CMSC389R

Cryptography II

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

homework ix recap

- Hash cracking
- Hash and submit challenge
- Questions?

agenda

- Some theory
  - Private key vs public key cryptography
- Heavy emphasis on applications
  - Focus on PGP
  - Message signing/verification
  - Attack on hash-based signatures

hashing

- Last week, we covered popular hashing algorithms
  - What they look like
  - How they can be cracked
  - Their weaknesses
- How does this tie into the course?

encryption

- Private key cryptography
  - Also known as symmetric cryptography
  - One key for encryption/decryption
  - Need to keep this *private*
- Public key cryptography
  - Also known as asymmetric cryptography
  - Public key -> encrypt, known to all
  - Private key -> decrypt, kept secret

symmetric key cryptography

- Meet your two new best friends:
  - Alice and Bob
- Alice wants to send Bob a message
  - Expectation: channel in which message is sent is actively wiretapped
  - Goal: send Bob a secret message without eavesdropper (Eve) recovering the message

symmetric key cryptography

- Alice generates a private key K and meets with Bob in person
  - Alice assures that Bob is *really* Bob
  - Alice give Bob K
- Alice and Bob part ways
  - Alice and Bob encrypt/decrypt messages with the *shared* key K.

symmetric key cryptography

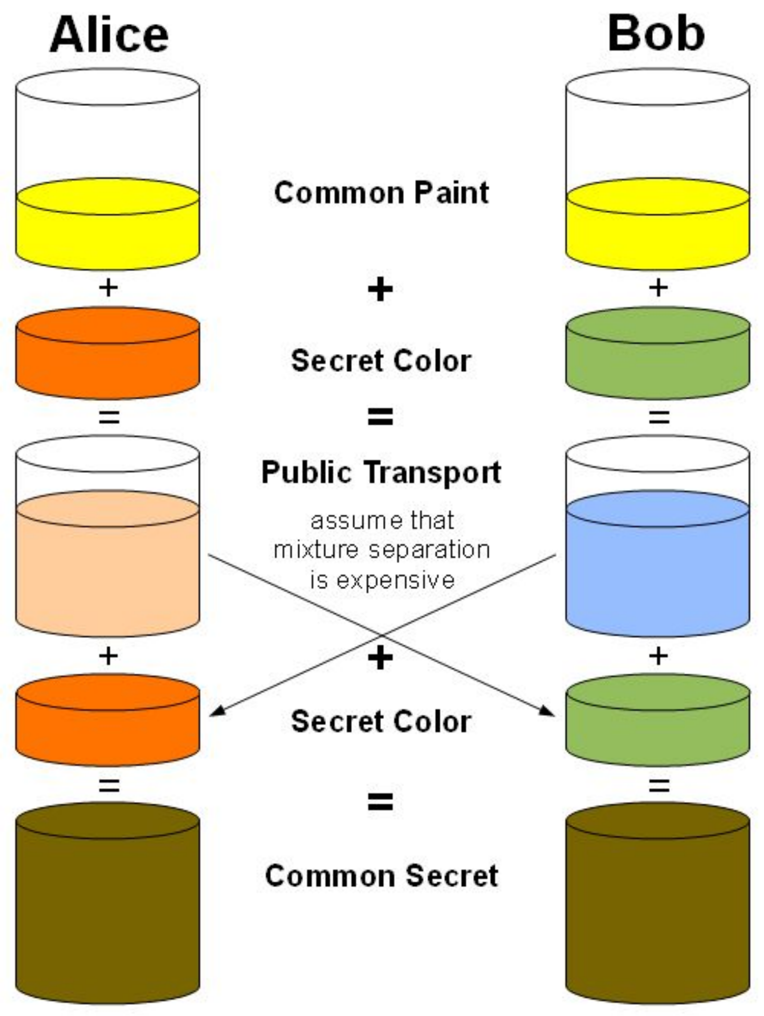
- Examples:
  - Vigenere Cipher
  - One-time pad (OTP)
  - Data Encryption Standard (DES)
  - Advanced Encryption Standard (AES)

symmetric key cryptography

- Advantages
  - Very secure with small key length
  - Relatively fast (vs asymmetric)
- Disadvantages
  - Difficult to share key
  - Both parties vulnerable if key is compromised

symmetric key cryptography

- Protocols exist for safe generation and sharing of symmetric keys
- e.g. Diffie-Hellman
  - Alice and Bob each pick their own secret
  - Each person transforms their secret based on a publicly agreed upon rule and exchange their transformations
  - Each person applies their secret to the other’s transformation to get the same key



