48c910b6614c4a0aa5851aa78571dd1e3c3a66ba

First off…

- A huge field on its own, steeped in mathematics
- We are taking a 1000 mile altitude view
- Crypto is very hard
- Do not try and implement your own crypto!

Full Courses

- If you are interested in crypto then take one of these free online courses:
  - https://www.coursera.org/course/crypto
  - https://www.udacity.com/course/cs387

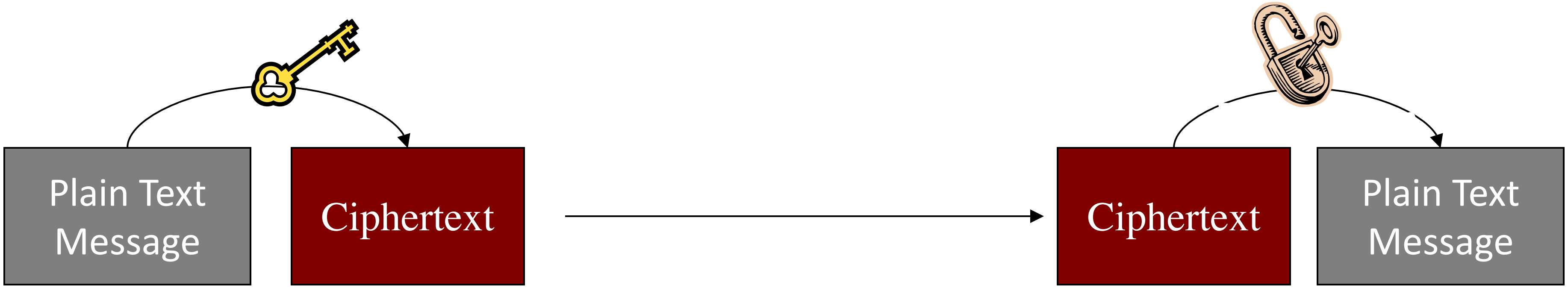
Cryptography

- Cryptography is about mathematics; security is about people.
- Most security problems cannot be solved with magical crypto fairy dust:
  - Denial-of-service attacks
  - Command injections
  - Viruses like Melissa
  - Some attacks against DNS servers
- Sometimes crypto can even “get in the way” of other security technologies e.g. data sanitizers

Goals of Cryptography

- Information Confidentiality
  - Ability to share confidential information over an insecure channel
- Information Integrity
  - Ability to verify that information has not been modified
- Authentication and Non-Repudiation
  - Ability to verify the identity of the author of information and ability to bind information to its author

Overview of Cryptography

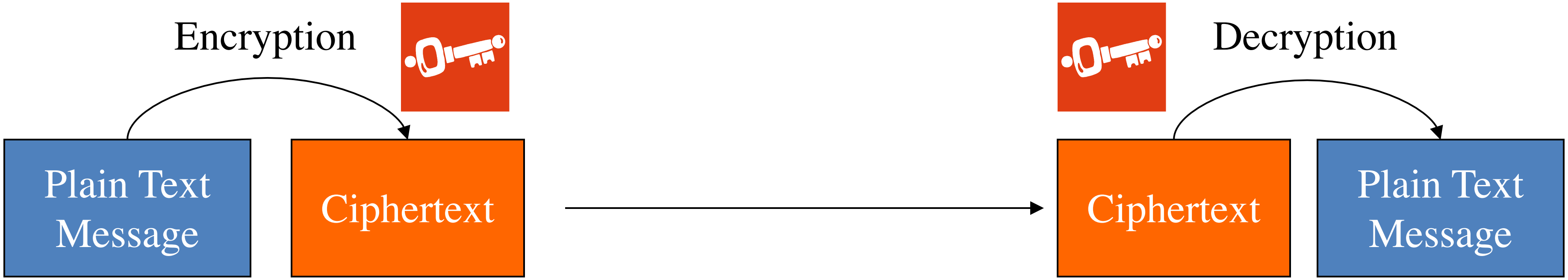


- Encryption and decryption algorithms may use:
  - Symmetric keys – Encryption key and decryption key are the same
  - Or
  - Asymmetric (Public) keys – One key is a secret private and the other is not

Primitive Types

- We have three primary types of crypto functions:
  - Symmetric
  - Asymmetric
  - Hash
- Each has a specific use case

Symmetric Key Cryptography



- In symmetric key cryptography the sender and the receiver share the same secret key (K)
- The plain text message (m) is encrypted using the secret key to obtain the ciphertext K(m)
- The ciphertext is decrypted using the secret key to recover the plain text message
  - K(K(m)) = m

