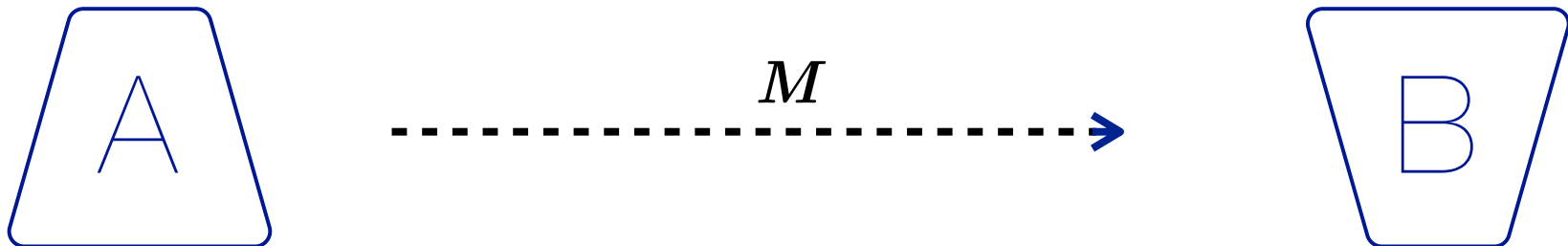
❖ **Property: unforgeability**

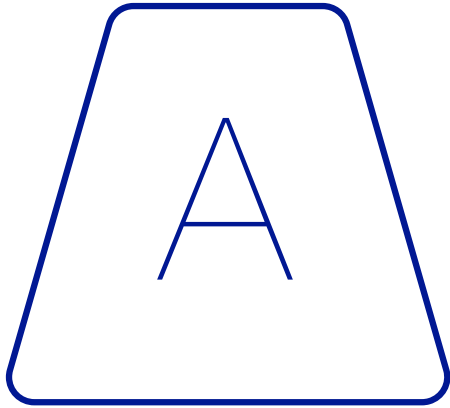
- Computationally hard to create S for M without knowing key

❖ Only need public key K to verify signature

Using Cryptography

- ❖ Alice wants to send (a plaintext) M to Bob
- ❖ Alice and Bob know each other's public keys
- ❖ **Want: Secrecy** (only Bob can read message)
- ❖ **Want: Authenticity + Integrity** (Bob knows it's from Alice)





generate random
ephemeral symmetric key

$$k' \stackrel{\$}{\leftarrow} \mathcal{K}_{\text{sym}}$$

sign hash of message using
Alice's private signing key

$$\sigma \leftarrow S_{k_A}(H(M))$$

encrypt plaintext and signature
using ephemeral symmetric key

$$C \leftarrow \langle E_{K_B}(k'), E_{k'}(\langle M, \sigma \rangle) \rangle$$

encrypt ephemeral key using
Bob's public encryption key

$$C = \langle E_{K_B}(k'), E_{k'}(\langle M, \sigma \rangle) \rangle$$

decrypt ephemeral key using
own private encryption key

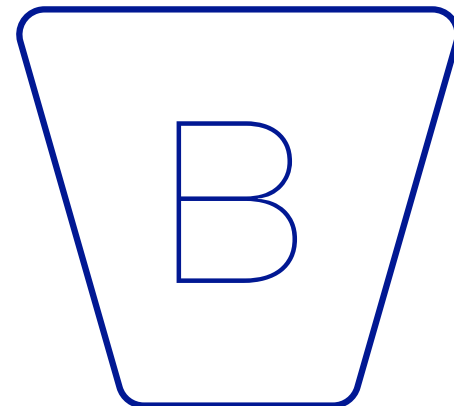
$$k' \leftarrow D_{K_B}(E_{K_B}(k'))$$

decrypt message and signature
using ephemeral symmetric key

$$\langle M, \sigma \rangle \leftarrow D_{k'}(E_{k'}(\langle M, \sigma \rangle))$$

verify signature on message
hash using Alice's public key

$$\text{OK} \stackrel{?}{=} V_{K_A}(H(M), \sigma)$$



Using Cryptography

- ❖ Alice and Bob got **secrecy + integrity + authenticity** and everyone lived happily ever after, right?
- ❖ *Almost ...*
- ❖ Let's try to understand exactly what we achieved

