Network Security
Authentication and Digital Signature
CME451 Tutorial 6

Hao Zhang
(Graduate Teaching Fellow)

Department of Electrical & Computer Engineering
University of Saskatchewan

Feb 10, 2017

*Most contents are from William Stallings, *Data and Computer Communications*, 8th edition, 2007 Pearson Education Inc.

Network Encryption

- Encrypt messages against passive attacks.
- Symmetric encryption:
  - Sender and receiver share the encryption key.
  - DES, 3DES, AES, ...
  - Key distribution.
- Asymmetric encryption (public-key encryption)
  - Public key made for others to use.
  - Private key known only to its owner.
  - RSA, ...
  - Sender encrypt message using receiver's public key.
  - Receiver decrypt the message using private key.
  - Help key distribution of symmetric encryption.

Network Encryption

Comparison

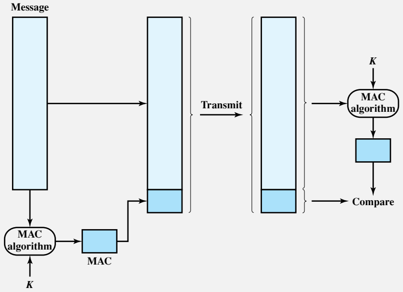
- **(Fallacy)** Public-Key encryption is more secure than symmetric encryption.
  - Length of key, computation intensity of crack.
  - Nothing in principle about one superior to another.
- **(Fallacy)** Public-Key encryption made symmetric encryption obsolete.
  - Public-Key encryption are more computational intensive.
- **(Fallacy)** Key distribution is trivial when using public-key encryption.
  - Symmetric encryption: handshake involved in key distribution center.
  - Public-Key encryption also needs protocol, involves central agent.

Authentication

- Protect against active attacks.
- Allow communication parties to verify the received messages are authentic.
  - Contents have not been altered.
  - Source is authentic.
  - Timeliness (delayed or replayed)
  - Sequence relative to other messages.
- Can be realized by symmetric encryption.
  - Only sender and receiver share the encryption.
  - Only the sender can successfully encrypt the message.
- For message without encryption: message authentication code (MAC) and hash function.
  - Broadcasting message.
  - Computer program.

Authentication

Message Authentication Code (MAC)



- Sender uses a secret key to generate a small block of data.
- Append the block to the message.
- Receiver use the same secret key to calculate a MAC on the received message.
- Compare the newly calculated MAC with the appended one.

If two MACs matched, then

- Message is not altered.
  - If message is altered, the MAC cannot match.
- Message is from trusted sender.
  - Other parties do not have the secret key and cannot generate a proper MAC code.

Authentication

Hash Function

- Hash function accepts a variable-size message M.
- Produce a fixed-size message digest $H(M)$.
- Do not need a secret key as MAC.
- Message digest is sent with message.

Visual Studio 2015 Update 3

Language	SHA-1 Hashes
Multilanguage	D59B21A64EDECAF6D127CE8FC0D0A6D40A6C3401

☑ **Quartus Prime Standard Edition**

☑ **Quartus Prime (includes Nios II EDS)**

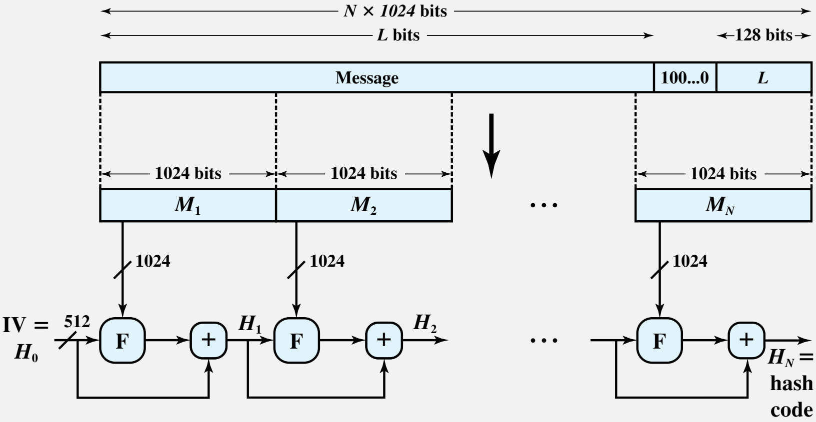
大小： 2.2 GB **MD5:** B0B6DE19632452B3D7757EE0E2A5C028

- The requirement of hash function H:
  - H can be applied to any size of data.
  - H produce a fixed-size output.
  - $H(x)$ is easy to compute, given x.
  - Given a hash code h, it is not feasible to find original x whose $H(x) = h$.
  - For $x \neq y$, it is not feasible that $H(x) = H(y)$.
- Popular Hash Function:
  - Message Digest: MD5
  - Secure Hash Algorithm: SHA-1, SHA-256, SHA-512

