PLTW COMPUTER SCIENCE

Activity 2.1.5

Secure Protocols

Introduction

People send data through the Internet for friendship, business, research, politics, and more. These realms of culture are thousands of years old. Betrayal is every bit as old. However, imposters, spies, and thieves now have global reach like never before. Remember, the Internet is new, with pervasive use beginning in the 1990s. Today all data are vulnerable.

Consider an exchange between Allie and Baba. Allie sends Baba a message, and gets a response from Baba – or so she thinks.

How can Allie know that the reply is from Baba and not some lurker in the Internet?

How can she be sure her message even got to Baba?

Packets on the Internet hop from host to host, taking multiple routes past potentially hostile parties. How can Allie and Baba exchange private information like credit card numbers and be certain that no one eavesdropped on their communication?

Materials

- Computer with Canopy

Resources

2.1.5 sourceFiles.zip

Procedure

Part I: Cryptography and Ciphers

1. Form pairs as directed by your teacher. Read the passages below and discuss the questions together.
2. As you learned in a previous activity, the Internet has scaled well because it is decentralized. Packets pass through multiple routes controlled by potentially unfriendly parties. Yet we depend on privacy for many uses of the Internet.

   Cryptography is the science and art of delivering a message securely and confidentially. The discipline is thousands of years old. An encryption algorithm is used to turn plaintext into ciphertext so that (hopefully) only the recipient can read it. Decryption restores the plaintext.

   The "s" in the "https" of a URL's scheme indicates that information exchanged in a Web request and response will be encrypted. Give an example of information you might exchange over the web that you would want to be encrypted.

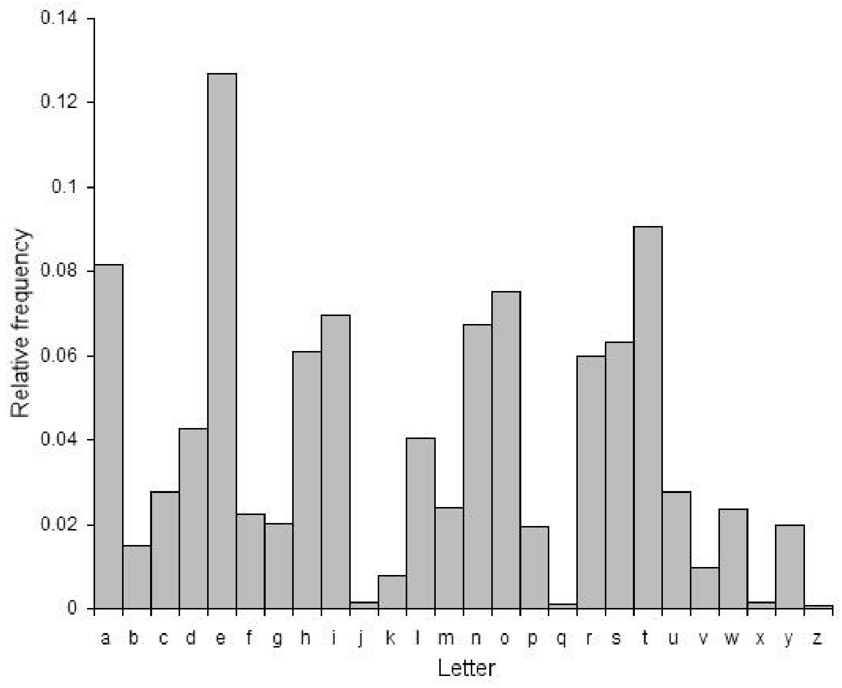
3. Julius Caesar, who took dictatorial control of the Roman Republic in 44 BC, used an encryption algorithm now known as a substitution cipher. In a substitution cipher, each letter of the plain text is replaced another letter. Caesar's algorithm shifted each letter by three within the alphabet so that plain text "A" becomes ciphertext "D," and so on, as shown in the table below. The table is an example of a **key**, which is data used by an encryption or decryption function.

