# Asymmetric Encryption



# Applied Cryptography

# The Four Primitives

- There are four primitives which are considered the building blocks of digital cryptography
  - Random Number Generation
  - Symmetric Encryption
  - **Asymmetric Encryption**
  - Hash Functions
- These primitives get combined to achieve the CIA (confidentiality, integrity, authentication) goals of cryptography

# Asymmetric Encryption

- Asymmetric encryption addresses the need to distribute keys in support of confidential, authenticated communications with random parties without revealing a secret key
- It uses two matched keys that provide the ability to encrypt/decrypt securely and with confidence that the communication has not been hijacked
- It can use a certificate that includes the certificate holder's identity and public key plus a signature from a trusted third party that guarantees the certificate contents are valid
- The authentication of parties is based on certificates being part of a PKI, or by implementing a WoT model
- Asymmetric encryption is only suitable for small amounts of data, hybrids using both symmetric and asymmetric are required for real-world solutions

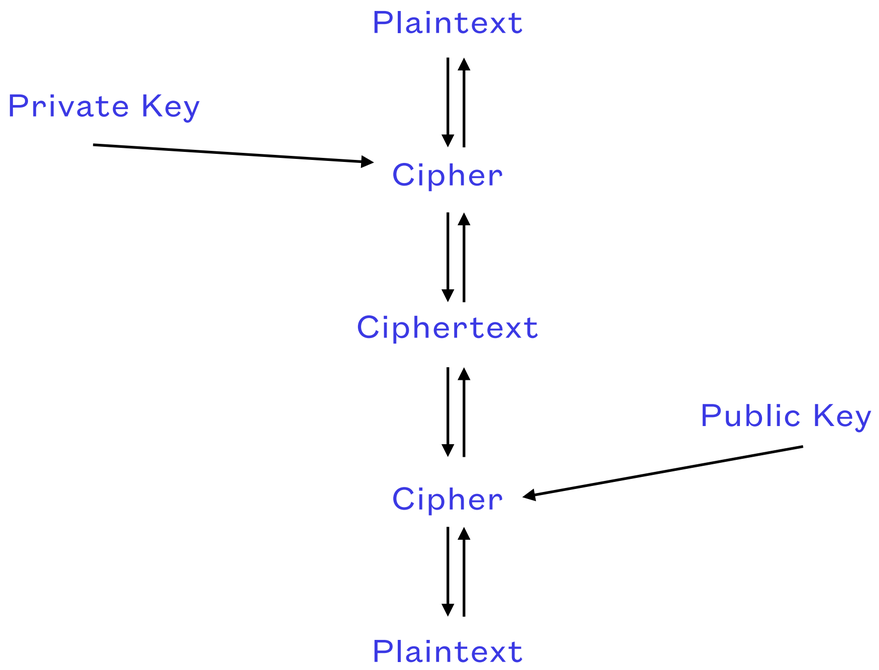
**PKI**

- Public Key Infrastructure

**WoT**

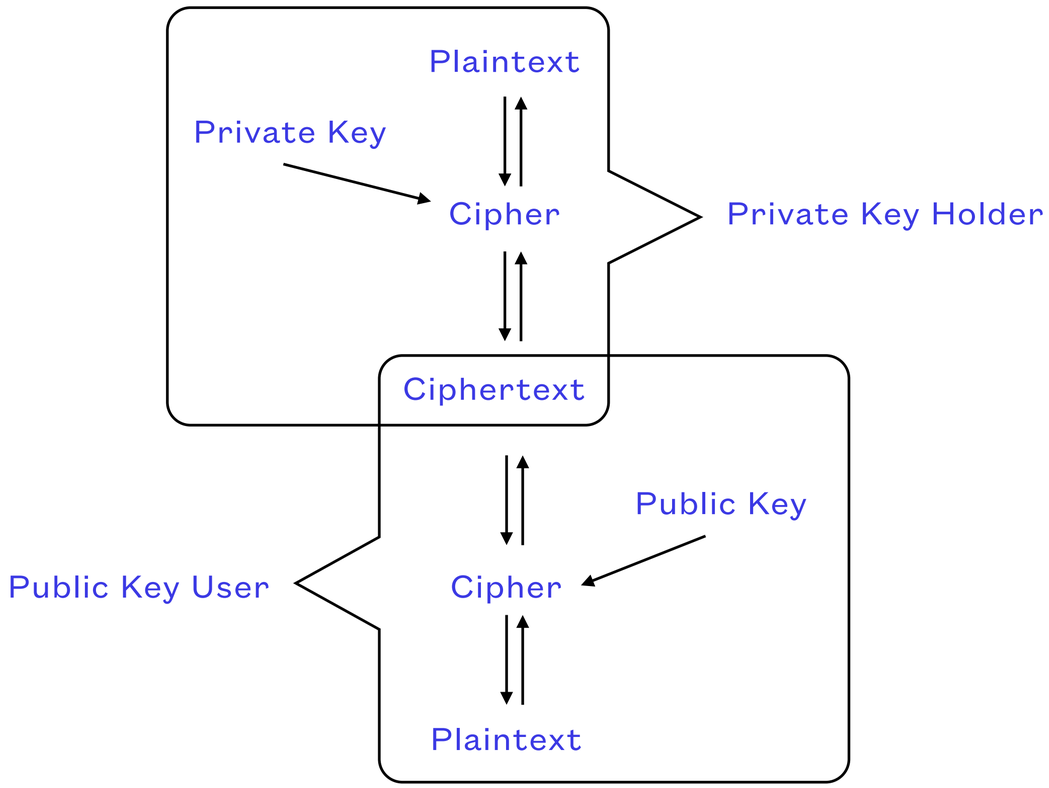
- Web of Trust

# Asymmetric Encryption
## Private/Public Keys



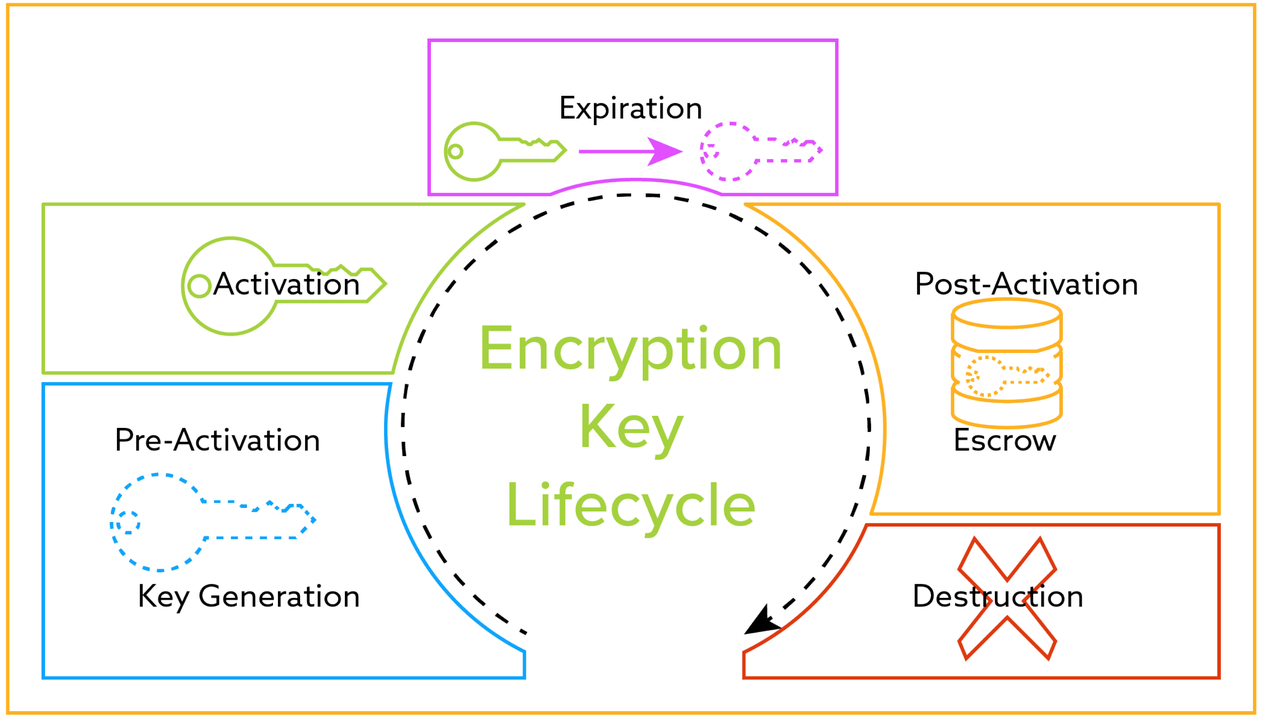

- An entity creates a randomly-generated private key
- The entity then derives and publishes a public key using the private key which only matches that specific private key
- Anything encrypted with the private key can only be decrypted with the matching public key
- Anything encrypted with the public key can only be decrypted with the matching private key
- The public key can be widely distributed and is not secret in any way
- The private key must remain secret

# Asymmetric Encryption
## Identity Confidence




- The private key holder can encrypt messages with the private key and anyone with the public key can read it and be confident that it could only have come from the holder of the private key
- Anyone can encrypt messages with the public key and be confident that only the holder of the private key can read it
- Asymmetric encryption guarantees that anyone can communicate with a private keyholder and be confident they are communicating with the private keyholder that matches the public key they are using
- This confidence is predicated on the infeasibility of determining the private key from the public key which is itself a result of using very large primes and number theory in the formula used to generate the public key
- The Public Key Infrastructure model provides a way to obtain a valid public key with confidence it is the right key
- The Web of Trust model provides another method for obtaining public keys with confidence that they are the right public keys

# It's all about the keys, man




https://info.townsendsecurity.com/designing-applications-for-encryption-key-management

