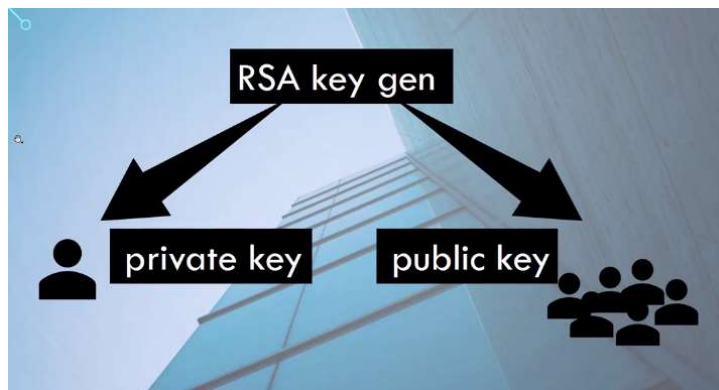


Vid 8: Asymmetric Encryption

LESSON 1: DIGITAL SIGNATURES

- What's a digital signature?
Easy to verify. Hard to mimic
- Asymmetric encryption
RSA

Public and Private Keys: Useful for when someone wants to send a message to you → they can use your public key to send a message to you that can only be read by you.

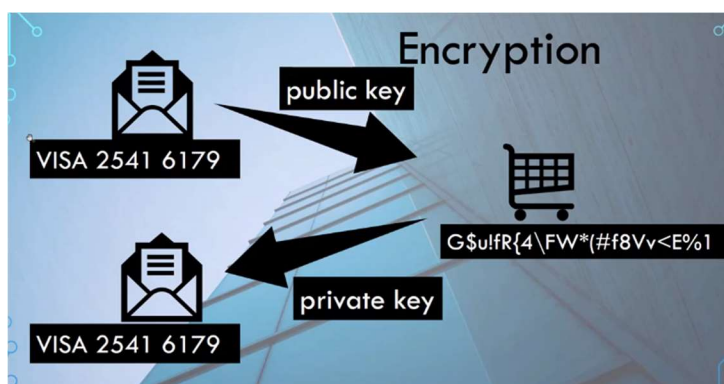


Example: When your trying to buy something from an online vendor, and you want to send them your credit card number. Internet is not a secure venue.

How we do this securely? → the vendor sends you their **public key** and you use that public key (using the math in asymmetric encryption) to encrypt a message (into random bits). Then those are sent over the internet to the vendor who then can decrypt those random bits using their **private key** and turn them back into your credit card number.

It can be intercepted in the middle of being sent, but without the private key the message cannot be decrypted.

Nothing that was sent over the internet is a secret at all (only the private key and your credit card # are a secret)



Vid 9: Digital Signatures

Works in reverse of asymmetric encryption

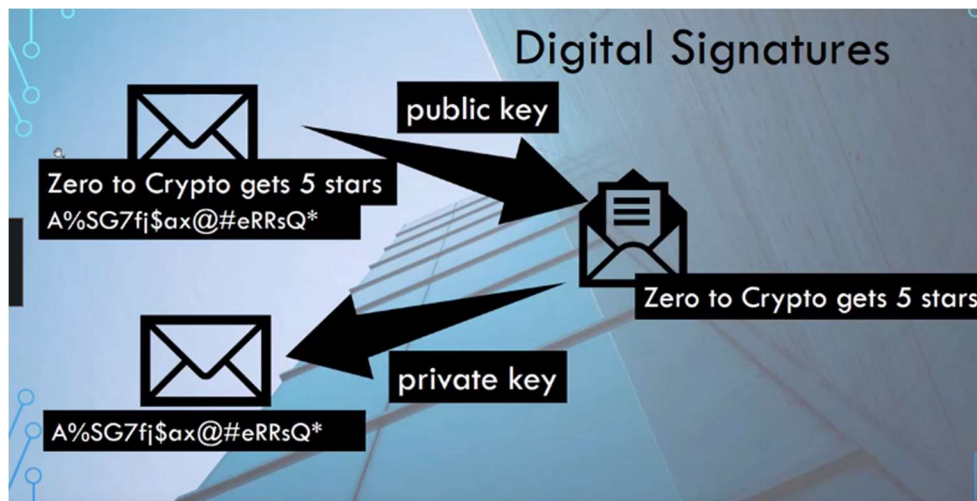
Say you, the owner of a private key wants to give zero to crypto a rating. You use your private key in this case to perform the same algorithm that you would use if someone had sent you an encrypted message.

So, you take this message, which is open text, and you sign it using your **private key**. It is the same algorithm that a vendor would use to decrypt your credit card number.

Now you're taking an open text message and decrypting it as if it were an encrypted message. You get a lot of garbled nonsense, looks very random to anyone who doesn't have the public key.