What Alice Knows

- ❖ While message is on its way to Bob —
 - No one can decrypt message without ephemeral key
 - No one knows the ephemeral key but Alice
 - No one but Bob can decrypt ephemeral key
- ❖ When Bob receives message —
 - Bob can decrypt ephemeral key, then decrypt message
 - Bob can verify that Alice signed the plaintext

What Bob Knows

- ❖ Upon receiving the ciphertext C —
 - At some point in the past someone encrypted a symmetric key using his public key
 - Decrypting the remainder of the message using the symmetric key yields a message M and signature σ
 - At some point the past Alice signed a (hash of) message M

**Where is the secrecy,
integrity, and authenticity?**

What Alice Knows

- ❖ Before Bob receives message —
 - No one but Alice knows the plaintext
- ❖ If Bob receives message —
 - Bob knows the plaintext
 - Bob knows Alice signed plaintext
 - Therefore: Bob knows that Alice knows plaintext

What Bob Knows

- ❖ At some point in the past Alice signed plaintext M
- ❖ Alice knows the plaintext (because she signed it)

Alice Can't Control . . .

- ❖ *Whether* Bob receives the message
- ❖ *When* Bob receives the message
- ❖ *How many times* Bob receives the message
- ❖ **Whether Bob keeps the message secret**
 - Cryptography only promises that *knowing ciphertext C alone* does not reveal anything about the plaintext

Bob Doesn't Know ...

- ❖ *Who* sent the message
- ❖ *When* the message was sent
- ❖ *Who else* knows the plaintext

What We Have

❖ **Secrecy**

- Alice depends Bob to keep plaintext secret
- Bob depends on Alice to keep plaintext secret

❖ **Integrity and Authenticity**

- Bob knows Alice signed plaintext

What Does Signing Mean?

Signing is a mechanical operation that has *no meaning in itself*.

What Does Signing Mean?

- ❖ **What Cryptography promises:**
Only someone who knows the private key can create a signature that verifies using the corresponding public key
- ❖ Meaning of a digital signature is a matter of convention
 - **Code signing:** signer attests she is developer of code
 - **Email signing:** signer attests she wrote message
 - **Certificate signing:** *(coming up next!)*
- ❖ Both signer and verifier should agree on meaning

Timing

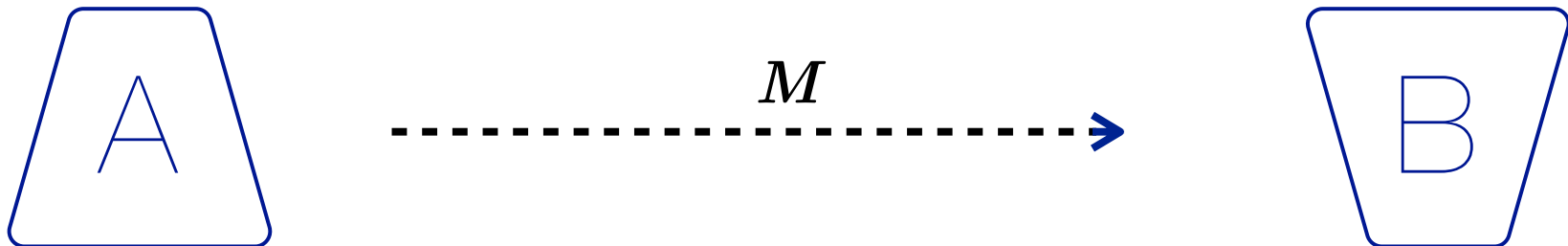
- ❖ Attacker can replay and delay/block (MitM) messages
 - **MitM:** Man-in-the-middle attacks
- ❖ Protocols should be robust against replay
 - **Idempotent protocol:** receiving message twice has no effect
 - Bad protocol example: “Transfer \$100 to account 34632.”
- ❖ Protocols should be robust against arbitrary delay
 - Example: lost check problem