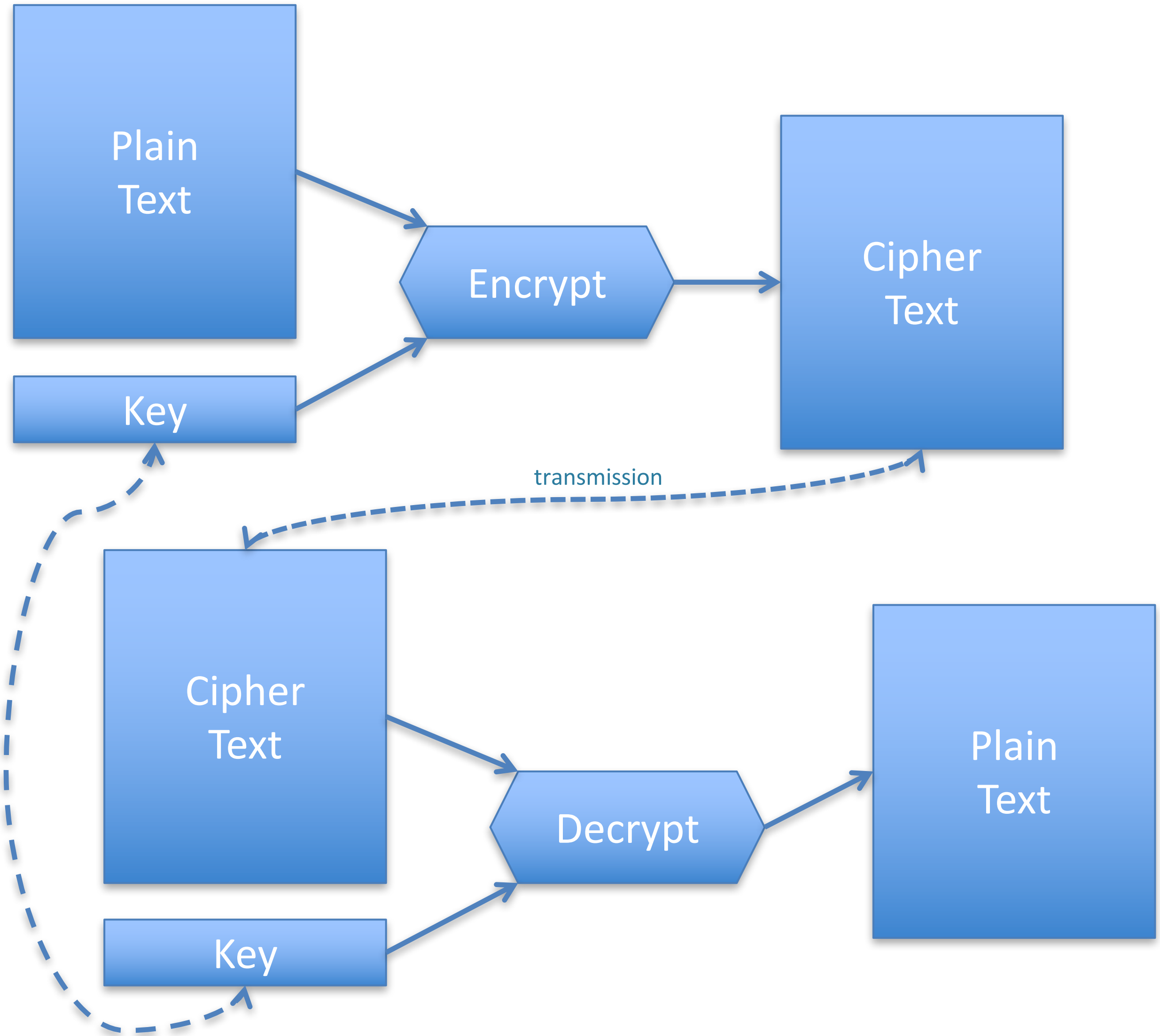
Symmetric Key Cryptography

- The sender and receiver have to agree on the secret key – How can they do this in a secure way?
  - Solution: Key Distribution Centers (KDC), key exchange algorithms or pre-sharing the keys
- If the symmetric key is stolen or discovered the secret message is compromised
- Well known symmetric key algorithms:
  - TDES, AES

Symmetric Operation

Keys should be chosen from a high-entropy random source

Note the same key used to Encrypt is used to Decrypt



Symmetric Algorithms

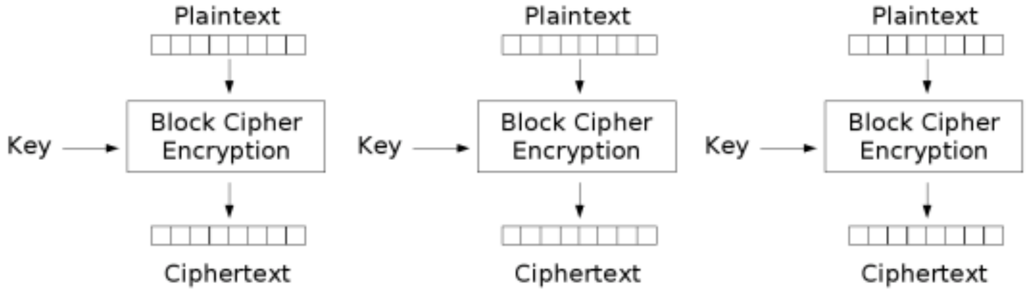
- Common Algorithms and Typical Key Sizes
  - DES – 56-bit
  - 3DES – 168-bit
  - AES – 128 or 256-bit
  - RC4 – 40, 64 or 128-bit
- Modes of Operation
  - ECB
  - CBC
  - CTR or XTS

Symmetric Strength & Weakness

- Well understood, very secure
- FAST
  - AES-NI instructions in modern CPUs
  - 15GB/sec…per core!
- Suffer from the ‘key-exchange’ problem