# Public Key Confidence
## Public Key Infrastructure

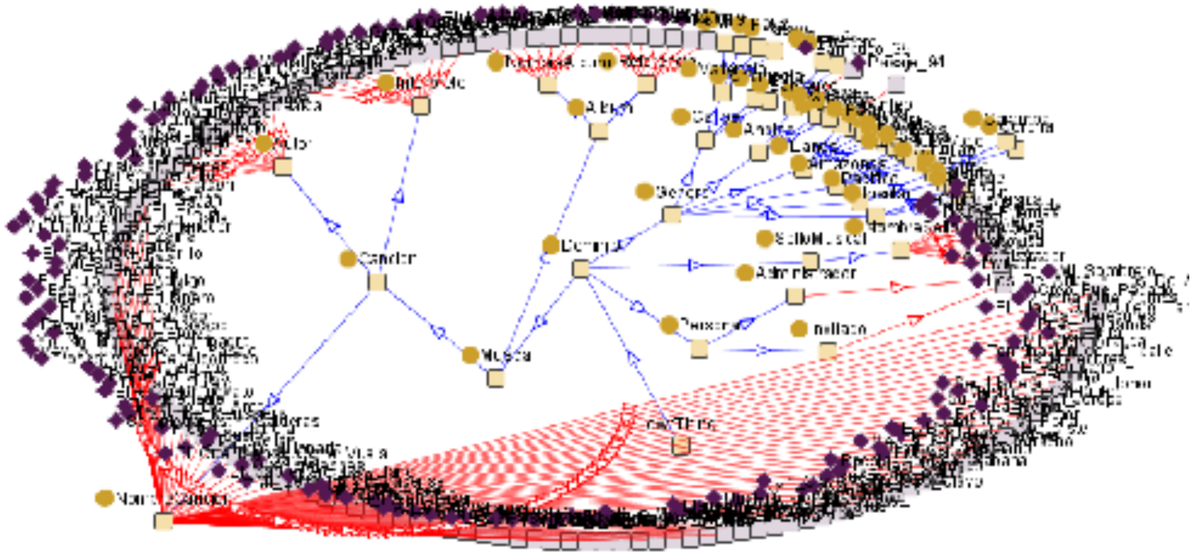

Private CA


Operating System
CA List

- A PKI defines the list of authorities (CAs) that are trusted to sign and revoke certificates
- There is a global internet PKI recognized by all major operating system and browser suppliers which supplies a well-maintained and automatically distributed list of trusted CAs
- Organizations can create their own CAs, and therefore can have their own PKIs with a list of trusted CAs for their own purposes such as VPNs
- Private PKIs can be used along with the internet's PKI on a given client machine or kept separate for specific applications to use
- PKIs are a centralized, "trust the authorities" approach to identity verification

**CA**

- Certificate Authority



https://medium.com/@aitheric/web-of-trust-a-taxonomy-for-claims-d136ae6f2ee3

# Public Key Confidence
## Web of Trust

- In the WoT model, each user builds a list of trusted public keys (their keyring) over time and distributes those to others
- WOT certificates can have multiple public keys endorsing them to raise confidence in them
- When you get public keys trusted by someone you already trust, you can add the new keys to your trusted keyring with tags to say how much you trust them
- There are public keyring servers that you can register your public keys in so that people who trust those servers also can retrieve and trust your public key
- Some people put their public key on their business card or website to encourage private communications
    - e.g. Journalists may publish their public keys on their employer's website to encourage secure communication directly with the journalist, while keeping that communication private to the journalist and the person contacting the journalist
- Web of Trust is a decentralized, "only trust the people you know" scheme

# Key Privacy

- Public keys do not need protection from prying eyes, but do need to be widely available and verifiably authentic
- Private keys need protection from prying eyes, and must be strictly restricted to the owner of the keypair
- Programs that generate keypairs can display them or store them
- If they are only displayed the user must deal with storing them, having them on the screen even temporarily exposes them to snoopers
- Key generating programs can normally encrypt the file the private key is stored in, as the file is created
- Personal key files may be kept on removable devices to reduce opportunities to attack them and make them portable

# Key File Formats

- Keys are stored in ASN.1 data structures, and are binary
- The structures vary depending on the key type, and options used when it was generated
- Various file formats exist for storing key files
  - SSL/TLS keys common uses DER(X.690), PEM, PKCS #12 for the private key, and public keys get stored as part of certificate files
  - SSH keys use PEM with extension .pub for the public key file, and no extension for the private key file
  - Key filename extensions are typically .der for DER, .pem or .key for PEM, .p12 for PKCS#12
  - See RFC7468 for full details on these formats and how they relate to the standards
- No matter which format is used, the private key should be stored encrypted

**ASN.1**

- Abstract Syntax Notation One
- A data structure definition language

**DER**

- Distinguished Encoding Rules, a derivative of X.690 format for encoding binary data

**PEM**

- Privacy-enhanced mail format for storing base64 encoded data

**PKCS #12**

- Public Key Cryptography Standards standard number 12 for encoding PKC data

# Creating keys
## SSH keys with ssh-keygen

```
ssh-keygen -t ed25519 -C "dennis on mbp"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/dennis/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/dennis/.ssh/id_ed25519.
Your public key has been saved in /home/dennis/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:KUE7q8d/i2BgdJWzxbjWfmSY+xeGUb3zGIVTyHbyk7g dennis on mbp
The key's randomart image is:
+--[ED25519 256]--+
|      . .+   . =.|
|     . o+ o   O +|
|    . =  * o o.*o|
|   . . ++.+ +..=.|
|    o o.S. + o.o+|
|   . + .  o oE+ .|
|    . =    o . . |
|     o o .. . .  |
|        o... .   |
+----[SHA256]-----+
```