Public Key

Infrastructure

Using Cryptography

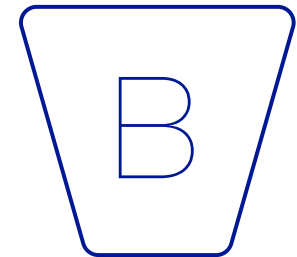
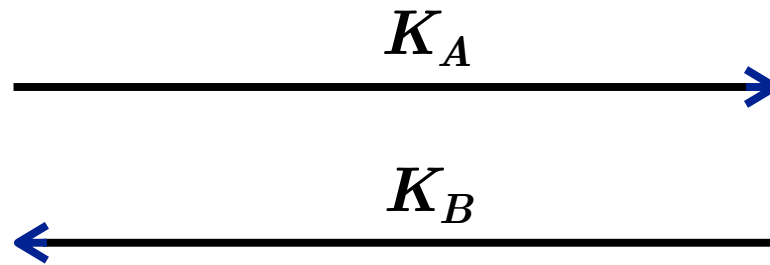
- ❖ Alice wants to send (a plaintext) M to Bob
- ❖ Alice and Bob know each other's public keys
- ❖ **Want: Secrecy** (only Bob can read message)
- ❖ **Want: Authenticity + Integrity** (Bob knows it's from Alice)



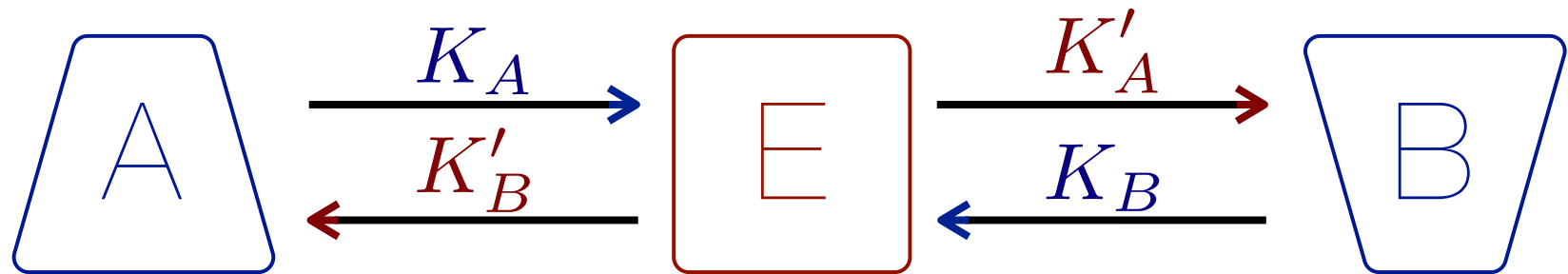
Getting Public Keys

- ❖ Alice and Bob need a way to get each other's public key
- ❖ Alice can send an unencrypted message to Bob:
“Hey, send me your public key. Here's mine.”
- ❖ Bob sends Alice his public key
- ❖ They communicate securely ever after?

What They Want to Happen



What Happens Instead



- ❖ If Eve has man-in-the-middle capability, she can impersonate Alice to Bob and Bob to Alice
- ❖ Eve becomes invisible gateway between them
- ❖ Alice and Bob have no idea Eve is there

Key Verification

- ❖ Alice and Bob need a way to know that each has the *real* public key of the other
- ❖ **Ideal solution:** Alice and Bob meet in person and exchange public keys
- ❖ **Equivalent:** Alice and Bob meet in person and exchange public key **fingerprints**
 - **Key fingerprint:** cryptographic hash of public key
 - Key itself can be sent in the open