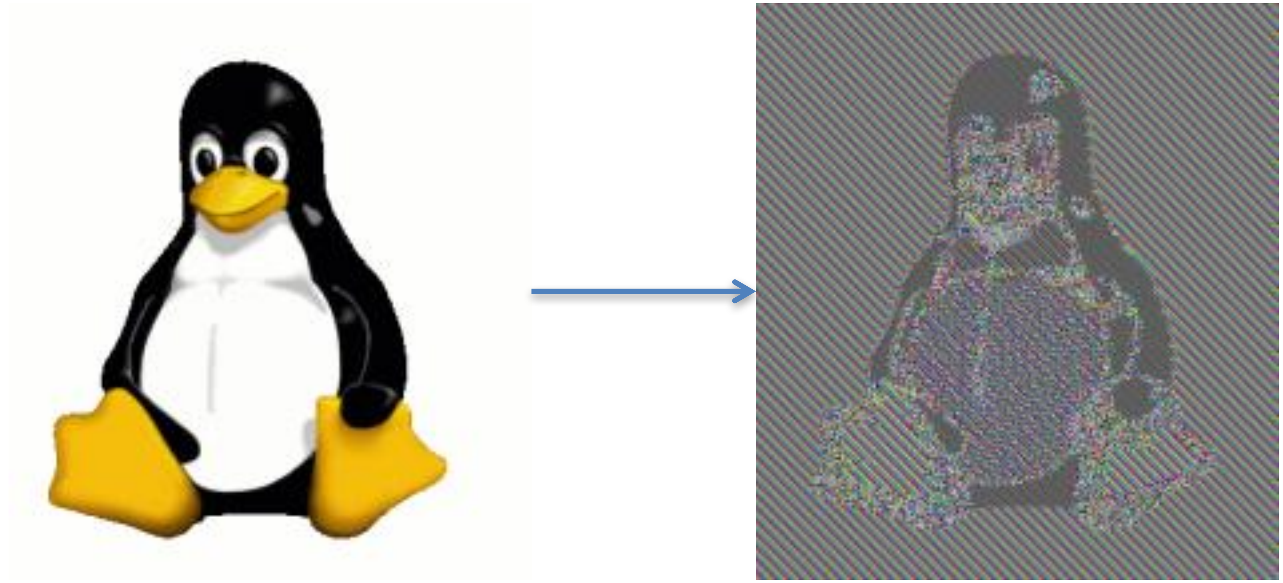
ECB Mode

- Electronic Code Book (ECB) is the simplest method of employing a block cipher
- Allows for decryption of ciphertext block randomly
- Unfortunately it can reveal information about the original data



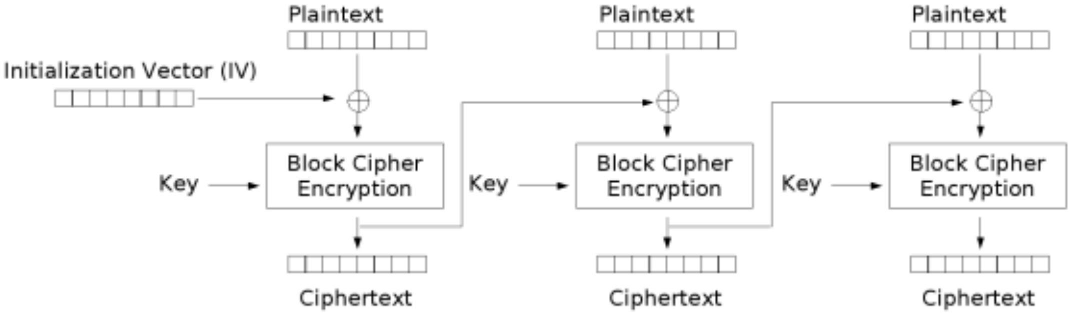
Electronic Codebook (ECB) mode encryption

ECB Illustrated



CBC Mode

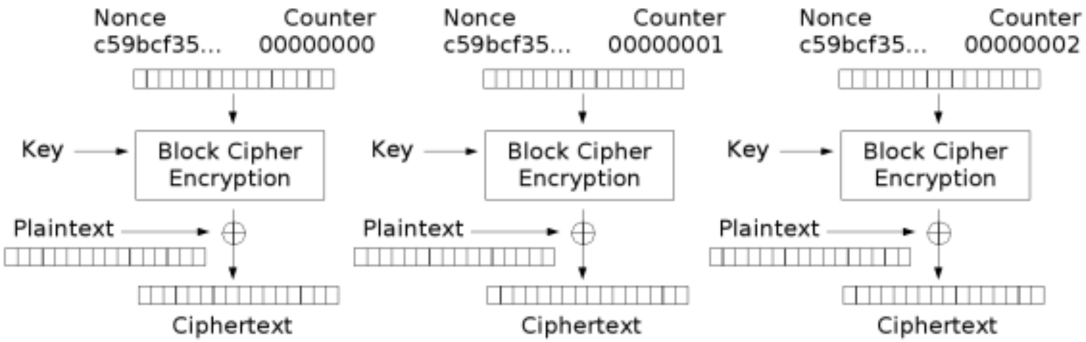
- Cipher Block Chaining mixes data from the previous round into the next round.
- Identical blocks of plaintext, result in differing ciphertext.
- To decrypt block 99 you must first decrypt 1-98