puttygen for windows

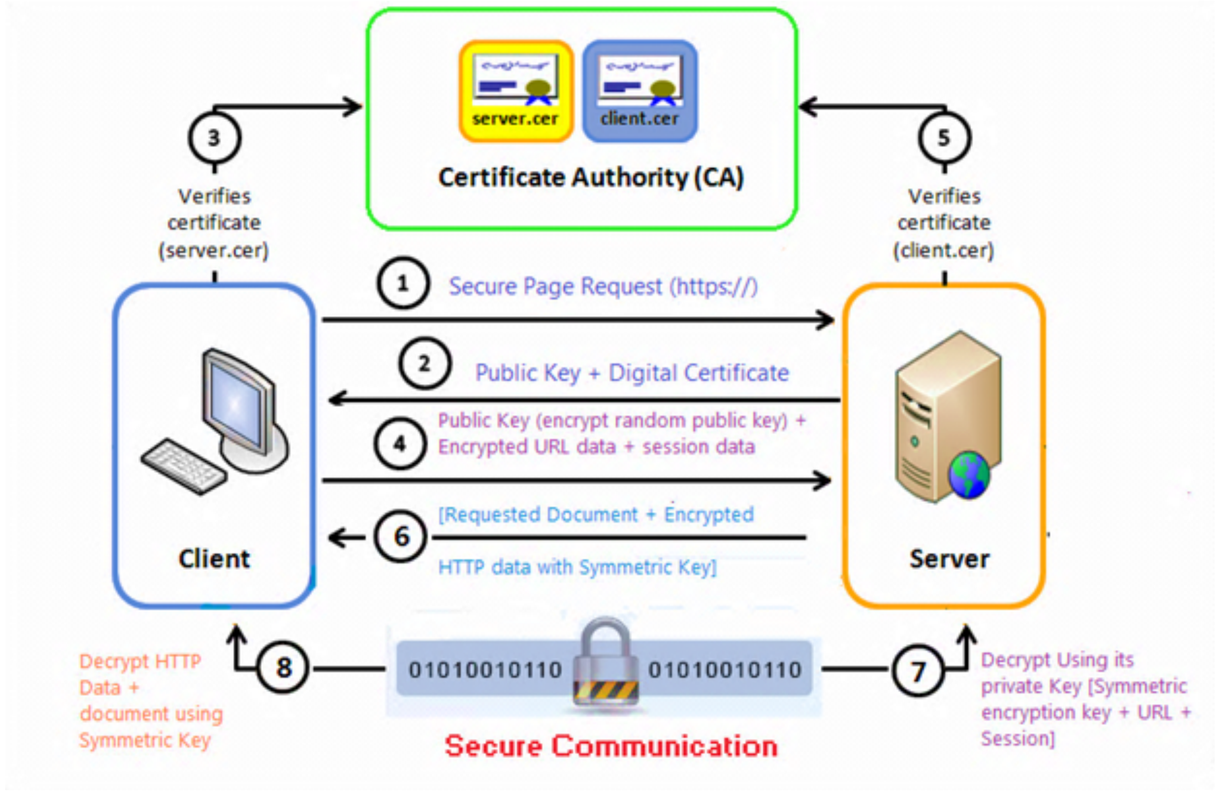
- Begin with a random hunk of data and use it to generate a private key/public key matched pair using the RSA, DSA, ECDSA, or ED25519 crypto algorithm
- ssh-keygen will generate a keypair and store them, offering to encrypt the private key file
- puttygen is a windows equivalent
- websites can generate ssh keys for you but then those websites potentially have access to your private key as well as the public one
- ssh public keys may be signed by an SSH CA

## Distributing/Installing SSH Public Keys

# The art of talking to yourself

https://www.tomorrowsworld.org/commentary/are-you-talking-to-yourself

- With SSH, a user or application will have possession of their private key, in a secure file
- Whatever service they intend to connect with should have their public key installed
- When a user creates an SSH keypair, it is for the purpose of establishing a connection, so once the keypair is generated, the public key is normally then immediately copied to wherever it needs to be so that the desired service programs can access it
  - e.g. ssh-copy-id user@server
- Some SSH tools require you to create keys with one application and use those keys in another and the workflow involves exporting keys from a key generator then importing them into a key consumer
  - e.g. putty and puttygen
- No matter how you get them there, typically the person who makes the keys puts them where they need to be for later communications, they only depend on themselves for key distribution and validation




https://resources.infosecinstitute.com/ssl-dot-net-volume-1-hypothesis/

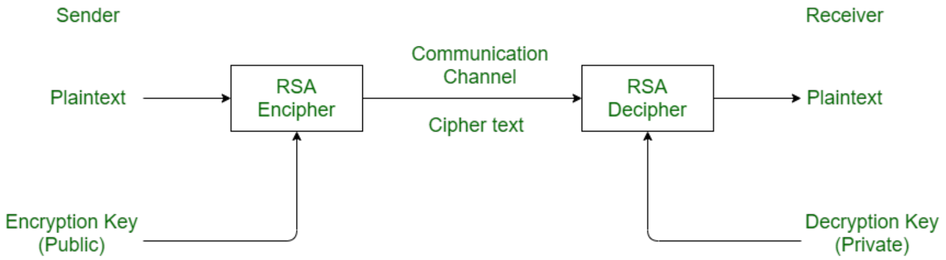
# Sharing PKI Public Keys
## CA role

- Public keys that are distributed using PKI are distributed inside certificate files
- Certificate files are supplied during establishment of connections, and trust is derived from trust of the certificate signature
- The certificate files associate the owner information with the public key