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

   The following message has been encrypted with Caesar's cipher. Decrypt it by shifting each letter three in the opposite direction.
   "*Wub wklv*!"

4. Simple ciphers are easily decrypted, even if the key is not known. If the ciphertext is long, the key can be deduced by using letter frequencies; the relative frequencies of letters in English are shown below. The relative frequency tells what percent of the time a particular letter is used.

   Based on the histogram, which letter is most common if the plaintext is English?



5. Another reason that simple ciphers are easily decrypted is that an individual can try all possibilities—to stumble upon it with certainty by brute force. Brute force is trial and error—with computational muscle. Here you will apply brute force to decipher the following ciphertext. The ciphertext was created with a Caesar-like cipher: a rotation of the alphabet, other than 3 letters. Instructions are below.

   Jvgure jrag gerzraqbhf qvabfnhef! Guvaxvat gurer fubhyq rkvfg nggrzcgrq uvynevgl urer?

   - Open the file `ciphers.py` in the code editor in Canopy. Your work in this assignment will mostly be in the iPython session. Turn on iPython session logging by setting the working directory and typing `%logstart -or` *`filename`* at the iPython prompt.
   - Execute the file. This will define the function `rotate()`. An excerpt showing the `def` statement and the docstring is shown here.

```
def rotate(string, n):
    '''Returns the ciphertext of string after it has been shiftec
    Works on both upper and lower case letters. Other characters
    '''
```

   - Use *Python*® programming language to decrypt the ciphertext *"Jvgure ..."* shown above. Use iterations of `rotate()` across all possible keys until the plaintext is English. This could be done in two ways: complete either Part i or Part ii as follows. Either way, you will need to copy the ciphertext and paste it into Canopy.

     - From the iPython command line, call `rotate()` repeatedly.

```
In []: rotate("Jvgure jrag...", 1)
Out[]: 'Kwhvsf ksbh...'
```

- In the code editor, complete `try_all_25()` by removing `pass` and writing your own algorithm for the `for` loop.

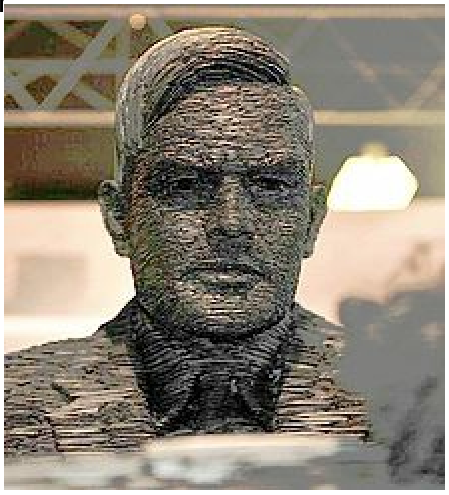
```
def try_all_25(string):
    ''' Use brute force to crack Caesar-like cipher,
    printing all 25 possible shifts
     '''
    for shift in range(1,26):
        pass  # replace with your algorithm
```

- Explain why you chose Part i or Part ii above.

6. Ciphers are an example of private key encryption, where the sender and the recipient have to agree beforehand on the key and then both manage to keep it secret. If either person reveals the key, the cryptographic code is broken. Ciphers are also described as symmetric key encryption. The same key is used by the encryption algorithm and the decryption algorithm, albeit in reverse.

   One of the most famous ciphers was implemented by the Enigma machine. Enigma was used by Germany and Italy during World War II, in which they formed the European front of the Axis powers (Germany, Italy, and Japan) against the Allied powers (the United States, the United Kingdom, and the Soviet Union). Germany transmitted submarine routes encrypted by Enigma.

   Alan Turing (pictured at right in statue) automated a method for cracking Enigma, allowing the British to sink many German submarines, making a major contribution to the Allied victory. Computer science's most prestigious award is named in Turing's honor. The Turing Prize is awarded annually.