Cipher Block Chaining (CBC) mode encryption

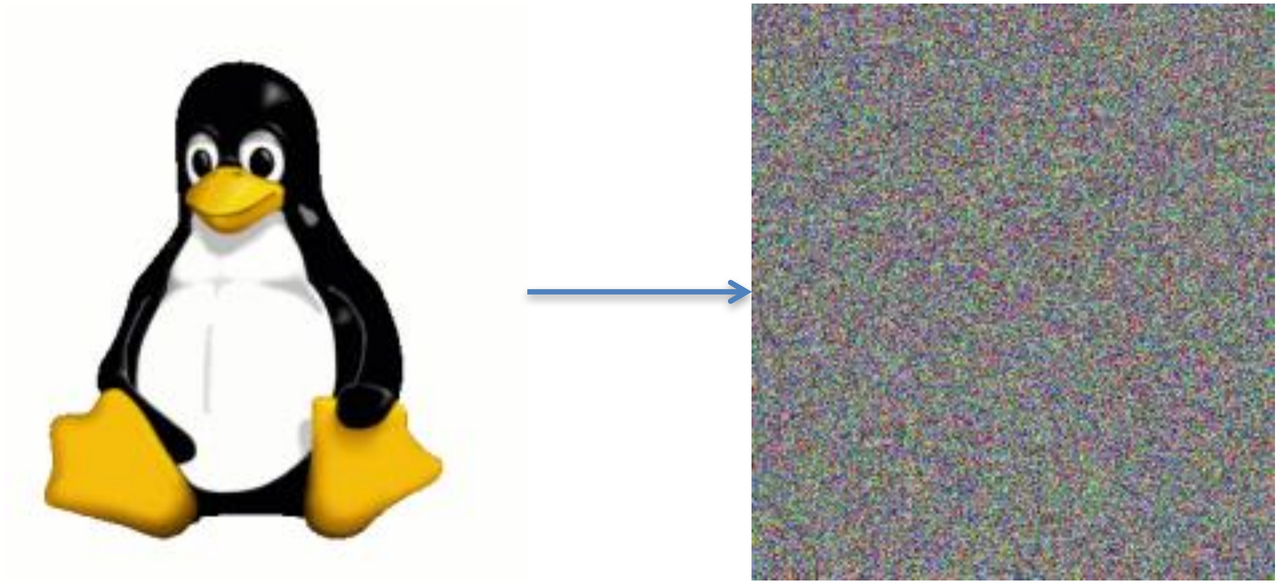
CTR Mode

- Resolves the issue with ECB mode, while keeping the ability to randomly decrypt a block.
- XTS is a similar method optimized for disk encry

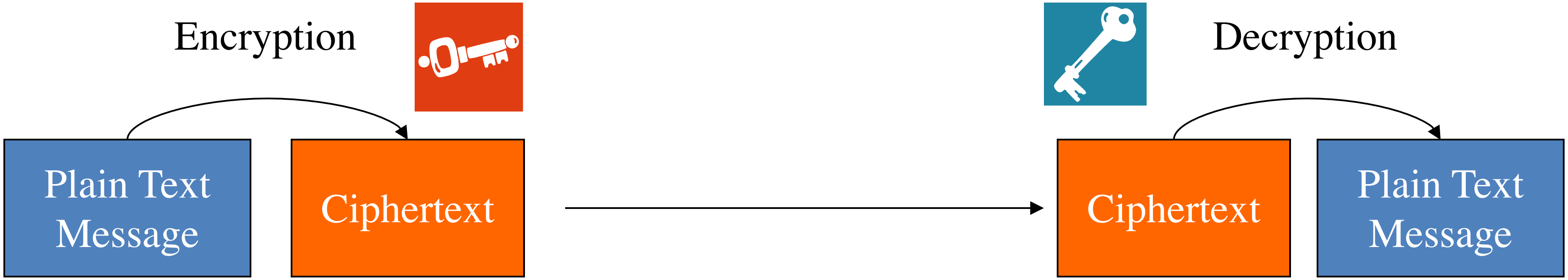


Counter (CTR) mode encryption

Anything but ECB Illustrated



Public Key Cryptography (Asymmetric)



- Public key cryptography uses the mathematical properties of a key pair to encrypt and decrypt messages
- The sender encrypts the message using the receiver’s public key (KRed)
- The receiver decrypts the ciphertext using his private key (KBlue) :
  - KBlue(KRed(m)) = m