# Encrypt/Decrypt with PKI

- SSL/TLS and key exchange components of other protocols are the primary PKI-based application of asymmetric encryption
- The requirement for certificates that are centrally managed makes this kind of key management amenable to online services
- The use of private CAs makes PKI-based encryption suitable for use in intranet security, such as for VPNs or intranet-based application services for which authenticated key exchange is required
- Since PKI-based asymmetric encryption is constructed around the centralized key management model, this type of encryption is normally embedded in applications such as browser, web servers, vpn clients, and custom applications
- PKI-based communications are primarily used to instantiate secure communications channels based on symmetric encryption, and then carry arbitrary traffic over that connection
- In addition to deciding how to manage keys, providers of asymmetric encryption also must choose an algorithm for encryption/decryption and that algorithm must be specified when creating keys for the algorithm, but the choice is fairly straightforward

# RSA Algorithm




https://www.geeksforgeeks.org/difference-between-rsa-algorithm-and-dsa/

- RSA is one of the oldest public key cryptosystems, having been published in 1977 to improve on the DH algorithm in key exchange by adding authentication (combat MITM attacks)
- RSA security is based on the difficulty in factoring large numbers, and the recommended minimum key length has been increased a few times as computing power and the ability to distribute the task has increased
- Current minimum key length recommendation is 2048 with 3072 bits being deployed by some organizations (e.g. Google) by default now despite performance hit
- The longest demonstrated broken key is 829 bits (took 2700 cpu years in a distributed systems attack), see https://en.wikipedia.org/wiki/RSA_numbers#RSA-250
- The use of RSA algorithm is considered a best practice with a key length of 2048 (112 bit security level)
- If 128 bit security or better is required, then ECDSA is considered best practice

**RSA**

- Rivest–Shamir–Adleman

**DH**

- Diffie-Hellman

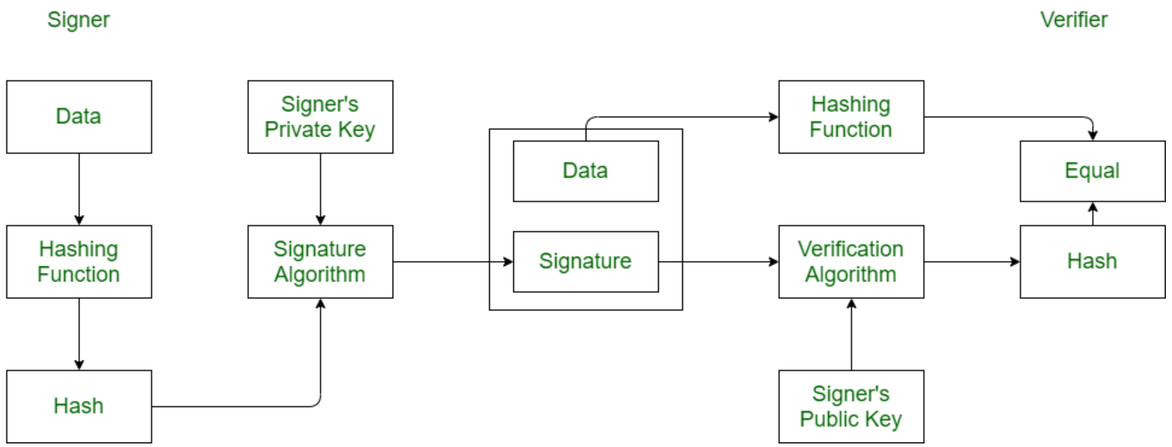
**MITM**

- Man in the middle

**ECDSA**

- Elliptic-curve Digital Signature Algorithm

# DSA
## Digital Signature Algorithm




https://www.geeksforgeeks.org/difference-between-rsa-algorithm-and-dsa/

- DSA is very similar to RSA, and is a public key cryptosystem
- Created in 1991 by NIST
- NIST pushed for this to be the federal standard over RSA for signature generation/verification
- DSA depends on discrete logarithm math instead of factoring
- DSA uses a nonce which is dependent on the quality of RNG used - e.g. SSH2 required if you want to use DSA, SSH1 had RNG problems
- DSA is deprecated for encryption and only used for signatures now

# Elliptic-curve Algorithms

- EC algorithms are based on the algebraic structure of elliptic curves, but is still essentially based on math that it is infeasible to reverse, and was first proposed in 1985
- Modified DH key exchange using ECC is an improvement over RSA key exchange and is called ECDH ( relabeled to X25519 recently as NIST curve 25519 is used)
- Modified DSA using ECC is a newer form of DSA called ECDSA that is less resource intensive than DSA but has similar security issues
- The NSA allows the use of ECDH and ECDSA for protecting information up to the "top secret" level when using 384 bit keys
- EC algorithms are preferred over RSA for the smaller key size, reducing resource consumption for storage and transmission by about 10:1 for comparable security - matters more to mobile devices and high-volume servers than desktop or low-volume systems

