

Secure Communication: Wrap Up

Lecture 10

Schemes

- So far
 - SKE, MAC, PKE, Digital Signatures
 - Building blocks: block-ciphers (AES), hash-functions (SHA-256), Random Oracle heuristics (in RSA-OAEP, RSA-PSS)
 - Authenticated Encryption (SKE+MAC)
 - Hybrid Encryption (Efficient PKE)
 - More communication primitives:
 - Authenticated Key Exchange
 - Identity-Based Encryption

Authenticated Key Exchange

Authenticated Key Exchange

- To Establish a “fresh” key for a session

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys

Ensure session information is part of the signed messages

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys
- Additional considerations

Ensure session information is part of the signed messages

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys
- Additional considerations
 - Parties share a “password” but neither has a public-key

Ensure session information is part of the signed messages

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys
- Additional considerations
 - Parties share a “password” but neither has a public-key
 - Note: Then, must hide a party’s password from impostors

Ensure session information is part of the signed messages

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys
- Additional considerations
 - Parties share a “password” but neither has a public-key
 - Note: Then, must hide a party’s password from impostors
 - Hide a party’s identity if the other party is an impostor

Ensure session information is part of the signed messages

Authenticated Key Exchange

- To Establish a “fresh” key for a session
 - Key appears random even if keys in other sessions revealed
 - Parties should know who the other party in the session is
 - And that the other party is participating live in the session
 - Not hard if parties have public (signature verification) keys
- Additional considerations
 - Parties share a “password” but neither has a public-key
 - Note: Then, must hide a party’s password from impostors
 - Hide a party’s identity if the other party is an impostor
 - Ensure forward security: even if (long term) secret-keys revealed later, past sessions remain secure

Ensure session information is part of the signed messages

Password-Based Authenticated Key Exchange

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Security?

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Security?
 - Password has low entropy; can be guessed well

Password-Based Authenticated Key Exchange

- Variant 1: User has password, Server has public-key
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Security?
 - Password has low entropy; can be guessed well
 - Ideal: Allow adversary online password guessing, but no more

Password-Based Authenticated Key Exchange

- Variant 1: User has only password, Server has public-key too
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)

Password-Based Authenticated Key Exchange

- Variant 1: User has only password, Server has public-key too
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Identity protection: user id (in session-info) sent encrypted using a public-key from the server

Password-Based Authenticated Key Exchange

- Variant 1: User has only password, Server has public-key too
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Identity protection: user id (in session-info) sent encrypted using a public-key from the server
- Forward security: new-key will be exposed if server's long term key is compromised later. So use new-key for authentication, and derive a session key using (say) Diffie-Hellman Key Exchange

Password-Based Authenticated Key Exchange

- Variant 1: User has only password, Server has public-key too
 - Server sends a nonce, and if it has only signature key, then a signed encryption key for a CCA-secure PKE (can weaken)
 - User sends an encryption of (pwd,session-info,new-key,nonce)
 - Server sends back (session-info,nonce) over a secure (authenticated) channel using new-key (symmetric key)
- Identity protection: user id (in session-info) sent encrypted using a public-key from the server
- Forward security: new-key will be exposed if server's long term key is compromised later. So use new-key for authentication, and derive a session key using (say) Diffie-Hellman Key Exchange
- Several optimizations/enhancements possible

Password-Based Authenticated Key Exchange

Password-Based Authenticated Key Exchange

- Variant 2: Neither user has a public-key, just a shared password

Password-Based Authenticated Key Exchange

- Variant 2: Neither user has a public-key, just a shared password
- Much harder: checking equality of passwords without revealing the passwords to each other

Password-Based Authenticated Key Exchange

- Variant 2: Neither user has a public-key, just a shared password
- Much harder: checking equality of passwords without revealing the passwords to each other
- Needs to depend on some “set-up” to achieve “secure composition”

Password-Based Authenticated Key Exchange

- Variant 2: Neither user has a public-key, just a shared password
- Much harder: checking equality of passwords without revealing the passwords to each other
- Needs to depend on some “set-up” to achieve “secure composition”
 - e.g. set-up: a common random string (could be hard-coded into a standard)

Password-Based Authenticated Key Exchange

- Variant 2: Neither user has a public-key, just a shared password
- Much harder: checking equality of passwords without revealing the passwords to each other
- Needs to depend on some “set-up” to achieve “secure composition”
 - e.g. set-up: a common random string (could be hard-coded into a standard)
- Constructions based on DDH, lattices etc. known (skipped)

Identity-Based Encryption

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK, SK) pair

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation
 - Then: Can it generate a valid (PK,SK) pair for any PK?