- Diffie-Hellman
- Uses modular arithmetic
- Security provable based on mathematically hard problem
  - Discrete log

$$(g^{a} \bmod p)^{b} \bmod p = (g^{b} \bmod p)^{a} \bmod p$$

asymmetric key cryptography

- Alice and Bob each generate a public and private key pair
- Each exchange their public keys, keep private keys secret
- Alice applies Bob’s public key to her message and sends to Bob, encrypted
- Bob and ONLY Bob can apply his private key to Alice’s encryption to recover the message

asymmetric key cryptography

- Examples:
  - RSA (*HTTPS*, DRM)
  - PGP (Commonly used in email, ACH)
  - ElGamal (Discrete Logarithm problem)
  - Elliptic-Curve (shorter keys than RSA)

Pretty Good Privacy (PGP)

- Pretty Good Privacy (PGP): Developed in 1991 by Phil Zimmermann. Allows for
  - Encryption/Decryption
  - Signing
- Frequently used in email, files, full disk encryption, etc.

Pretty Good Privacy (PGP)

- Use gpg command line tool to generate public key/private key pair
- Can then share public key with the world
  - MIT PGP Key server
  - Email (ie. enigmail)
  - Keybase
  - …
- Decrypt messages using PGP private key

Pretty Good Privacy (PGP)

- gpg --gen-key
  - Generate key -- ID based on name/email
- gpg --list-secret-keys
- gpg --export --armor you@email.com > pubkey.asc
  - Create a key to send to friends
- gpg --import pubkey.asc
  - Import a friend's key
- gpg -e -u "Your name" -r "Their name" msg.txt
  - Generates msg.txt.gpg
- gpg --decrypt msg.txt.gpg
  - Display decrypted message

Pretty Good Privacy (PGP)

- Can sign documents too
  - Analogous to “encrypting” with private key
  - Anyone can then use your public key to verify the signiture
- gpg --output myfile.sig --sign myfile
  - “Encrypts” file, can’t see message without “decrypting”
- gpg --output myfile --decrypt myfile.sig
- gpg --clearsign myfile
  - Generates myfile.asc
  - Wraps file in signature rather than “encrypting” it
  - Useful for sending/posting publicly as it provides the message/signature in ASCII

Pretty Good Privacy (PGP)

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBFrHpXsBCADeJrGA5Rwaj4GvAwzGtKt6PFC
oaXj7uJTKl3h2IR2YTFSbyQV2...

-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----

02I1BJm5AQ0EWselewEIALahcUsgcJTyUb+yWka
+cN2Tsh3oItAAndhXUR0/zsEN...

-----END PGP PRIVATE KEY BLOCK-----

Merkle-Damgard Construction

- Recall from last week
- Can be used to make hash functions
    - Take a fixed-length block cipher or function
    - Split message into blocks
    - Pipe output of one block into input of next
    - Pad last block to be the right size
    - Final output = internal state of hash

Merkle-Damgard Construction

- Used by MD5 and SHA family of hashes
- Note: final output = final internal state
  - What if we initialize a hash with a given internal state?
  - Can add more message blocks to continue hashing from last state

Hash-based signatures

- Can use hashes to verify integrity of data
- Could use hashes to “sign” data
  - Signature: proof of authenticity by owner
  - hash(secret + data) = signature