Key Verification

- ❖ **Problem with ideal:** Alice and Bob need to meet
 - Impractical to meet and verify key of everyone you talk to
- ❖ **Practical solution:** Use a trusted intermediary
 - Alice and Bob have already exchanged keys with Charlie
 - Charlie sends signed message with Alice's key to Bob
 - Charlie sends signed message with Bob's key to Alice
 - Alice and Bob trust Charlie to send the real public keys
 - Alice and Bob now have each other's public key

Key Verification Improved

- ❖ Charlie creates a **certificate**:
“I, Charlie, verified that Alice’s key is ... ”
- ❖ Charlie signs the message and gives it to Alice
 - Alice now has certificate attesting to her public key
- ❖ Alice sends Bob her public key and Charlie’s certificate
- ❖ Bob verifies signature on certificate
- ❖ Bob trusts Charlie, accepts public key from Alice

Who is Charlie?

- ❖ PGP world: Charlie is any other person you trust
- ❖ SSL world: Charlie is a Certificate Authority

PGP Web of Trust

- ❖ PGP allows one user to attest to the accuracy of another user's public key — **key signing**
 - PGP does not use the term “certificate”
 - Public key has set of signatures (certificates)
- ❖ A user can indicate how much she trusts another user's signature on a key
- ❖ We signed your keys with the TA key
 - This means we are attesting that the key is your key

PGP Web of Trust

- ❖ Alice's signature on Bob's PGP key means Alice has verified that this is really Bob's key
 - Email address associated with key is Bob's address
 - Name associated with key is Bob
- ❖ Other people who trust Alice can use her signature on Bob's key to be sure it is Bob's key

Midterm Extra Credit!

- ❖ In October 27, 2017 discussion, you will sign each other's keys
- ❖ You *must* verify the other person's identity before signing
 - Verify identity (name)
 - Sign public key
 - Send them signature
- ❖ 1 point of extra credit per signature
 - Up to 10 points

Certificate Authorities

- ❖ **Certificate Authority:** A trusted authority that signs keys
- ❖ Your browser ships with public keys of trusted CAs
 - Who makes this list?
- ❖ CA model used to sign certificates used on Web

Certificate Semantics

- ❖ Issuer (CA) attests:
 - Public key belongs to subject
C=US, ST=California, L=La Jolla,
O=University of California, San Diego,
OU=ACT Data Center, CN=*.ucsd.edu
 - The domain listed in CN belongs to subject
- ❖ Certificate has expiration and limitations on use

Data:

Version: 3 (0x2)

Serial Number:

0f:77:30:d4:eb:75:d6:c4:22:1e:4b:a1:f6:16:2b:83

Signature Algorithm: sha1WithRSAEncryption

Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com,
CN=DigiCert High Assurance CA-3

Validity

Not Before: Sep 7 00:00:00 2012 GMT

Not After : Nov 11 12:00:00 2015 GMT

Subject: C=US, ST=California, L=La Jolla,
O=University of California, San Diego,
OU=ACT Data Center, CN=*.ucsd.edu

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (2048 bit)

Modulus (2048 bit):

00:cf:73:a9:a0:dd:69:de:98:c5:65:2d:fa:c0:dc:
47:ed:ff:f9:0b:16:3a:ee:e4:74:6a:de:26:37:7b:
ce:f7:de:3e:50:25:13:49:23:ec:c8:b3:19:5f:05:
9e:05:72:41:a9:f7:26:b3:d2:bd:88:37:51:e8:d5:
c3:01:d9:c2:15:bf:eb:87:a3:4b:80:3b:6c:f6:ce:
c5:78:4c:d2:b3:24:af:3d:8b:d8:ba:b9:c9:eb:16:
b4:83:68:06:b6:1e:96:0e:2e:1c:78:91:41:b4:8d:
3c:fe:2a:f5:93:ac:e5:bd:98:78:e5:db:4a:c2:88:
46:3a:1f:1e:07:fd:79:8a:96:c7:e9:b7:05:4d:40:
5d:4d:52:2c:e4:bc:6b:eb:2c:3e:09:e1:27:49:1b:
16:eb:53:cf:d9:df:8f:35:71:b1:10:1f:0b:7f:c1:

X509v3 extensions:

X509v3 Authority Key Identifier:

keyid:50:EA:73:89:DB:29:FB:10:8F:9E:E5:01:20:D4:DE:79:99:4

X509v3 Subject Key Identifier:

42:DB:21:92:81:A4:E0:44:A4:61:43:51:06:4C:26:37:21:9E:D5:E

X509v3 Subject Alternative Name:

DNS:*.ucsd.edu, DNS:ucsd.edu

X509v3 Key Usage: critical

Digital Signature, Key Encipherment

X509v3 Extended Key Usage:

TLS Web Server Authentication, TLS Web Client Authentication

X509v3 CRL Distribution Points:

URI:<http://crl3.digicert.com/ca3-g14.crl>

URI:<http://crl4.digicert.com/ca3-g14.crl>

X509v3 Certificate Policies:

Policy: 2.16.840.1.114412.1.1

CPS: <http://www.digicert.com/ssl-cps-repository.htm>

User Notice:

Explicit Text:

Authority Information Access:

OCSP - URI:<http://ocsp.digicert.com>

CA Issuers - URI:<http://cacerts.digicert.com/DigiCertHighA>

X509v3 Basic Constraints: critical

-- -- --

Revocation

- ❖ What happens if someone steals your private key?
- ❖ They can impersonate you and read messages encrypted to you
- ❖ Key expiration helps with this but not enough
- ❖ CA and PGP PKIs support **revocation**
 - Owner says: “I, Alice, revoke my public key ... do not use it.”
 - Signs revocation with her private key
 - Others can verify Alice’s signature, stop using key

Revocation

- ❖ In CA model, Alice asks CA to revoke certificate
 - Alice does not need private key to do this
 - CAs publish a Certificate Revocation List (CRL)
- ❖ In PGP model, only Alice can revoke her own key
 - If Alice loses her private key, she can't revoke
 - Do not lose private key
 - Option: generate revocation with key, store in secure place

Revocation

- ❖ How does Bob know if Alice's key has been revoked?
- ❖ Bob asks Alice: "Has your key been revoked?"
- ❖ Alice sends signed message: "No."
- ❖ Does not work: if Alice's key has been compromised, then Eve could have forged the message "No."
- ❖ *Availability* of revocation list critical

Revocation Today

- ❖ Two Mechanisms: CRL and OCSP
- ❖ **Published CRL:** certificate also says whereto get CRL
 - What if CRL server is down?
- ❖ **Online Certificate Status Protocol:** Query CA about cert
- ❖ **OCSP stapling:** Web server includes recent OCSP cert

X509v3 extensions:

X509v3 Authority Key Identifier:

keyid:50:EA:73:89:DB:29:FB:10:8F:9E:E5:01:20:D4:DE:79:99:4

X509v3 Subject Key Identifier:

42:DB:21:92:81:A4:E0:44:A4:61:43:51:06:4C:26:37:21:9E:D5:E

X509v3 Subject Alternative Name:

DNS:*.ucsd.edu, DNS:ucsd.edu

X509v3 Key Usage: critical

Digital Signature, Key Encipherment

X509v3 Extended Key Usage:

TLS Web Server Authentication, TLS Web Client Authentication

X509v3 CRL Distribution Points:

URI:<http://crl3.digicert.com/ca3-g14.crl>

URI:<http://crl4.digicert.com/ca3-g14.crl>

X509v3 Certificate Policies:

Policy: 2.16.840.1.114412.1.1

CPS: <http://www.digicert.com/ssl-cps-repository.htm>

User Notice:

Explicit Text:

Authority Information Access:

OCSP - URI:<http://ocsp.digicert.com>

CA Issuers - URI:<http://cacerts.digicert.com/DigiCertHighA>

X509v3 Basic Constraints: critical

-- -- --