**EC**

- Elliptic-curve

**ECC**

- Elliptic-curve Cryptography

**ECDH**

- Diffie-Hellman key exhange using EC math

# Edwards Curve Algorithms

- The elliptic-curve algorithms use math that relates to elliptic curve geometry
- Different curves can have different geometries and still be elliptic - the Ed algorithms use curves from the Twisted Edwards Curve family
- EdDSA generates a nonce using a hash, avoiding the problem that DSA and ECDSA have with the nonce being overly dependant on the RNG used to create their nonces
- Curve25519 is the basis for the ECDH  key exchange algorithm
- Curve25519 is the basis for the Ed25519 signature algorithm
- NIST has proposed the Ed448 signature algorithm too

# Creating keys
## PKI keys with openssl

```
openssl list -cipher-algorithms
openssl list -public-key-algorithms

openssl genpkey -algorithm ED448 -bf -out myprivatekey.PEM
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

openssl pkey -in myprivatekey.PEM -bf
Enter pass phrase for myprivatekey.PEM:
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIGjME8GCSqGSIb3DQEFDTBCMCkGCSqGSIb3DQEFDDAcBAimZHWEUpnf0AICCAAw
DAYIKoZIhvcNAgkFADAVBgkrBgEEAZdVAQIECCFQZaHt9DBeBFDMUFsaYyx4hOx6
W4dJsTSbKVh4+zwTUZO5NpSbfQH/xY4XNSuAJo5Rh79Gis4N+kKIYb5KG0jDPPNa
R3SjYXCGX8CDDwskX2MeFHr2+dkbvw==
-----END ENCRYPTED PRIVATE KEY-----
```

```
openssl pkey -in myprivatekey.PEM -bf -text -pubout
Enter pass phrase for myprivatekey.PEM:
-----BEGIN PUBLIC KEY-----
MEMwBQYDK2VxAzoAk2FUXLE9ih5m1s00zS0zVhTU0DiSd+B6cvNqTiGkK4K/Spao
NFqrrv4RHuqII3XFDebkCiLUY8yA
-----END PUBLIC KEY-----
ED448 Private-Key:
priv:
    e0:3f:0e:73:3f:ac:c3:f8:e2:f9:f5:de:aa:63:48:
    40:a4:91:7b:99:3e:6f:2a:aa:f9:75:a9:b0:c6:3c:
    98:c0:1e:b4:96:b3:b2:e3:80:73:38:92:ba:c7:21:
    7e:1a:0c:fa:89:5d:9f:cb:ae:2c:b2:8a
pub:
    93:61:54:5c:b1:3d:8a:1e:66:d6:cd:34:cd:2d:33:
    56:14:d4:d0:38:92:77:e0:7a:72:f3:6a:4e:21:a4:
    2b:82:bf:4a:96:a8:34:5a:ab:ae:fe:11:1e:ea:88:
    23:75:c5:0d:e6:e4:0a:22:d4:63:cc:80
```

```
openssl pkey -in myprivatekey.PEM -bf -text
Enter pass phrase for myprivatekey.PEM:
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIGjME8GCSqGSIb3DQEFDTBCMCkGCSqGSIb3DQEFDDAcBAhrz9nQooKo8wICCAAw
DAYIKoZIhvcNAgkFADAVBgkrBgEEAZdVAQIECFyvQ7aUswl5BFDVvpOI4bm44bYO
+1UkxGNpx7RnwN4iSDyEjCRY+16Tn+W8vYXsBBKBrFZrtnaANGQ0eIaa7YJMl8rV
Nvll9EokebovmUmgIRSdrE3/h2sxHw==
-----END ENCRYPTED PRIVATE KEY-----
ED448 Private-Key:
priv:
    e0:3f:0e:73:3f:ac:c3:f8:e2:f9:f5:de:aa:63:48:
    40:a4:91:7b:99:3e:6f:2a:aa:f9:75:a9:b0:c6:3c:
    98:c0:1e:b4:96:b3:b2:e3:80:73:38:92:ba:c7:21:
    7e:1a:0c:fa:89:5d:9f:cb:ae:2c:b2:8a
pub:
    93:61:54:5c:b1:3d:8a:1e:66:d6:cd:34:cd:2d:33:
    56:14:d4:d0:38:92:77:e0:7a:72:f3:6a:4e:21:a4:
    2b:82:bf:4a:96:a8:34:5a:ab:ae:fe:11:1e:ea:88:
    23:75:c5:0d:e6:e4:0a:22:d4:63:cc:80
```

- Begin with a random hunk of data to use as a private key, these can be generated using openssl and a cipher
- Openssl will generate a public key from that private key automatically so that you have a keypair
- There are lots of other ways to create your keys, but they need to be random to be good
- There are web sites that can do this, but your keys are then exposed to the website owners