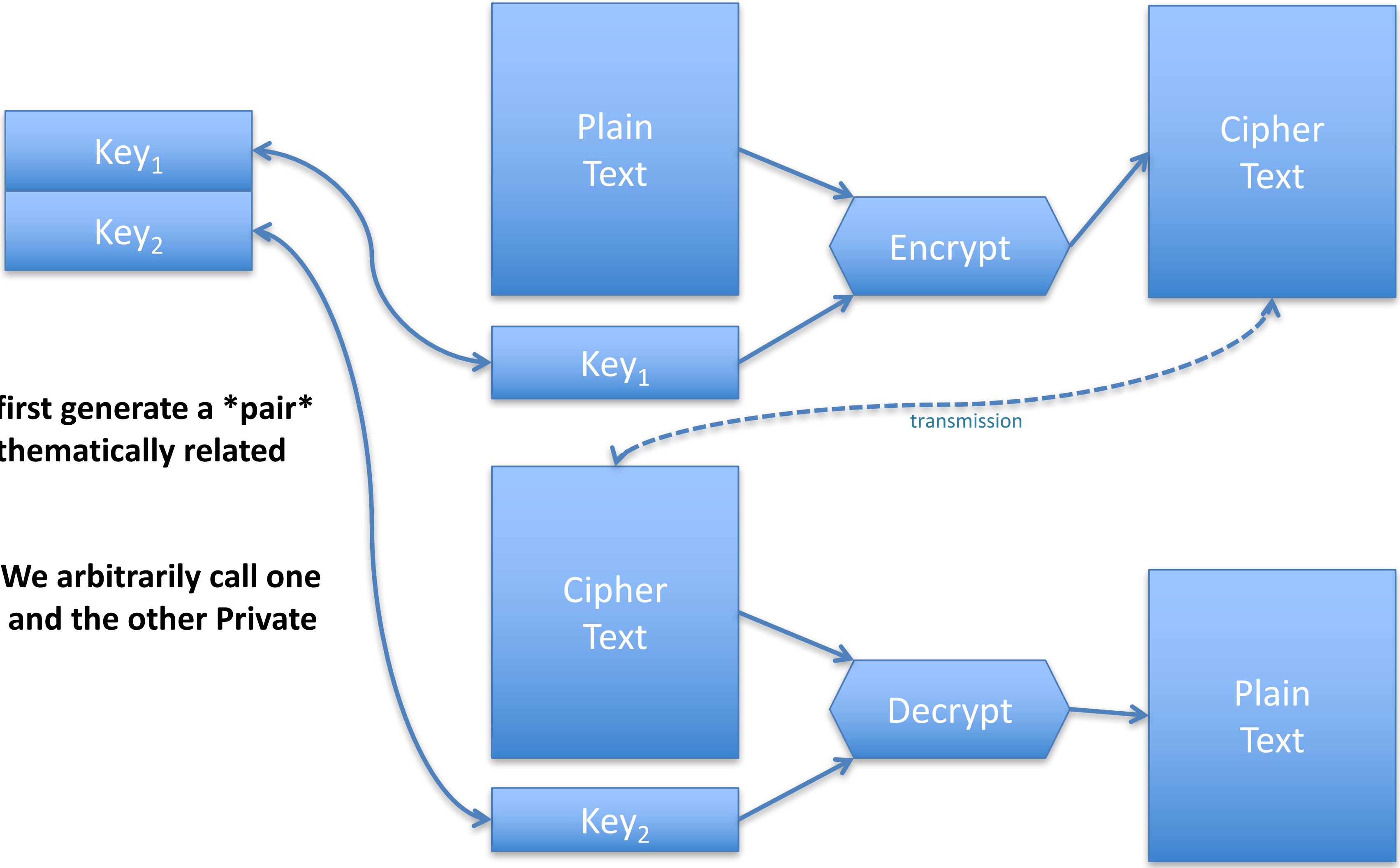
Public Key Infrastructure and Certificate Authority

- How do you authenticate that the public key belongs to someone?
  - Solution: Public Key Infrastructure (PKI) and Certificate Authorities
- A public key infrastructure is a system for a trusted third party to vouch for the authenticity of a public key
- A PKI includes one or more Certificate Authorities which issue certificates that bind public keys to particular individuals or organizations

Public Key Cryptography and Digital Signatures

- Digital Signatures
- Advantage over symmetric key cryptography:
    - Does not need to risk shared private key
    - Management of public keys is easier and safer than symmetric private keys
- However it is computationally expensive to encrypt long messages
    - Solution: Compute the one-way hash (e.g. MD5) value of the message and digitally sign it with the private key
    - This solves multiple problems: message integrity, authentication and non-repudiation are guaranteed

Asymmetric Operation



Must first generate a *pair* of mathematically related keys

Note: We arbitrarily call one Public and the other Private

Asymmetric Algorithms

- Common Algorithms and Typical Key Sizes
  - RSA
    - Key sizes range from 512, 768, 1024, 2048 up-to 8192
    - 1024-bit is similar to 80-bit symmetric
    - 2048-bit is similar to 112-bit symmetric
  - Diffie-Hellman (DH)
    - Similar to RSA
  - Elliptic-Curve (EC)
    - Divide by 2…aka 224-bit EC is roughly equal to 112-bit symmetric

Asymmetric Strength & Weakness

- Allows for the encryption between parties without previously exchanging encryption (symmetric) keys.
- Comparatively slow to Symmetric algorithms
  - And I mean extremely slow!
- Keys are generated, not random.
  - Breaking RSA is a problem of factoring the product of two large primes.
  - We make up for than with large key sizes (1024/2048/more)

RSA Cryptography

- Video time!
- https://www.youtube.com/watch?v=wXB-V_Keiu8

Diffie-Hellman Key Exchange

- More videos!
- https://www.youtube.com/watch?v=YEBfamv-_do

Secret Key vs. Public Key Cryptography

Secret Key (symmetric)	Public Key (asymmetric)
Requires parties to maintain a shared secret	Does not require parties to maintain a shared secret
Logistics of Key exchange is a problem	No Key exchange problems
Parties who mistrust each other may not want to have a shared secret key	No issue of trust since there is no shared secret
Computationally cheap	Computationally extensive

Key Exchange

- Rather than encrypting large amounts of data with Asymmetric algorithms, we instead are better off to use them for exchanging a Symmetric Key
- Use Symmetric functions for what they are good for…fast, secure encryption of bulk data

SSL/TLS

- Video if you want: https://www.youtube.com/watch?v=iQsKdtjwtYI
- SSL/TLS consists of two parts, key exchange, and session encryption
  - Key exchange is done asymmetrically
  - Session encryption is done symmetrically

A bit of history - SSL

- SSL 1.0, 2.0, 3.0
  - “The SSL protocol was originally developed by Netscape.[10] Version 1.0 was never publicly released; version 2.0 was released in February 1995 but "contained a number of security flaws which ultimately led to the design of SSL version 3.0."[11] SSL version 3.0, released in 1996, was a complete redesign of the protocol produced by Paul Kocher working with Netscape engineers Phil Karlton and Alan Freier. Newer versions of SSL/TLS are based on SSL 3.0. The 1996 draft of SSL 3.0 was published by IETF as a historical document in RFC 6101.”