To prove that it was actually you that loved my course and not me just generating a lot of fake reviews: you can send to anyone who asks that statement: "Zero to Crypto gets five stars" and your signature. And they can use your public key to encrypt that message.

They perform the same algorithm that they would if they were sending you an encrypted message. They're going to encrypt it using your public key and from your signature they'll get out the original message: "Zero to Crypto gets five stars."

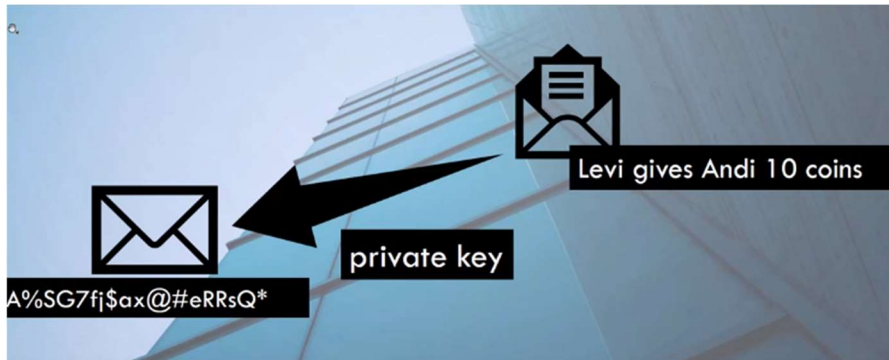


Instead of decrypting and encrypting the entire message, we perform it on a **hash** of the message. So we send our message or document through a hash function which makes it shorter. **(takes a long message and creates a fingerprint of that message in a predictable way)**

Vid 10: Signatures for Cryptocurrencies

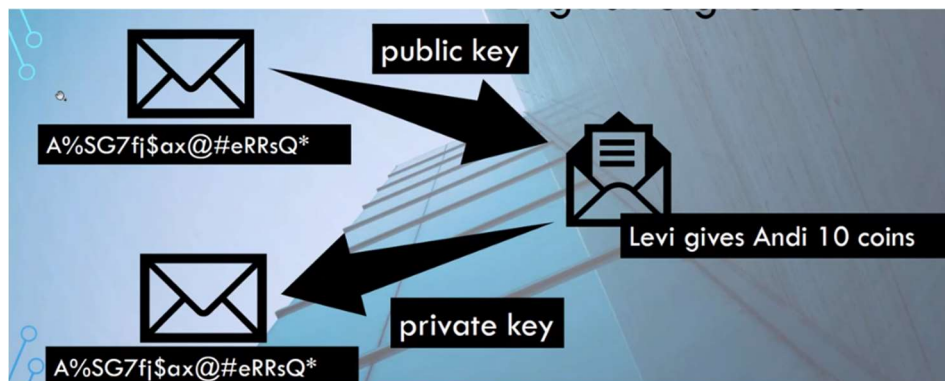
The connection to crypto is that the thing we'll be signing are entries in a public ledger.

- ➔ When you want to send coins to someone ill take the statement: "Levi gives Andi ten coins" and use my private key too generate a signature for that message
- ➔ Then sign it and get these random looking bits



Then Andi is going to want to spend the coins, so in order to spend them, whoever shes giving them too needs to know that she actually owns them.

This signature on this statement "Levi gives andi 10 coins" is the verification that she owns those coins.



As soon as you create digital signatures, you have statements that you can sign that are verifiable easily and can be posted to a public ledger

Vid 12: Installing Modules with pip

The screenshot shows a Udemy course page for 'Installing Modules with pip'. The course content list on the right includes:

- 12. Installing modules with pip (2min)
- 13. Assignment: Sign and verify (5min)
- 14. Solution: Part 1 (9min)
- 15. Solution: Part 2 (4min)
- 16. Strings or Bytes (5min)

The main content area displays a list of pip options and their descriptions:

- retries <retries>: Maximum number of retries each connection should attempt (default 5 times).
- timeout <sec>: Set the socket timeout (default 15 seconds).
- exists-action <action>: Default action when a path already exists: (s)witch, (i)gnore, (u)se, (b)ackup, (a)bort.
- trusted-host <hostname>: Mark this host as trusted, even though it does not have valid or any HTTPS.
- cert <path>: Path to alternate CA bundle.
- client-cert <path>: Path to SSL client certificate, a single file containing the private key and the certificate in PEM format.
- cache-dir <dir>: Store the cache data in <dir>.
- no-cache-dir: Disable the cache.
- disable-pip-version-check: Don't periodically check PyPI to determine whether a new version of pip is available for download. Implied with --no-index.