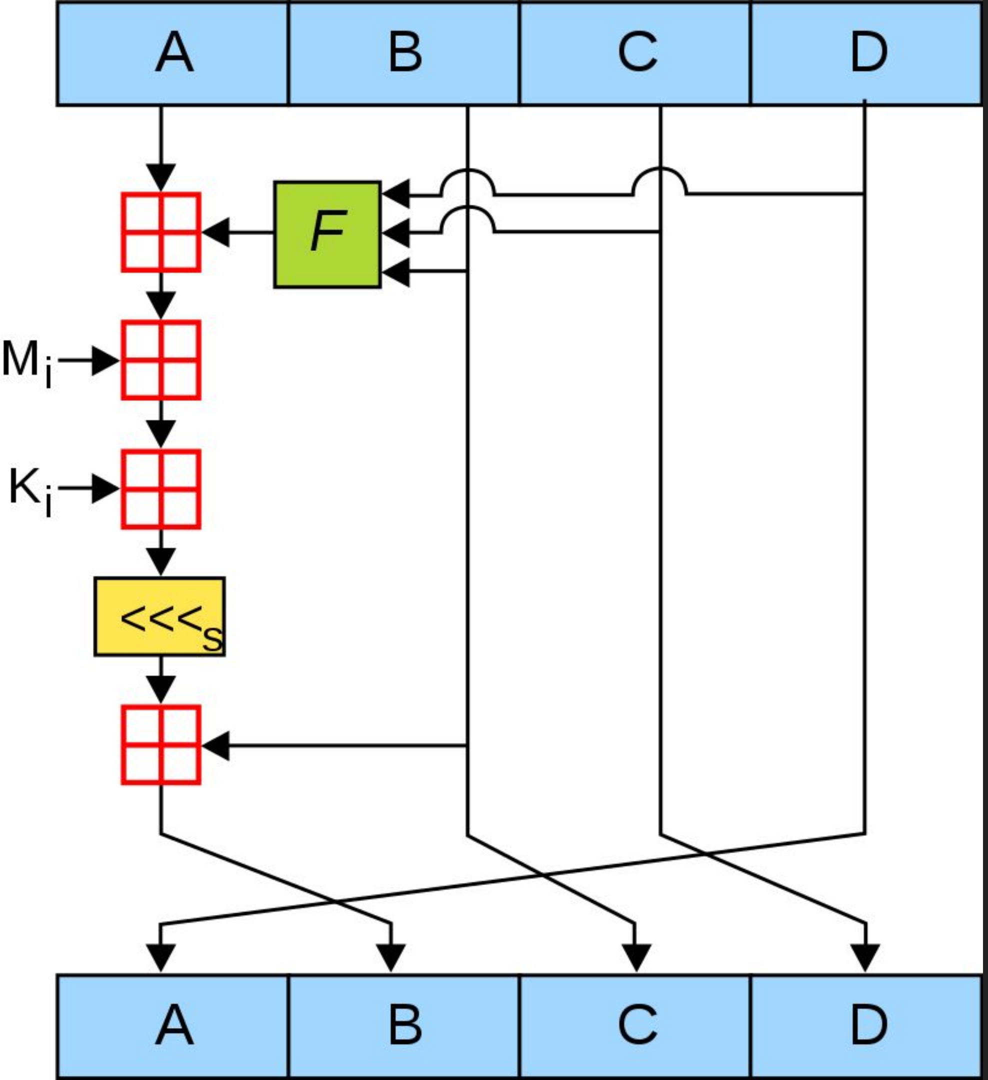
- Safe?

an attack on hash-based signatures

- hash(secret + data) signature is NOT safe for Merkle-Damgard-based hashes
  - Compute output = hash(secret + data)
  - Initialize another hash with state being the “output” of another hash
  - Update hash with payload
    - becomes hash(secret + data + payload)
  - Can forge signatures w/o secret!

MD5 state

- Example: MD5
- Output = final state
  - State variables A, B, C, D corresponding to 1/4 of final hash
  - e.g. 6057f13c496ecf7fd777ceb9e79ae285
  - A = 6057f13c, ..., D = e79ae285



MD5

- Output = A+B+C+D

MD5 padding

- MD5 is padded to a multiple of 512-bits
  - If message is a multiple of 512-bits, just make a new block containing only padding
- ‘1’ bit appended to message
- ‘0’ bits appended until block length is 64-bits less than a multiple of 512
- Last 64 bits filled with len(msg) in **bits**, modulo 2^{64}

MD5 padding

- Want to pad msg = “CMSC389R Rocks!”
  - len(msg) = 15 bytes = 120 bits
  - 512 - 64 = 448 bits of msg + pad
  - 448 - 120 = 328 bits of pad = 41 bytes
    - 1 ‘1’ bit, 327 ‘0’ bits
    - or one ‘0x80’ and 40 ‘0x00’
  - Message length 15 bytes => 120 bits = 0x78
    - Bit length stored in little endian
    - i.e. 78 00 00 00 00 00 00 00

MD5 padding

```
‘CMSC389R Rocks!\x80\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x78\x00\x00
\x00\x00\x00\x00\x00’
```

- use \x escape sequence in python for hex literals

attack on MD5 signatures

- Query w/ data: h = hash(secret + data)
- Initialize local state: A,B,C,D = h_1,h_2,h_3,h_4
  - Now have hash function $hash_{ABCD}()$
- Craft padding to match data
  - data + padding should be multiple of 512-bit
- Combine data + padding and arbitrary payload
- Perform $hash_{ABCD}$(payload) = h’
- h’ = hash(secret+data+padding+payload)
- Valid signature forged on data+padding+payload!

attack in python

- hashlib provides NO access to internal states of any algorithm, MD5 or SHA
- use separate module that opens access
    - We’ll be using one called md5py.py, supplied on Github

Homework #9

Will be posted soon.

Let us know if you have any questions!

This assignment has 2 parts.