Legal Notice

This lecture will be recorded and made available to registered students on Canvas.
Programming Assignment 6

Logistics:

- Ciphertext for first part available on Gradescope
- Early submission is Tuesday, March 2
- Regular/final submission is Friday, March 5
Outline

5 parts, see bottom of assignment for submission checklist

1. Vigenère cipher
2. MD5 Length Extension
3. MD5 collisions
4. RSA signature forgery
5. Writeup
Caesar Cipher

reportedly used by Julius Caesar, so circa 44 BC

idea is to shift letters of plaintext by fixed amount to get ciphertext

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>A</th>
<th>T</th>
<th>D</th>
<th>A</th>
<th>W</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciphertext</td>
<td>D</td>
<td>W</td>
<td>W</td>
<td>D</td>
<td>F</td>
<td>N</td>
<td>D</td>
<td>W</td>
<td>G</td>
<td>D</td>
<td>Z</td>
<td>Q</td>
</tr>
</tbody>
</table>
Vigenère Cipher

Historical cipher (circa 1550 AD), evolution of substitution ciphers sitting between the Caesar cipher and more recent rotor machines like the Enigma (WWII)

Ciphertext available on gradescope

Description of some different attacks in linked wikipedia article

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>K</th>
<th>A</th>
<th>T</th>
<th>D</th>
<th>A</th>
<th>W</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>B</td>
<td>L</td>
<td>A</td>
<td>I</td>
<td>S</td>
<td>E</td>
<td>B</td>
<td>L</td>
<td>A</td>
<td>I</td>
<td>S</td>
<td>E</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>B</td>
<td>E</td>
<td>T</td>
<td>I</td>
<td>U</td>
<td>O</td>
<td>B</td>
<td>E</td>
<td>D</td>
<td>I</td>
<td>O</td>
<td>R</td>
</tr>
</tbody>
</table>
MD5 Length Extension

First of two parts with md5. There is code to help with both parts.

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

2a in the assignment walks you through this and should make the attack understandable
MD5 Collisions

We provide code `fastcoll` which generates md5 collisions.

You might need to build this code if it's 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.

Also, think about how you can hide junk you are creating, will be useful later as well.

Goal: two programs with different behavior that hash to the same thing.
RSA Signature - Textbook

Alice has public key (N, e) and private key d where $x^{(de)} = x$ (modulo N)

To sign a message $m$, Alice computes $s = m^d$ and with $m$ and $s$ Bob can verify by checking that $s^e = m$

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
```

\[
\text{\underline{k/8 - 38 bytes wide}} \quad || \quad \text{\underline{20-byte SHA-1 digest}}
\]

\[
\text{ASN.1 "magic" bytes}
\]

Where the SHA-1 digest is the hash of the original message m and the ‘magic bytes’ are an encoding of the type of hash and the size of the hashed value

Alice now signs this padded value, m’, by computing \( s = m'^d \) and Bob verifies it by checking \( s^e = m' \) and SHA-1(m) matches the digest in 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 \( e=65537 \)).
Writeup

7 questions, 4 from part 3a and 3 from part 5 in assignment

Autograder will only check that you submitted properly named file

Answers should be concise and complete
Questions

That is the it for the the final programming assignment

And for the recording