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation
 - Then: Can it generate a valid (PK,SK) pair for any PK?
 - Identity-Based Encryption: a key-server (with a master secret-key) that can generate such pairs

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation
 - Then: Can it generate a valid (PK,SK) pair for any PK?
 - Identity-Based Encryption: a key-server (with a master secret-key) that can generate such pairs
 - Encryption will use the master public-key, and the receiver’s “identity” (i.e., fancy public-key)

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation
 - Then: Can it generate a valid (PK,SK) pair for any PK?
 - Identity-Based Encryption: a key-server (with a master secret-key) that can generate such pairs
 - Encryption will use the master public-key, and the receiver’s “identity” (i.e., fancy public-key)
 - In PKE, sender has to retrieve PK for every party it wants to talk to (from a trusted public directory)

Identity-Based Encryption

- In PKE, KeyGen produces a random (PK,SK) pair
- Can I have a “fancy public-key” (e.g., my name)?
 - But no one should be able to pick a PK and find an SK for it
- But suppose a trusted authority for key generation
 - Then: Can it generate a valid (PK,SK) pair for any PK?
 - Identity-Based Encryption: a key-server (with a master secret-key) that can generate such pairs
 - Encryption will use the master public-key, and the receiver’s “identity” (i.e., fancy public-key)
 - In PKE, sender has to retrieve PK for every party it wants to talk to (from a trusted public directory)
 - In IBE, **receiver has to obtain its SK** from the authority

Identity-Based Encryption

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)
 - Or, CCA security: also gets (guarded) access to decryption for honest parties’ IDs

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)
 - Or, CCA security: also gets (guarded) access to decryption for honest parties’ IDs
- IBE (even CPA-secure) can easily give CCA-secure PKE!

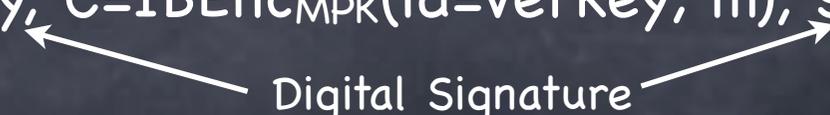
Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)
 - Or, CCA security: also gets (guarded) access to decryption for honest parties’ IDs
- IBE (even CPA-secure) can easily give CCA-secure PKE!
 - IBE: Can’t malleate ciphertext for one ID into one for another

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)
 - Or, CCA security: also gets (guarded) access to decryption for honest parties’ IDs
- IBE (even CPA-secure) can easily give CCA-secure PKE!
 - IBE: Can’t malleate ciphertext for one ID into one for another
 - $\text{PKEnc}_{\text{MPK}}(m) = (\text{verkey}, C = \text{IBEnc}_{\text{MPK}}(\text{id} = \text{verkey}; m), \text{sign}_{\text{signkey}}(C))$

Identity-Based Encryption

- Security requirement for IBE (will skip formal statement):
 - Environment/adversary decides the ID of the honest parties (in the beginning or later on)
 - Adversary can adaptively request SK for any number of IDs (which are not used for honest parties)
 - “Semantic security” for encryption with the ID of honest parties (CPA: with no access to decryption)
 - Or, CCA security: also gets (guarded) access to decryption for honest parties’ IDs
- IBE (even CPA-secure) can easily give CCA-secure PKE!
 - IBE: Can’t malleate ciphertext for one ID into one for another
 - $\text{PKEnc}_{\text{MPK}}(m) = (\text{verkey}, C = \text{IBEnc}_{\text{MPK}}(\text{id} = \text{verkey}; m), \text{sign}_{\text{signkey}}(C))$


Randomness Extractor

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group
 - A standard bit-string representation of a random group element may not be (pseudo)random

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group
 - A standard bit-string representation of a random group element may not be (pseudo)random
 - Can we map it to a pseudorandom bit string? Depends on the group...

