Asymmetric Key Encryption

Symmetric Vs. Asymmetric

Symmetric

- Single shared key
  - Encrypts and decrypts
- Faster than Asymmetric Encryption
- Problem: How do you share the key Securely?

Asymmetric

- 2 keys
  - 1 public
  - 1 private
- Asymmetric
  - Encrypt using public key then decrypt using private key
  - Encrypt using private key then decrypt using public key
- Does not require a shared secret

2 Keys (Public and Private)

- The generated keys are related to each other
- But one cannot be guessed from the other
- A message encrypted with the public key can ONLY be decrypted with the private key and vice versa
- What do I do with my keys?
    - Share your public key with the world
    - Hide your private key from everyone

Principles of Asymmetric Key Encryption

- Alice picks 2 prime numbers, p and q as the private key.
- She multiplies them together for the public key.
- Bob encrypts his message to Alice with the her public key.
- Alice Decrypts Bob’s message with her private key.
- Eve would need to know p and q to decrypt the message.
- Prime generation and multiplication is easy
- Prime factorization is hard (for now…)

Digital Signing (Basic)

- Not only do people wish to hide what they are sending, but they want to make sure the message really came from the stated sender
  - Did this really come from Alice?
- Asymmetric Key encryption allows “signing” of messages
- Signing -
  - Alice encrypts message with her **private** key
  - Everyone with the Alice’s **public** key can decrypt, not very secure
  - Assumed only Alice has access to her private key
  - Therefore, only Alice could have encrypted/signed the message

Certificate Authorities

- How do you get someone’s public key??
  - Could post it to a keybase
- Problem - Eve can give Alice a fake public key and claim it is Bob’s public key
- How does Alice know she really has Bob’s public key?
- Need a trusted 3rd party to distribute public keys - Certificate Authority
  - 3 Biggest: Comodo, Symantec, GoDaddy
- New Problem: How does Alice know a key really came from her trusted CA?

Certificate Authorities (2)

- Signing
  - The CA can sign the key before they send it to you
  - This makes it so you know that the key came from the CA
- Same problem as before
  - How do you get the CA's public key?
- Hard coded into web browsers
  - Set of trusted keys directly coded into your web browser (potential attack vector?).
  - CA's can be forced to create a “valid” key for someone (the government?)

Use in Conjunction with Symmetric Key Encryption

- Why??
  - Asymmetric is pretty great, why bother with symmetric?
- Speed
- Asymmetric is extremely slow compared to symmetric
  - This is important for large files
- Don’t want to encrypt an entire message using asymmetric key encryption
- Ideally, users exchange a symmetric key using asymmetric encryption then use the shared key from then on

Sending a message (more advanced)

Alice wants to send a message to Bob:

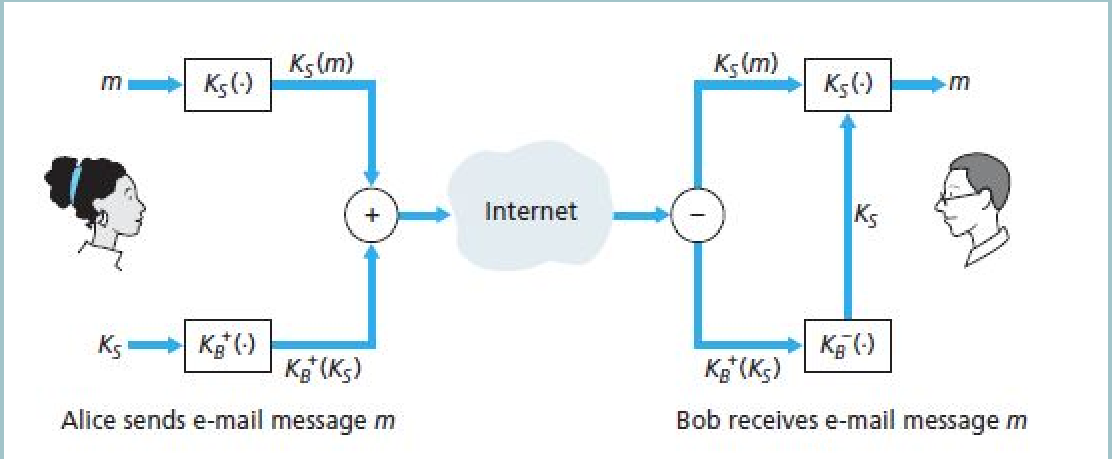
1. Alice generates a symmetric key (session key) K_s
2. Alice encrypts message with K_s
3. Alice encrypts K_s with Bob's public key K_B^+
4. Concatenate encrypted key onto encrypted message

Bob receives the package:

1. de-concatenates message
2. decrypts K_s with his private key K_B^-
3. decrypts entire message with K_s

This allows Alice to implement higher security measures while still getting the message to Bob quickly

Sending a message (more advanced)



Use in Conjunction with Hashing

- Why?
- Speed, same as before
- Don’t want to encrypt an entire message using asymmetric key encryption
- Hash the message, then sign the hash

Signing a message (More Advanced)

Alice wants to sign her message to Bob:

1. Alice hashes the message
2. Alice signs the hash H, with her private key K_A^-
3. Concatenate message and signed hash, send package

Bob receives the package:

1. Bob decrypts the hash H_a, with K_A^+. This proves it was signed by the real Alice.
2. Bob generates a hash of the message H_b
3. Bob compares H_b with H_a

This allows Bob to be sure the message was unchanged and definitely came from Alice

Signing a message (More Advanced)

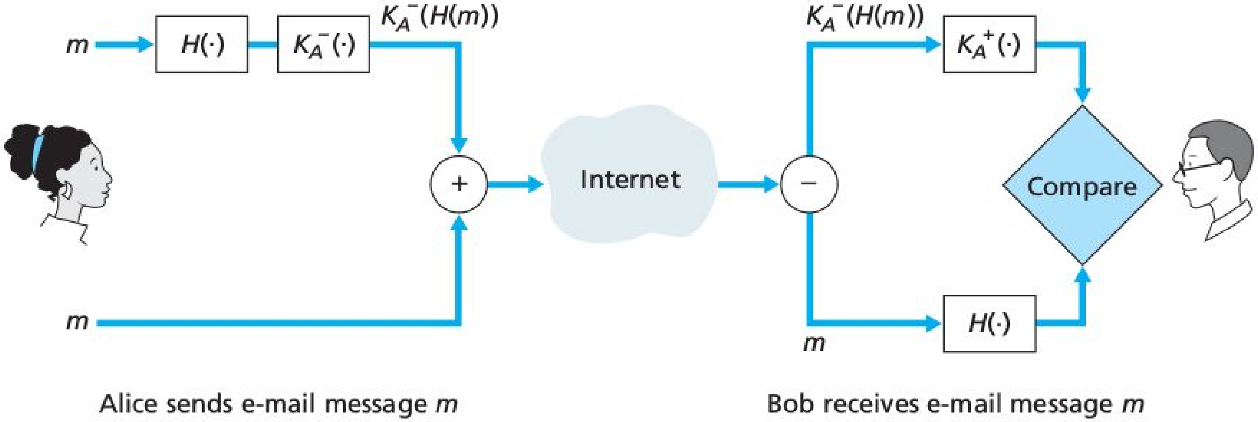


Figure 8.20 ♦ Using hash functions and digital signatures to provide sender authentication and message integrity

Super Secure Communication

Alice wants to send a message to Bob:

1. Alice hashes the message, H_a
2. Alice signs the message, ie encrypts H_a with K_A^-
3. Concatenates the message and signature together
4. Alice generates a session key K_s
5. Alice encrypts the concatenation with K_s
6. Alice encrypts K_s with Bob's public key K_B^+
7. Concatenates encrypted key and encrypted package together sends

Super Secure Communication

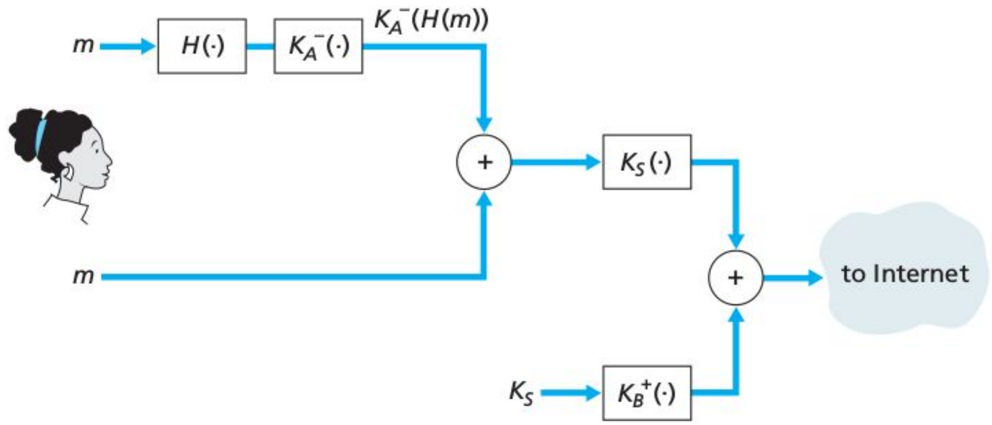


Figure 8.21 ♦ Alice uses symmetric key cyptography, public key cryptography, a hash function, and a digital signature to provide secrecy, sender authentication, and message integrity

Super Secure Communication (2)

Bob receives the package:

1. Bob de-concatenates the encrypted K_s from the encrypted package
2. Bob uses K_B^- to decrypt K_s
3. Bob uses K_s to decrypt the package
4. Bob de-concatenates the encrypted hash H_a from the message
5. Bob decrypts H_a using K_A^+
6. Bob generates a hash of the message H_b
7. Compare H_a and H_b

All together this provides a communication form that hides data, ensures the data has not been altered, and verifies the sender.

A Ton of Work Though...

- A lot of steps to just send a message. What if you forget a step or do something out of order?
- A program that uses this is PGP, pretty good privacy, which implements the steps listed above to make sending an an encrypted email easier
- Most secure communication over the internet utilizes asymmetric key encryption
- For Example:
  - SSL
  - HTTPS
  - SSH