# OpenPGP

- OpenPGP is an Internet standard for WoT cryptography that is the result of decades of legal wrangling, ownership fights, corporate transfers, and US federal government meddling in open source
- GPG is OSS and OpenPGP-compliant and is the toolset most security nerds think of when thinking about OpenPGP
- There are several commercial closed source PGP software toolsets and libraries supported by the vendors of said software
- PGP is owned by Symantec, they sell the Symantec Encryption Desktop (SED, not to be confused with self-encrypting disk) product and the Symantec Encryption Management Server (SEMS) for the Windows and MacOS platforms
- Symantec also provides the PGP Command Line product, and the PGP Support Package (PGP product for blackberry devices)
- OpenPGP compliant applications being used in significant volumes today include email signing, LibreOffice document signing, and protected activist/journalist communications

```
gpg --gen-key
gpg (GnuPG) 2.2.4; Copyright (C) 2017 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Dennis Simpson
Email address: dennis@zonzorp.net
You selected this USER-ID:
    "Dennis Simpson <dennis@zonzorp.net>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 5F32926F333351C7 marked as ultimately trusted
gpg: directory '/home/dennis/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/dennis/.gnupg/openpgp-revocs.d/
1B1382412A8A6B2C823065425F32926F333351C7.rev'
public and secret key created and signed.

pub   rsa3072 2020-10-12 [SC] [expires: 2022-10-12]
      1B1382412A8A6B2C823065425F32926F333351C7
uid                      Dennis Simpson <dennis@zonzorp.net>
sub   rsa3072 2020-10-12 [E] [expires: 2022-10-12]

gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2022-10-12
/home/dennis/.gnupg/pubring.kbx
------------------------------
pub   rsa3072 2020-10-12 [SC] [expires: 2022-10-12]
      1B1382412A8A6B2C823065425F32926F333351C7
uid           [ultimate] Dennis Simpson <dennis@zonzorp.net>
sub   rsa3072 2020-10-12 [E] [expires: 2022-10-12]
```

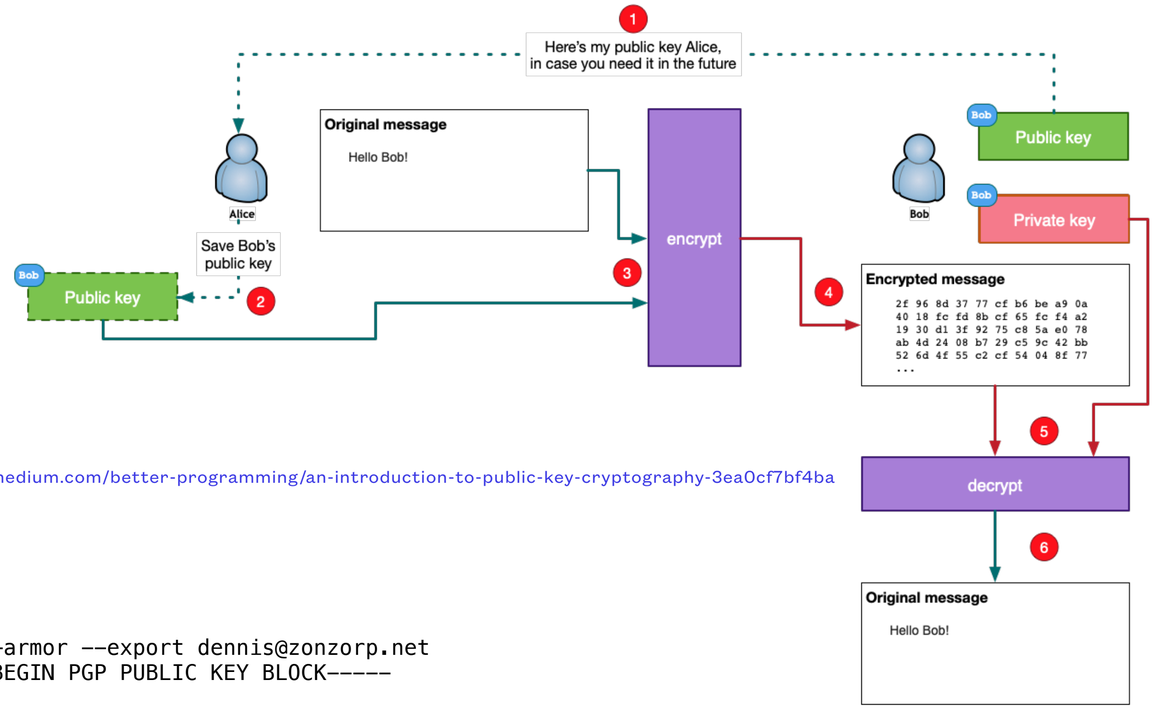
# Creating keys

## WoT keys with gpg

- gpg keys are decentralized, and a gpg keypair may be stored as a keypair bundle, and bundles are often simply referred to as a keypair
- A gpg keypair can have a primary key and multiple subordinate keys, using the ElGamal or DSA algorithms
- The primary key is used to sign other keys
- It may be used to make a subordinate key used for encryption, or it may be used directly for encryption
- Multiple subordinate keys may be created
- gpg keys can have expiry dates
- gpg keys must have an identity string that names who the key belongs to
- gpg key files have their own format and can be encrypted

# Sharing Web of Trust Public Keys

## Giving out your public key




https://medium.com/better-programming/an-introduction-to-public-key-cryptography-3ea0cf7bf4ba