The terminal output shows the user navigating to the Downloads folder and running the command to install cryptography using get-pip.py:

```
C:\Users\Fixixgeek>cd Downloads
C:\Users\Fixixgeek\Downloads>python get-pip.py
Requirement already up-to-date: pip in c:\users\Fixixgeek\AppData\Local\Programs\Python\Python36\lib\site-packages
C:\Users\Fixixgeek\Downloads>pip install cryptography
```

```
Command Prompt
C:\Users\Russ>Python36
'Python36' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Russ>cd Python36

C:\Users\Russ\Python36>python get-pip.py
python: can't open file 'get-pip.py': [Errno 2] No such file or directory

C:\Users\Russ\Python36>python get-pip.py
Collecting pip
  Downloading https://files.pythonhosted.org/packages/a4/6d/6463d49a933f547439d6b5b98b46af8742cc03ae83543e4d7688c2420f8b/pip-21.3.1-py3-none-any.whl (1.7MB)
    100% |#####| 1.7MB 11.5MB/s
Collecting wheel
  Downloading https://files.pythonhosted.org/packages/27/d6/003e593296a85fd6ed616ed962795b2f87709c3ee2bca4f6d0fe55c6d00/wheel-0.37.1-py2.py3-none-any.whl
Installing collected packages: pip, wheel
Found existing installation: pip 9.0.1
Uninstalling pip-9.0.1:
  Successfully uninstalled pip-9.0.1
The script wheel.exe is installed in 'C:\Users\Russ\Python36\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-21.3.1 wheel-0.37.1

C:\Users\Russ\Python36>cd Scripts

C:\Users\Russ\Python36\Scripts>python get-pip.py
Python was not found; run without arguments to install from the Microsoft Store, or disable this shortcut from Settings > Manage App Execution Aliases.

C:\Users\Russ\Python36\Scripts>pip install cryptography
Collecting cryptography
  Downloading cryptography-36.0.2-cp36-abi3-win_amd64.whl (2.2 MB)
    100% |#####| 2.2 MB 2.2 MB/s
Collecting cffi>=1.12
  Downloading cffi-1.15.0-cp36-cp36m-win_amd64.whl (178 kB)
    100% |#####| 178 kB ...
Collecting pycparser
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)
    100% |#####| 118 kB ...
Installing collected packages: pycparser, cffi, cryptography
Successfully installed cffi-1.15.0 cryptography-36.0.2 pycparser-2.21

C:\Users\Russ\Python36\Scripts>
```

Vid 13: Sign and Verify

Vid 14: Solution Part 1

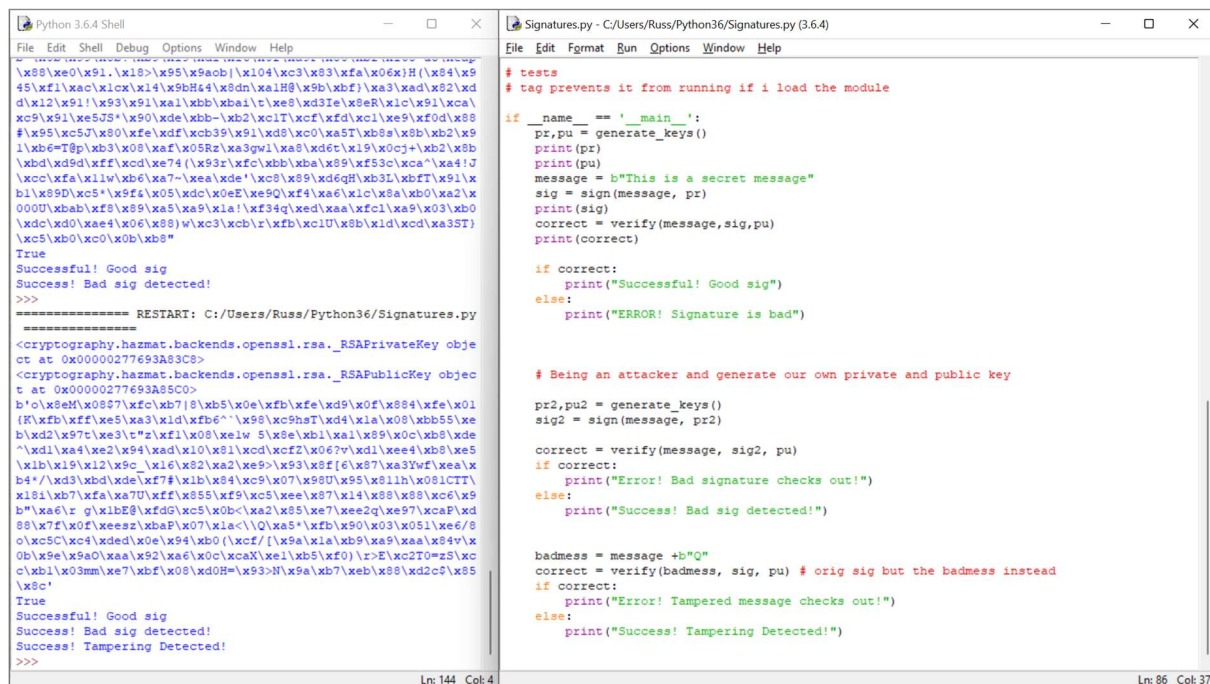
With this solution we can

- ✓ Detect a good signature

Vid 15: Solution Part 2

With this solution we can:

- ✓ Detect a good signature
- ✓ Flag a bad signature
- ✓ Flag a bad message



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
>>> from signatures import generate_keys, sign, verify
>>> pr, pu = generate_keys()
>>> print(pr)
>>> print(pu)
>>> message = b"This is a secret message"
>>> sig = sign(message, pr)
>>> print(sig)
>>> correct = verify(message, sig, pu)
>>> print(correct)
True
Successful! Good sig
Success! Bad sig detected!
>>>
===== RESTART: C:/Users/Russ/Python36/Signatures.py
=====
>>> from cryptography.hazmat.backends.openssl.rsa import RSAPrivateKey, RSAPublicKey
>>> ct = RSAPrivateKey.from_private_bytes(b'-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA...
-----END RSA PRIVATE KEY-----')
>>> pt = RSAPublicKey.from_public_bytes(b'-----BEGIN RSA PUBLIC KEY-----
MIIEPjBBBgkqhkiG9w0BBQwwDAQg...
-----END RSA PUBLIC KEY-----')
>>> b'0x8eM\x0877\xfc0xb718\x8b5\x0e\xfb\xfe\x09\x0f\x884\xfe\x01
(R\xfb\xff\x05\x03\x1d\xfb6^\x98\x09hsT\x04\x1a\x08\xbb55\x0e
b\x02\x97t\x03\tz\x01\x08\x0e1w 5\x0e\x01\x01\x08\x0c\x0b\x0e
^\x01\x04\x02\x04\x0d\x10\x01\x0d\x02\x067v\x01\x0e4\x0b\x0e5
\x1b\x19\x12\x9c_\x16\x02\x02\x0e9>\x93\x8f[6\x07\x03Ywf\x0e\x
b4*/\x03\x0b\x0e\x07\x01\x04\x09\x07\x080U\x095\x011h\x081CTT\x
x181\x07\x0fa\x070\xff\x055\x0f9\x05\x0e\x07\x14\x08\x08\x0c\x09
b^\x06\t gh\x1bE9\x0d0\x05\x0b<\x02\x057\x0e2q\x0e9\x0aF\x0d
89\x07f\x0d0\x0e2\x0b0\x07\x1a<\x01\x055^\x0fb\x090\x03\x051\x0e/8
o\x05c\x04\x0e4\x0e4\x0b0(\x0c/\x09\x1a\x0b9\x09\x0a\x049\x0
0b\x09\x09a0\x02\x06\x0c\x0c\x01\x05\x0f0)\r>E\x02T0=z5\x0c
c\x0b1\x03mm\x0e7\x0b\x08\x0d0H=\x93>N\x09a\x0b7\x0eb\x08\x0d2c5\x05
\x0c'
>>> True
Successful! Good sig
Success! Bad sig detected!
Success! Tampering Detected!
>>>

Signatures.py - C:/Users/Russ/Python36/Signatures.py (3.6.4)
File Edit Format Run Options Window Help
# tests
# tag prevents it from running if i load the module

if __name__ == '__main__':
    pr, pu = generate_keys()
    print(pr)
    print(pu)
    message = b"This is a secret message"
    sig = sign(message, pr)
    print(sig)
    correct = verify(message, sig, pu)
    print(correct)

    if correct:
        print("Successful! Good sig")
    else:
        print("ERROR! Signature is bad")

# Being an attacker and generate our own private and public key

pr2, pu2 = generate_keys()
sig2 = sign(message, pr2)

correct = verify(message, sig2, pu)
if correct:
    print("Error! Bad signature checks out!")
else:
    print("Success! Bad sig detected!")

badmess = message + b"Q"
correct = verify(badmess, sig, pu) # orig sig but the badmess instead
if correct:
    print("Error! Tampered message checks out!")
else:
    print("Success! Tampering Detected!")
```

Vid 16: Strings or Bytes

Encryption functions don't work with str types the need to be in bytes

```
>>> type(b"message")
<class 'bytes'>
>>> type("Messages")
<class 'str'>
```

Modified the program so that you can either pass strings or bytes