1. **Append padding bits**: Message is padded so that `length` mod 1024 = 896. Padding is always added. The number of padding bits is in the range of 1 to 1024. The padding consists of single one bit followed by necessary number of zeros.
2. **Append Length**: A block of 128 bits is appended to the message. It is an unsigned 128-bit integer containing the length of the original message.
3. **Initialize MD Buffer**: A 512-bit buffer is initialized to hold the intermediate and final result.
4. **Process in 512-bit Blocks**: Perform 80 rounds of processing. Same kind of operations, but vary in constants and logical functions.
5. **Output**: After all blocks are processed, the 512-bit result is generated.

Authentication

SHA-512 Generation Process



+ = word-by-word addition mod 2^{64}

- MD5: generate a 128-bit hash value.
- SHA-1: generate a 160-bit hash value.
- SHA-256: generate a 256-bit hash value.
- SHA-512: generate a 512-bit hash value.

Digital Signature

- Another way of using public-key encryption.
- A want to send a message to B:
  - A encrypt the message with B's public key.
  - B decrypt the ciphertext with his own private key.
- A want B to **be certain that the message is from him.**
  - A encrypt the message using his private key.
  - B finds he can decrypt the message only with B's public key.
  - B can confirm the message is from A.
- The entire encrypted message is the **digital signature.**
- Used to verify the data source and data integrity.

Digital Signature

- For efficiency, the sender usually signs one hash code of his original message.
- The receiver generates hash code with same algorithm and verify if they are matched.
- Do not need to consume more storage to store both plaintext and ciphertext.
- More efficient for data transmission.
- Summary:
  - In encryption, encrypt using public key, decrypt using private key.
  - In authentication, sign using private key, verify using public key.

Network Security using Python

- `pycrypto` module can be used to implement network security techniques.
- Installation (with Anaconda)

```
conda install pycrypto
```

```
>>> from Crypto.Cipher import DES
>>> des = DES.new('01234567', DES.MODE_ECB)
>>> text = 'abcdefgh'
>>> cipher = des.encrypt(text)
>>> cipher
b'\xec\xc2\x9e\xd9] a\xd0'
>>> decrypt_text = des.decrypt(cipher)
>>> decrypt_text
b'abcdefgh'
```

```
>>> from Crypto.PublicKey import RSA
>>> from Crypto import Random
>>> random_generator = Random.new().read
>>> key = RSA.generate(1024, random_generator)
>>> key
<_RSAobj @0x526ff28 n(1024),e,d,p,q,u,private>
>>> public_key = key.publickey()
```

Network Security using Python

Asymmetric Encryption

```
>>> enc_data = public_key.encrypt('abcdefgh', 32)
>>> enc_data
(b'\x95\x91Q\xb7h\x00\xac\x82\x92#\x8em
=\x8e\xe7\xdc\x19O\xdb\xbcq\xf9\x1b\xdf\xdb\xc5J1\xce|U\xd3
\x8d\x15gM\xb1\xd1OV\x9eD\x87\xd3\x9c\xc9\x9d\xf6{\xc8f\x10
\x01\xc2\xa1\xe9\xf9\xb8_\xde\x8a\xd0\xa6&r\xa8\xe0\xbb\xae
\x1f\xc7\x8c\xc7\xe3\xdf\x00\xae\x03j4\x91 \xdb\x99I\xe6\xe
c\xd0\x89\xde\x1c\xa0[\x96w:\x18\xc0\x17\x8b\xe3\xe73@$K|+\
xcb\xa0\xc7\xf3\x80"$\xec\xca\xa5b\x04w\x11\xda\xf3\x89!\xa
5',)
>>> decrypt_text = key.decrypt(enc_data)
b'abcdefgh'
```

Network Security using Python

Hash Function

```
>>> from Crypto.Hash import MD5
>>> from Crypto.Hash import SHA
>>> from Crypto.Hash import SHA256
>>> hash_md5 = MD5.new(b'CME451 Course').digest()
>>> print(hash_md5.hex())
41297691a2a72437702f9b1217227773
>>> hash_sha1 = SHA.new(b'CME451 Course').digest()
>>> print(hash_sha1.hex())
7e2d8f626f5cd2c55da0d3334859b79013f2d923
>>> hash_sha256 = SHA256.new(b'CME451 Course').hexdigest()
>>> print(hash_sha256)
dd779f6741eae2026ea05343d5ae006b12363682cd918fa55f4809533c
1484dd
```

```
>>> signature = privatekey.sign(hash_of_message, '')
>>> publickey.verify(hash_of_decrypted, signature)
```