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group
 - A standard bit-string representation of a random group element may not be (pseudo)random
 - Can we map it to a pseudorandom bit string? Depends on the group...
- Suppose a chip for producing random bits shows some complicated dependencies/biases, but still is highly unpredictable

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group
 - A standard bit-string representation of a random group element may not be (pseudo)random
 - Can we map it to a pseudorandom bit string? Depends on the group...
- Suppose a chip for producing random bits shows some complicated dependencies/biases, but still is highly unpredictable
 - Can we purify it to extract uniform randomness? Depends on the specific dependencies...

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group
 - A standard bit-string representation of a random group element may not be (pseudo)random
 - Can we map it to a pseudorandom bit string? Depends on the group...
- Suppose a chip for producing random bits shows some complicated dependencies/biases, but still is highly unpredictable
 - Can we purify it to extract uniform randomness? Depends on the specific dependencies...
- A general tool for purifying randomness: Randomness Extractor

Randomness Extractors

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length
- Constructions with short seeds

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length
- Constructions with short seeds
 - e.g. Based on expander graphs

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length
- Constructions with short seeds
 - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length
- Constructions with short seeds
 - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
 - Can be based on iterated-hash functions or CBC-MAC

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
 - “Optimal” in all parameters except seed length
- Constructions with short seeds
 - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
 - Can be based on iterated-hash functions or CBC-MAC
 - Statistical guarantee, if compression function/block-cipher is a random function/random permutation (not random oracle)

Randomness Extractors

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
 - 2-UHF is an example

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
 - 2-UHF is an example
- Useful in key agreement

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
 - 2-UHF is an example
- Useful in key agreement
 - Alice and Bob exchange a non-uniform key, with a lot of pseudoentropy for Eve (say, g^{xy})

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
 - 2-UHF is an example
- Useful in key agreement
 - Alice and Bob exchange a non-uniform key, with a lot of pseudoentropy for Eve (say, g^{xy})
 - Alice sends a random seed for a strong extractor to Bob, in the clear

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
 - 2-UHF is an example
- Useful in key agreement
 - Alice and Bob exchange a non-uniform key, with a lot of pseudoentropy for Eve (say, g^{xy})
 - Alice sends a random seed for a strong extractor to Bob, in the clear
 - Key derivation: Alice and Bob extract a new key, which is pseudorandom (i.e., indistinguishable from a uniform bit string)

Secure Communication in Practice

Secure Communication in Practice

- Can do at application-level

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
 - e.g. between OS kernels, or between network gateways

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
 - e.g. between OS kernels, or between network gateways
- Standards in either case

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
 - e.g. between OS kernels, or between network gateways
- Standards in either case
 - To be interoperable

Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
 - e.g. between OS kernels, or between network gateways
- Standards in either case
 - To be interoperable
 - To not insert bugs by doing crypto engineering oneself