   Roughly half of the work on cryptography is conducted within government agencies, with the NSA leading the effort in the United States. Why has cryptography always been crucial to nations and empires?

Part II: Public Key Encryption

7. Encrypted Internet traffic often establishes a secure connection using an RSA algorithm, discussed in a later activity. RSA is an example of public key encryption, which uses two paired keys that belong to one owner. The owner of the keys shares one of them publicly and keeps the other secret. The keys are large numbers used by a function. Using either key with the function will transform plaintext into ciphertext. Using the other key will undo the

encryption.

This encryption can be used for two different purposes.

- Encrypt a message with someone else’s public key. Only they will be able to read the message.
- Encrypt a message with your own private key. Anyone with the message will know the message came from you.

Suppose you get a message from Target, Inc. It looks like gibberish. Target's public key turns it into sensible English. Explain why that should increase your confidence that Target actually sent you that message.

8. You send Target a response that contains your credit card number. You use Target's public key. Explain why that should increase your confidence that only certain Target employees will be able to read your credit card number.
9. To have hands-on practice with these two uses of RSA encryption, generate a pair of RSA keys.
   - Open `paired_keys.py` in the Canopy code editor.
   - Execute the code to define the functions `make_keys()` and `use_key()`.
   - In the iPython session, make two keys `public_key` and `private_key` as shown below.

```
In []: public_key, private_key = make_keys()
In []: print public_key
Out[]: (23707, 3343)
In []: print private_key
Out[]: (23707, 7)
```

   - Each of the keys contains two numbers. The modulus is the number that is the same in the two keys. In the example output shown above, 23707 is the modulus. Repeat Step c until your key's modulus is at least 10000.
   - Publish your public key for the class to use. You can use a class-wide **Google document** to publish your keys. Alternatively, write your public key and your name on a notecard and place your notecard on a bulletin board for people to use.

10. Follow the steps below to send a message that only the recipient will be able to decrypt. Include your name and at least one other word in the message.

   - Turn your secret message into a numeric message. As shown below, use the `numerize()` function that was defined when you executed `paired_keys.py`. This is a substitution cipher that replaces each character with a 2-digit number. `numerize()` returns the message as ciphertext. The ciphertext is a sequence of numbers smaller than the RSA key's modulus.

```
In []: message = numerize("Hey Jon, see you after school? -Ted")
In []: message
Out[]: '0646-7595-0648-8584-1806-8975-7506-9585-9106-7176-9075-88
```

- Use the recipient's public key as shown below to encrypt the numeric message. Use the `use_key()` function that was defined when you executed `paired_keys.py`. The example below uses Jon's public key.

```
In []: jons_key = (20453, 61) # Use recipient's public key
In []: encrypted_message = use_key(jons_key, message)
In []: encrypted_message
Out[]: '14139-16645-3362-9259-18839-14535-16569-5603-13993-3556-25
```

- Pass the encrypted message to its recipient. Use email, Skype, a class-wide Google document, pencil and paper, or any other method as directed by your teacher.
- Assign the message that you have received to `encrypted_message`, as shown below. Decrypt the message using your private key. In the example below, Jon (the recipient) has typed the message he received at the iPython prompt to assign `encrypted_message` and used his private key.

```
In []: encrypted_message = '14139-16645-3362-9259-18839-14535-1656
In []: private_key # Jon's private key
Out[]: (20453, 661)
In []: decrypted_message = use_key(private_key, encrypted_message)
In []: decrypted_message
Out[]: '0646-7595-0648-8584-1806-8975-7506-9585-9106-7176-9075-880
In []: plain_message = denumerize(decrypted_message)
In []: plain_message
Out[]: 'Hey Jon, see you after school? -Ted'
```

11. Follow the steps below to publicize a message that only you could have sent. Other people will use your public key to decrypt your message. That will authenticate that the message came from you and not from someone else.
    - Enter your message at the iPython prompt. Use the substitution cipher `numerize()` to make it a numeric message. Encrypt your message using your private key.

```
In []:  message = "I make this payment -JSmith."
In []:  numerize(message)
Out[]: '4706-7183-0690-7875-0685-8475-687989-0689-7879-8890'
In []: use_key(private_key, _ )
Out[]: '15790-19762-15967-10221-16029-8252-16029-28499-25391-1491
```

Note the call to `use_key(private_key, _)` and the second parameter, the lone underscore character. Used in this way, underscore character _ a special variable in the iPython session. It contains the value returned by the previous command.