TLS 1.0

- TLS 1.0
  - “TLS 1.0 was first defined in RFC 2246 in January 1999 as an upgrade of SSL Version 3.0. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant to preclude interoperability between TLS 1.0 and SSL 3.0." TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0, thus weakening security.”

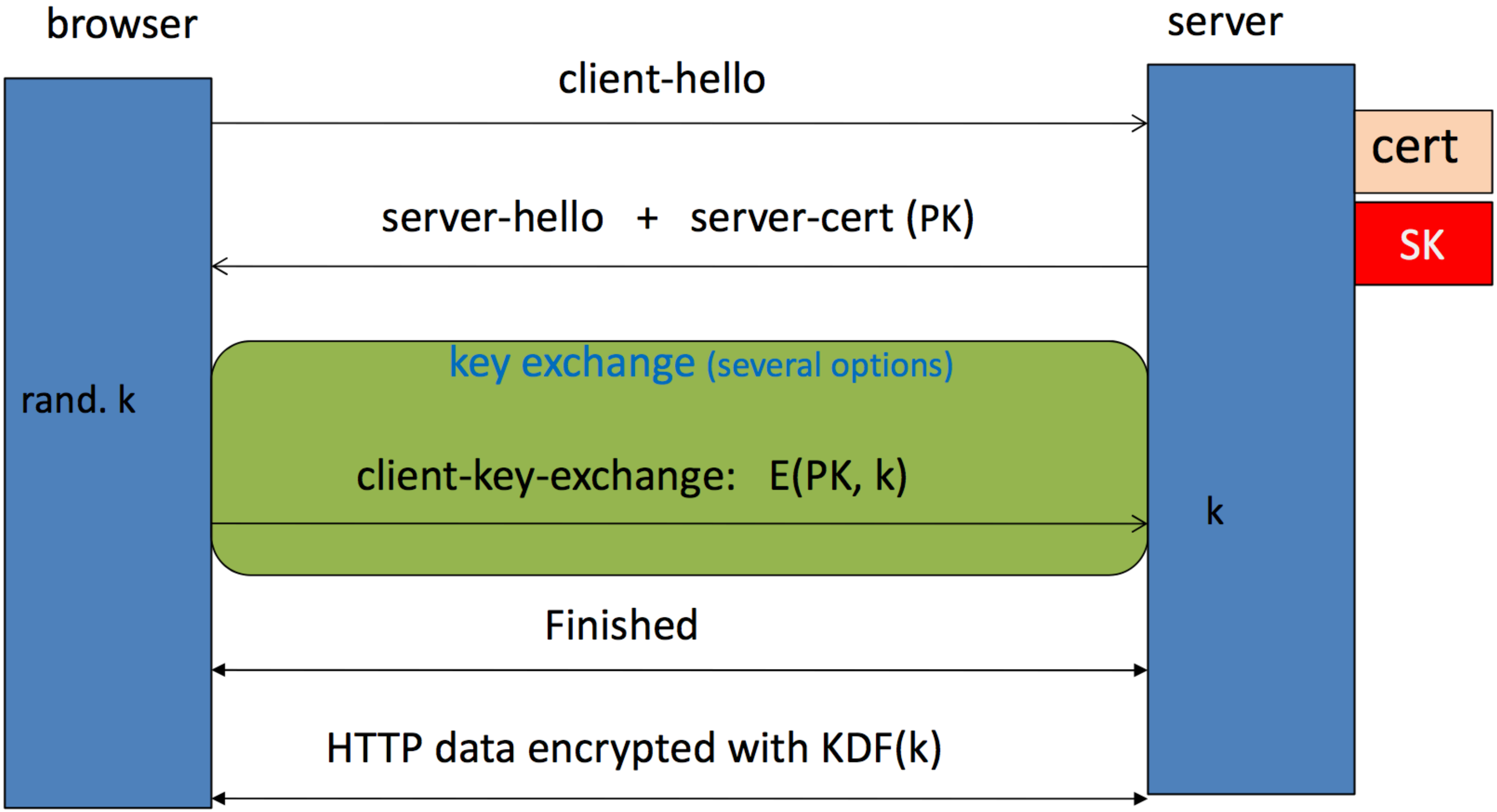
TLS 1.1

- TLS 1.1
  - TLS 1.1 was defined in RFC 4346 in April 2006.[12] It is an update from TLS version 1.0. Significant differences in this version include:
    - Added protection against Cipher block chaining (CBC) attacks.
    - The implicit Initialization Vector (IV) was replaced with an explicit IV.
    - Change in handling of padding errors.
    - Support for IANA registration of parameters.

TLS 1.2

- TLS 1.2 was defined in RFC 5246 in August 2008. It is based on the earlier TLS 1.1 specification. Major differences include:
  - The MD5-SHA-1 combination in the pseudorandom function (PRF) was replaced with SHA-256, with an option to use cipher suite specified PRFs.
  - The MD5-SHA-1 combination in the Finished message hash was replaced with SHA-256, with an option to use cipher suite specific hash algorithms. However the size of the hash in the finished message is still truncated to 96-bits.
  - The MD5-SHA-1 combination in the digitally signed element was replaced with a single hash negotiated during handshake, defaults to SHA-1.
  - Enhancement in the client's and server's ability to specify which hash and signature algorithms they will accept.
  - Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard encryption.
  - TLS Extensions definition and Advanced Encryption Standard cipher suites were added.