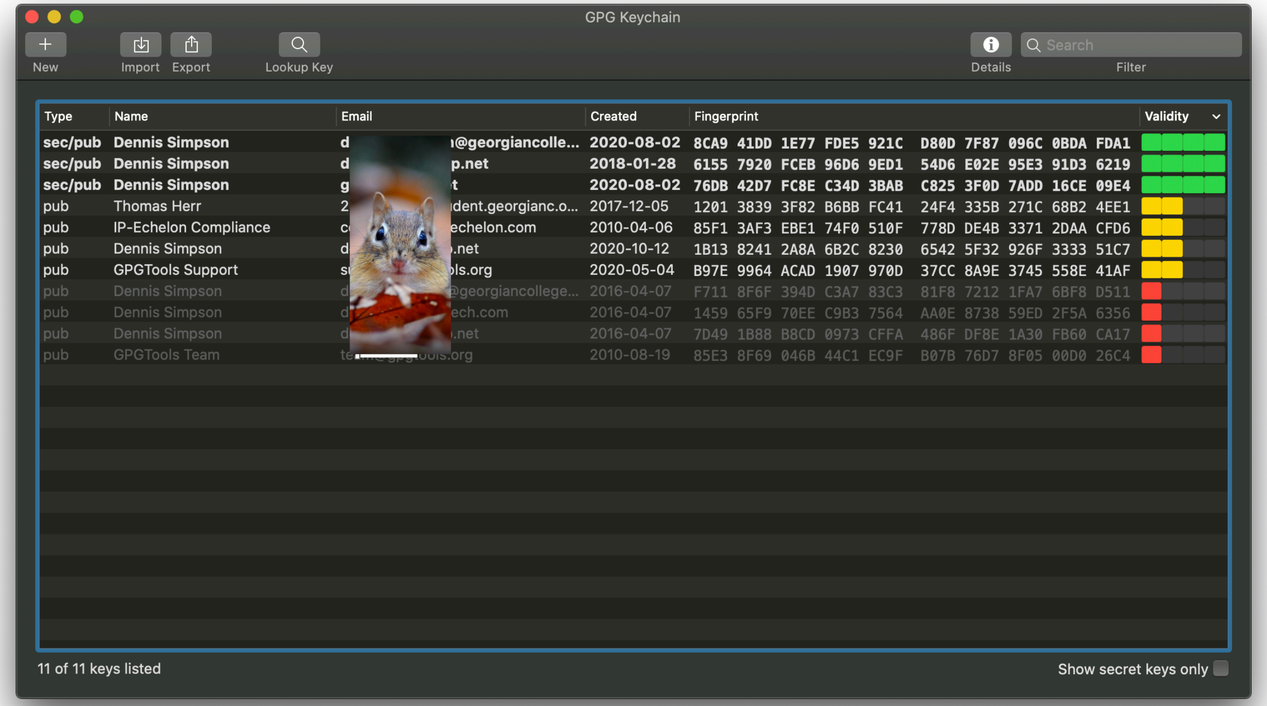
```
gpg --armor --export dennis@zonzorp.net
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF+EdQwBDAC9UIn97POcwZKH4Nq9DkjUBmZvfu0uR47+07AG7wgjCqgkWYXW
.
.
.
c2qPZymJ+EAxFz6OwiFVgtDB8xcv6GbX9Zt3RxtqO7WvLXCMu4BLMXAwg2zcSykd
339VhjQtJcbplJxyW2w9JhVdDXrveblbCgAY
=pVaU
-----END PGP PUBLIC KEY BLOCK-----

gpg --export dennis@zonzorp.net > f1
gpg --armor --export dennis@zonzorp.net|sed -e '/^$/d' -e '/^-----/d' -e '/^=/d'|base64 -d > f2
cmp f1 f2
```

- To communicate with gpg securely, you need to give your public key to whomever wants to communicate with you
- gpg can export your public key and encode it in ascii armor (PGP's name for base64 stored in PEM format) for transmission
- Your public keys can also be submitted to a gpg keyserver so that anyone can get your keys from that server instead of from you, but they have to request them
- Programs that integrate gpg can query the keyserver automatically on your behalf

```
gpg --import pubkey-base64file
gpg: key 5F32926F333351C7: public key "Dennis Simpson <dennis@zonzorp.net>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```




# Sharing Web of Trust Public Keys

## Importing and trusting a public key

- gpg can import keys
- gpg keys are stored in a trustdb file and usually trusted by default
- gpg supports the concept of just how much you trust any particular key
- There are gui tools to help with this
- GUIs are a good way to do GPG key management because they are more or less only used for person to person communications which means you are rarely lacking a desktop computer when working with gpg

# Encrypt/Decrypt with WoT

- The gpg command can be used to encrypt/decrypt data using your gpg keys
- When sending a message, you encrypt it with the recipient's public key
  - e.g. createmessage | gpg --encrypt --sign --armor --recipient dennis@zonzorp.net
- Multiple recipients can be specified
- The receiver decrypts it with their private key
  - e.g. encryptedmessage | gpg --decrypt
- Private key file passwords will be requested as needed
- WoT-based asymmetric encryption is primarily used for protecting specific messages between specific individuals and requires significant effort on the part of users of the technology to use it
- Regular users of WoT messaging will configure their keys into a messaging tool that incorporates support for OpenPGP messaging to reduce their manual intervention to use it