- Publish your message. The whole class should publish everyone's messages in a single Google document. A bulletin board is an alternative.
- Select one or more of the published statements as directed by your teacher. Enter that

person's published message at the iPython prompt. Decrypt it with their public key, as shown below.

```
In []: authenticated_message = '15790-19762-15967-10221-16029-82
       52-16029-28499-25391-14919'
In []: jons_public_key = (20453, 61)
In []: use_key(jons_public_key, authenticated_message)
Out[]: '0646-7595-0648-8584-1806-8975-7506-9585-9106-7176-9075-88
In []: plain_message = denumerize(_)
Out[]: 'I make this payment -JSmith'
```

Record the message here.

- Explain why using the sender's public key to read the message increased your confidence that the message was really from them.
- Why doesn't the use of the sender's public key guarantee that the message was from them?

Part III: Authentication and Trusted Authorities

12. A public key is an important part of an SSL certificate. SSL certificates are encrypted messages that confirm the key owner's identity and provide their public key. In the previous example, you assumed that the public key came from the person who said it was theirs. They could have been an imposter, of course. When you access a URL that begins with `https`, your browser downloads an SSL certificate from that website. Your browser will then use the public key contained in the certificate to send your confidential information. But what are the implications for you if an imposter makes a pair of keys and gives you one, tricking you into thinking it is the website's public key?
13. A Web browser authenticates that a public key really comes from the owners of the domain name that you are accessing and not from an imposter. When a company gets an SSL certificate, they apply to a Certificate Authority (like Symantec) to sign their certificate. The certificate authority uses their own private key to encrypt the applicant's public key and the applicant's domain name. The certificate authority's public key is built into the browser; if that public key can decrypt the certificate, it must be from the certificate authority. The decrypted certificate contains the domain name and public key of the owner of the certificate. If it matches the domain name you are trying to reach, you know they are who they say they are.

    To simulate the way the SSL relies on a handful of trusted authorities, use Python and trust authentication to get a message from an untrusted sender, as follows.

    - Select a small number of people in your class to trust, as directed by your teacher. Who are the trusted authorities in your class?
    - As a simulation, apply for an SSL certificate from one of the trusted authorities. Do this by giving them your public key and your name. This can be done with a Google document.
    - The trusted authority will confirm with you that the public key is in fact yours.
    - The trusted authority will encrypt a message containing your public key and your name, using their private key. An example is shown below.

```
In []: applicant = '(20453, 61) Jon Smith' # Applicant's key and r
In []: numerize(applicant)
Out[]: '0646-7595-0648-8584-1806-8975-7506-9585-9106-7176-9075-880
In []: use_key(authoritys_private_key, _ ) # Returns the certifica
Out[]: '10252-15596-1552-3433-2078-16302-16914-26648-10542-14912'
```

- The trusted authority will return the result to you, the applicant. This is your SSL certificate. It has been signed by the trusted authority. You can create a new column in the same Google document for each student's certificate.
- Select an "untrusted" partner as directed by your teacher. Obtain your partner's public key, as follows.
  - Obtain your untrusted partner's certificate.
  - Use the trusted authority's public key to obtain your untrusted partner's public key, as shown below.

```
In []: certificate = '10252-15596-1552-3433-2078-16302-16914-2664
In []: CAs_public_key = (44719, 11) # from certificate authority
In []: use_key(CAs_public_key, certificate)
Out[]: '(20453, 61) Jon Smith'
```

- Exchange messages with the untrusted partner using each other's public keys, as directed in Step 10 above. Record your untrusted partner's plaintext message here.
- Compare the use of public keys in Part d of Step 10 with the use of signed public keys in Part g of this step. Explain why the involvement of the Certificate Authority in this step should increase your confidence that your message in this step was only readable to its intended recipient.