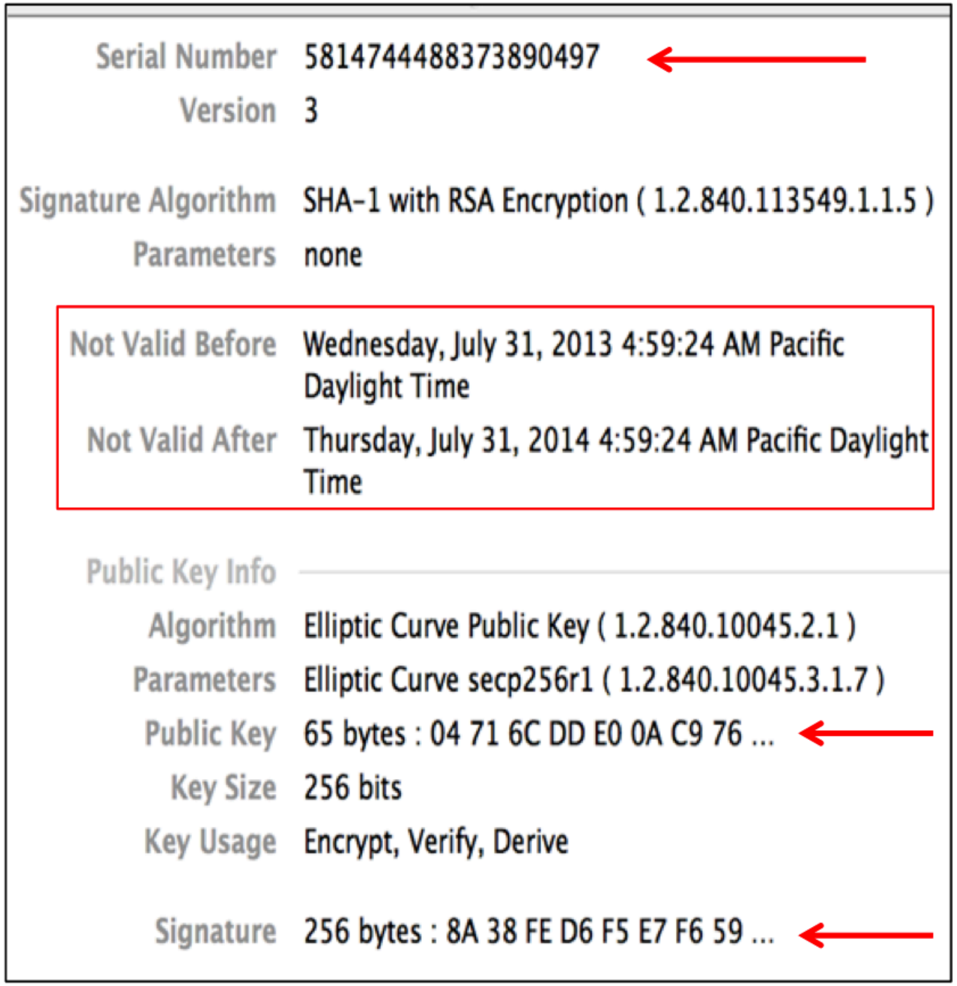
SSL/TLS


browser
server
client-hello
cert
SK
server-hello + server-cert (PK)
key exchange (several options)
rand. k
client-key-exchange: E(PK, k)
k
Finished
HTTP data encrypted with KDF(k)