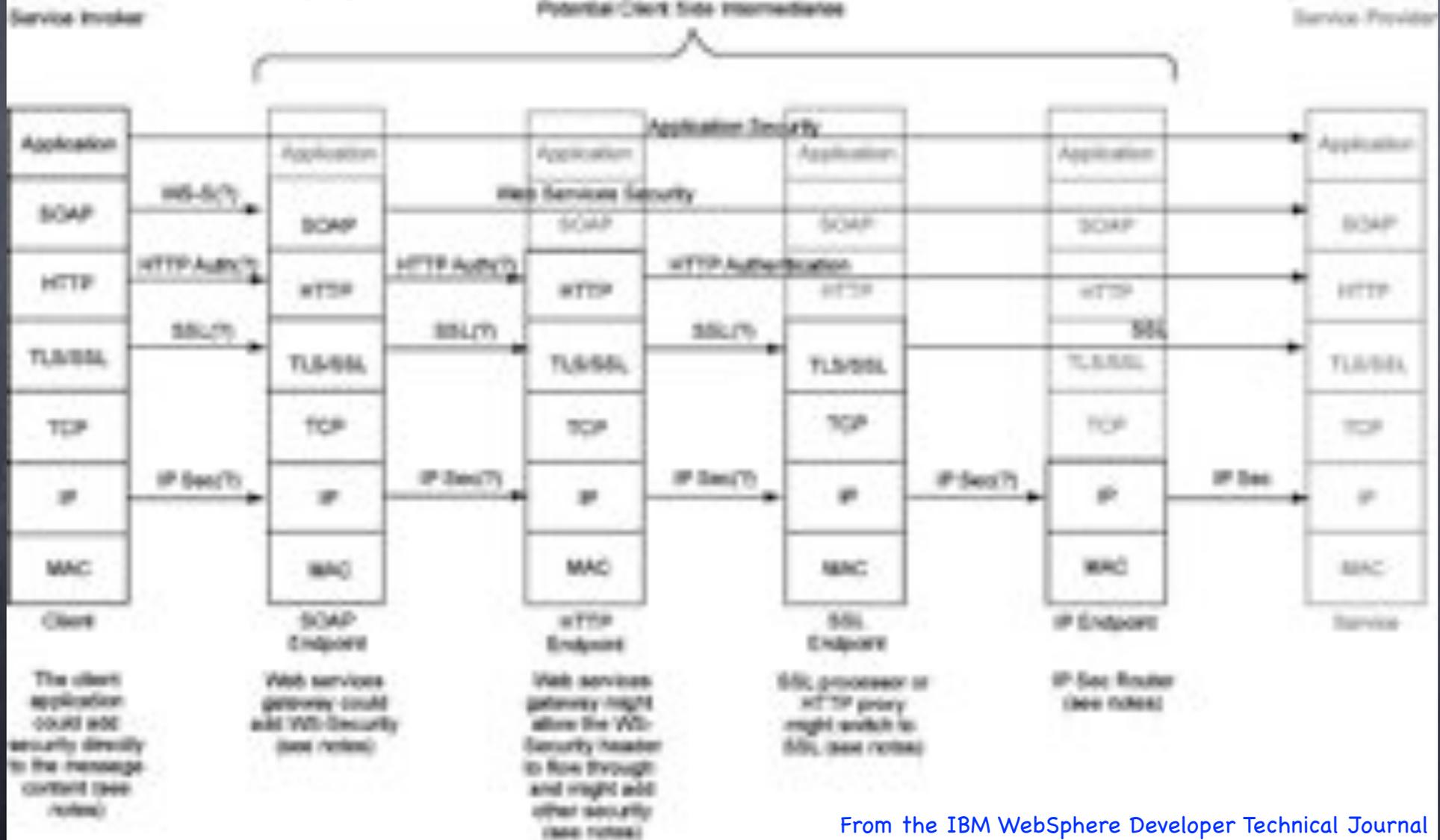
Secure Communication in Practice

- Can do at application-level
 - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
 - e.g. between OS kernels, or between network gateways
- Standards in either case
 - To be interoperable
 - To not insert bugs by doing crypto engineering oneself
 - e.g.: SSL/TLS (used in https), IPSec (in the “network layer”)

Security Architectures

(An example)

Security architecture (client perspective)



From the IBM WebSphere Developer Technical Journal

Secure Communication Infrastructure

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)
- Alice, Bob need to know each other's public-keys

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)
 - Alice, Bob need to know each other's public-keys
- But typically Alice and Bob engage in "transactions," exchanging multiple messages, maintaining state throughout the transaction

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)
 - Alice, Bob need to know each other's public-keys
- But typically Alice and Bob engage in "transactions," exchanging multiple messages, maintaining state throughout the transaction
 - Makes several efficiency improvements possible

Secure Communication Infrastructure

Secure Communication Infrastructure

- Secure Communication Sessions

Secure Communication Infrastructure

- Secure Communication Sessions
 - Handshake phase: establish private shared keys

Secure Communication Infrastructure

- Secure Communication Sessions



Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

Secure Communication Infrastructure

- Secure Communication Sessions



Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

Secure Communication Infrastructure

- Secure Communication Sessions



Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

Secure Communication Infrastructure

- Secure Communication Sessions



Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

- Client wants to establish a session with a server it “knows”.
Server is willing to talk to all clients

Secure Communication Infrastructure

- Secure Communication Sessions

Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

- Client wants to establish a session with a server it "knows".
Server is willing to talk to all clients

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

Secure Communication Infrastructure

- Secure Communication Sessions

Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

- Client wants to establish a session with a server it "knows".
Server is willing to talk to all clients

- Server has (certified) public-keys

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

Secure Communication Infrastructure

- Secure Communication Sessions

Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

- Client wants to establish a session with a server it "knows".
Server is willing to talk to all clients

