CSE 127 Week 10 Discussion

PA5: Cryptography

- Due: Sunday, March 13 at 6:00pm
- PA5: Ciphertext available on Gradescope for Part 1
- Five parts
  - Vigenère Cipher
  - MD5 Length Extension
  - MD5 collisions
  - RSA signature forgery
  - Writeup

Caesar Cipher

- shift letters of plaintext by fixed amount to get ciphertext

```
Plaintext   A T T A C K A T D A W N
Ciphertext  D W W D F N D W G D Z Q
```

```
A + 3 → D
T + 3 → W
C + 3 → F
…
```

Vigenère Cipher

- the combination of several Caesar Ciphers

Plaintext	A	T	T	A	C	K	A	T	D	A	W	N
Key	B	L	A	I	S	E	B	L	A	I	S	E
Ciphertext	B	E	T	I	U	O	B	E	D	I	O	R

Key ‘A’ means no shift
Key ‘B’ means shift by 1
Key ‘C’ means shift by 2
…

Tips

- Caesar Cipher is vulnerable to frequency analysis
- Vigenère Cipher is composed of `|Key|` Caesar Ciphers that can be defeated individually
- How can you figure out `|Key|` ?
- How do you know you got the correct key?
- User either member’s ciphertext is ok for group submissions

MD5 Length Extension

- Goal: generate an URL where the token is the valid MD5 hash of extended parameters
- For this part it is pymd5.py which has some functions to get at individual steps of md5 hashing
- Key idea: padding is 1 followed by necessary number of zeros at end of message, but you need to be able to have a 1 followed by zeros as part of the message as well
- 2.a in the assignment walks you through this and should make the attack understandable

Tips

- `python3 len_ext_attack.py "http://………NoOp"`
- Only use `urllib.parse.quote()` for the padding

MD5 Collisions

- Goal: two programs with different behavior that hash to the same thing
- We provide `fastcoll` which generates MD5 collisions
- You might need to build this code if its not available on your OS so there is also a makefile to help
- Key idea: once you have a collision, you can use your previous part to add identical suffixes to them and they will continue to collide
- think about how you can hide junk you are creating, will be useful later as well

Tips

- suffix should have a new line at the beginning
- Checkout piazza @610 if you run into compiling `fastcoll` on macOS

RSA Signature - Textbook

- Alice has public key `(N, e)` and private key `d` where `x^(de) = x mod N`
- To sign a message `m`, Alice computes `s = m^d` and Bob can verify by checking that `s^e = m mod N`
- Eve can trivially generate a signed message `(m=s^e, s)`, where `s^e` is the message and `s` the signature
- Bob verifies the signature by checking by `s^e=m`

RSA Signature

To combat the previous problem, structure is added to the message

A k-bit RSA key used to sign a Sha-1 hash digest will generate the following padded value of m:

```
00 01 FF...FF 00 3021300906052B0E03021A05000414 XX...XX
      ^^^^^^^    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^
  k/8 - 38 bytes wide         ||             20-byte SHA-1 digest
                   ASN.1 "magic" bytes
```

```
Sig = padding(SHA1(m))^d mod N
Verify =( strip_padding(Sig^e mod N) == SHA1(m) )
```

RSA Signature Forgery

- So now Eve can't compute just any `s^e` because it needs to match the format
- Note that number of `FF` bytes is determined in specification
- **What happens if this is not checked? (i.e. implementation just discards `FF` bytes until reaches a `00` byte)**
- Instead of generating a signature `s` such that `s^e` is of the form on the previous slide, it only needs to match on a certain number of high order bytes with any number of `FF` padding bytes
- Problem compounded if `e=3`, because can then work in integers (compare `e=65537`)

Tips

- If got stuck finding a valid root, think about how many higher bytes in the signature the verification process should recover?
- Don't use `openssl` to test your solution. Write your own validation code that doesn't check the length of `FF` s

Writeup

- 7 questions, 4 from part 3a and 3 from part 5 in assignment
- Answers should be concise and complete
- Write a comment if you used your code from previous classes (e.g. CSE 107)