14. Save your iPython session log by typing `%logstop` at the iPython prompt.

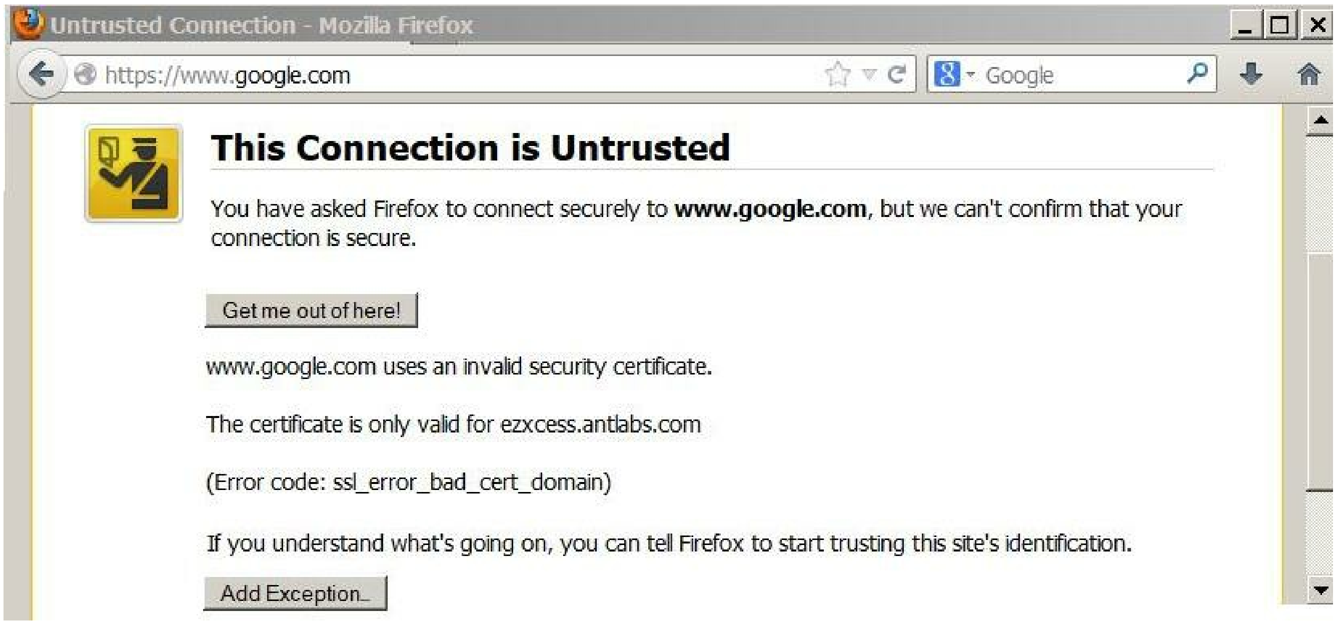
Conclusion

1. Explain the relationship among the following concepts.

   - The `https://` designation in your browser
   - Public key encryption
   - SSL certificate
   - Certificate Authority
   - Domain name

2. You typed `https://www.google.com` in the address bar in your browser and received the notice shown below. Choose "Get me out of here!" or "Add Exception."

   Explain why you made the choice you did.


Untrusted Connection - Mozilla Firefox

https://www.google.com

This Connection is Untrusted

You have asked Firefox to connect securely to **www.google.com**, but we can't confirm that your connection is secure.

Get me out of here!

www.google.com uses an invalid security certificate.

The certificate is only valid for ezxcess.antlabs.com

(Error code: ssl_error_bad_cert_domain)

If you understand what's going on, you can tell Firefox to start trusting this site's identification.

Add Exception...