- Server has (certified) public-keys

- Peer-to-peer session

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

Secure Communication Infrastructure

- Secure Communication Sessions

Essentially,
Authenticated
Key-Exchange

- Handshake phase: establish private shared keys

- Communication phase: use efficient shared-key schemes

- Client-server session

- Client wants to establish a session with a server it "knows".
Server is willing to talk to all clients

- Server has (certified) public-keys

- Peer-to-peer session

- Both parties have (certified) public-keys

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

A Simple Secure Communication Scheme

A Simple Secure Communication Scheme

- Handshake

A Simple Secure Communication Scheme

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

- In fact, a "stream-MAC": To send more than one message, but without allowing reordering

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

- In fact, a "stream-MAC": To send more than one message, but without allowing reordering

Recall "inefficient" domain-extension of MAC: Add a session-specific nonce and a sequence number to each message before MAC'ing

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

- In fact, a "stream-MAC": To send more than one message, but without allowing reordering

Recall "inefficient" domain-extension of MAC: Add a session-specific nonce and a sequence number to each message before MAC'ing

- Authentication + (CCA secure) Encryption: encrypt-then-MAC

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

- In fact, a "stream-MAC": To send more than one message, but without allowing reordering

Recall "inefficient" domain-extension of MAC: Add a session-specific nonce and a sequence number to each message before MAC'ing

- Authentication + (CCA secure) Encryption: encrypt-then-MAC

- stream-cipher, and "stream-MAC"

A Simple Secure Communication Scheme

- Handshake

- Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (server explicitly checks for replays?!)

- For authentication only: use MAC

- In fact, a "stream-MAC": To send more than one message, but without allowing reordering

Recall "inefficient" domain-extension of MAC: Add a session-specific nonce and a sequence number to each message before MAC'ing

- Authentication + (CCA secure) Encryption: encrypt-then-MAC

Authentication for free: MAC serves dual purposes!

- stream-cipher, and "stream-MAC"

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

TLS (SSL)

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-MD5 for MAC

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
 - For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
 - Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-MD5 for MAC

Server sends a certificate of its PKE public-key, which the client verifies

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-MD5 for MAC

Server sends a certificate of its PKE public-key, which the client verifies

Server also "contributes" to key-generation (to avoid replay attack issues): Roughly, client sends a key K for a PRF; a master key generated as $\text{PRF}_K(x,y)$ where x from client and y from server. SKE and MAC keys derived from master key

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-MD5 for MAC

Server sends a certificate of its PKE public-key, which the client verifies

Server also "contributes" to key-generation (to avoid replay attack issues): Roughly, client sends a key K for a PRF; a master key generated as $\text{PRF}_K(x,y)$ where x from client and y from server. SKE and MAC keys derived from master key

Uses MAC-then-encrypt! Not CCA secure in general, but secure with stream-cipher (and with some other modes of block-ciphers, like CBC)

TLS (SSL)

- Handshake
 - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
 - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- Authentication + (CCA secure) Encryption: encrypt-then-MAC
 - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-MD5 for MAC

Server sends a certificate of its PKE public-key, which the client verifies

Server also "contributes" to key-generation (to avoid replay attack issues): Roughly, client sends a key K for a PRF; a master key generated as $\text{PRF}_K(x,y)$ where x from client and y from server. SKE and MAC keys derived from master key

Uses MAC-then-encrypt! Not CCA secure in general, but secure with stream-cipher (and with some other modes of block-ciphers, like CBC)

Several details on closing sessions, session caching, resuming sessions ...

Coming Up

Coming Up

- Beyond communication

Coming Up

- Beyond communication
 - Secure Multi-party computation

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash
 - Fancy Signatures

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash
 - Fancy Signatures
 - Attribute-Based cryptography

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash
 - Fancy Signatures
 - Attribute-Based cryptography
- Tools

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash
 - Fancy Signatures
 - Attribute-Based cryptography
- Tools
 - Secret sharing, homomorphic encryption, bilinear-pairings...

Coming Up

- Beyond communication
 - Secure Multi-party computation
 - Electronic Voting
 - Private Information Retrieval
 - Anonymous Credentials & Digital Cash
 - Fancy Signatures
 - Attribute-Based cryptography
- Tools
 - Secret sharing, homomorphic encryption, bilinear-pairings...
 - Quantum cryptography (secure communication)