# CMSC389R

## Cryptography II

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

# homework vii recap

- Hash cracking
- Hash and submit challenge
- Questions?

# announcements

- Bitcamp this weekend! Come see our talk
- UMDCTF next weekend (umdctf.io)


# agenda

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

# cryptography

- Science behind securing information
  - In our class, we'll focus on digital information security including:
    - Authentication
    - Data integrity
    - Message secrecy
    - Access control

# 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?

# 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

# 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)

```
-----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

- Method of building hash functions from one-way compression functions
  - f(x) = y, where len(x) > y -> compression
  - one-way -> hard to find inverse $f^{-1}(y) = x$
- Split message into blocks, pad the last block
- Use a seed, compute $f(seed + m_1) = g_1$
  - Then $f(g_1 + m_2) = g_2$, ... $f(g_n + m_n) = g_n$
  - $g_n$ is final hash output

# Merkle-Damgard Construction

- Used by MD5 and SHA family of hashes
- Note: final output = final internal state
  - Can add more message blocks to continue hashing from last state

# recall

- 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 state = hash(secret + data)
  - Update hash with state + 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 padding

- Input padded before fed into MD5
  - Format: message, padding, message length
  - ALWAYS need 8 bytes (64 bits) for message length
  - Padding = 64B - ((len(msg) + 8B) % 64)
  - Message padded by a ‘1’ bit followed by ‘0’s

# MD5 padding

- Want to pad msg = “CMSC389R Rocks!”
  - len(msg) = 15 bytes
  - 64 - ((15 + 8) % 64) = 64 - 23 = 41 bytes
  - 41 bytes = 328 bits, 1 “1” bit, 327 “0” bits
    - Or, 1 “0x80” byte and 40 “0x00” bytes
    - 0x80 = 0b10000000
  - Message length 15 bytes = 0x0f
    - Stored in little endian
    - i.e. 0f 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\x0f\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 = $h_1$,$h_2$,$h_3$,$h_4$
- Update state w/ payload to get:

  h’ = hash(secret + data + padding + payload)
- Form valid signature h’ on message

  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
- Demo?

# Homework #8

Will be posted tonight.

Let us know if you have any questions!

This assignment has 2 parts.

It is due by 4/12 at 11:59PM.