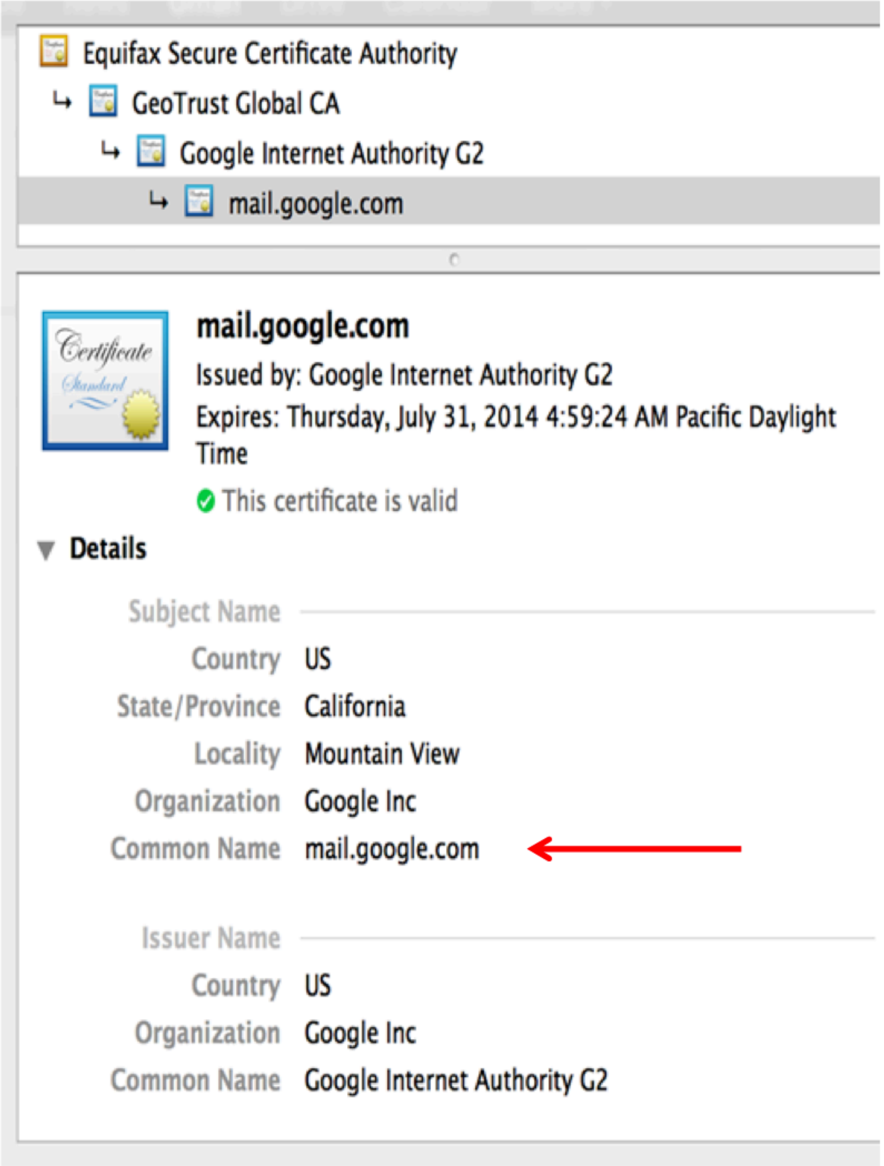
Most common: server authentication only

Certificates

Important Fields:



Serial Number	5814744488373890497
Version	3
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters	none
Not Valid Before	Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time
Not Valid After	Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 71 6C DD E0 0A C9 76 ...
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : 8A 38 FE D6 F5 E7 F6 59 ...



Equifax Secure Certificate Authority
↳ GeoTrust Global CA
↳ Google Internet Authority G2
↳ mail.google.com

mail.google.com
Issued by: Google Internet Authority G2
Expires: Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
This certificate is valid

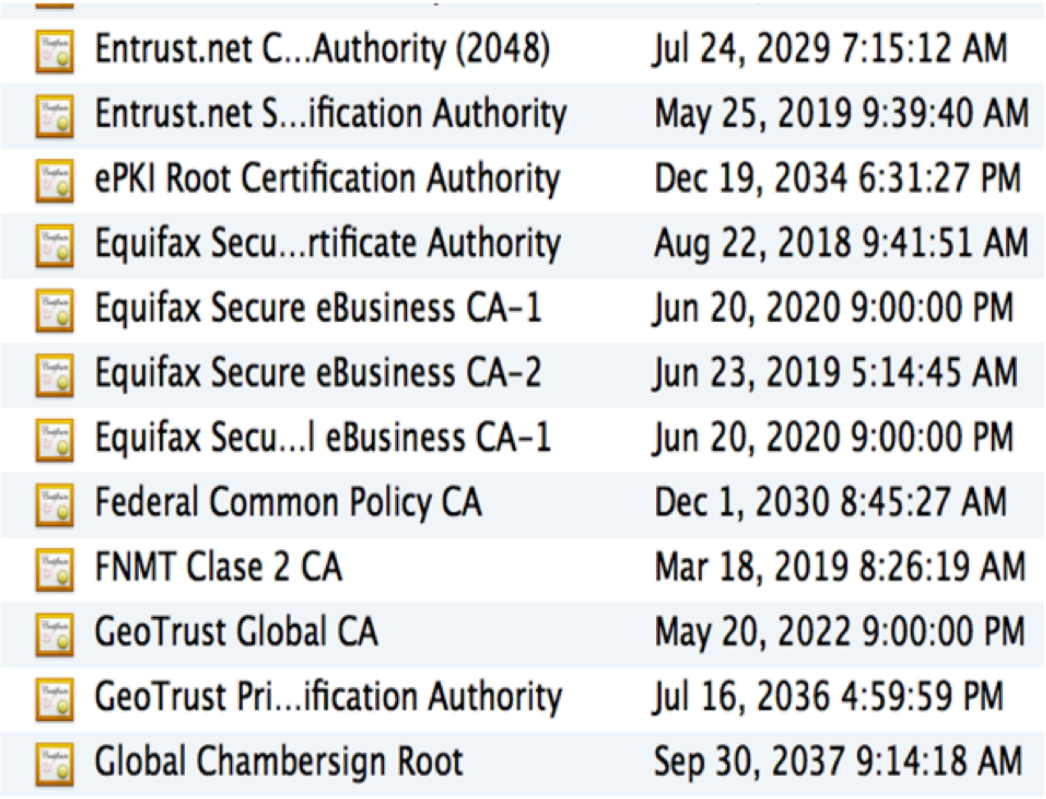
▼ **Details**

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google Inc
Common Name	mail.google.com
Issuer Name	
Country	US
Organization	Google Inc
Common Name	Google Internet Authority G2

Certificate Authorities

- Browsers accept certificates from a large number of CAs
  - Top level CAs ≈ 60
  - Intermediate CAs ≈ 1200

⋮



Entrust.net C...Authority (2048)	Jul 24, 2029 7:15:12 AM
Entrust.net S...ification Authority	May 25, 2019 9:39:40 AM
ePKI Root Certification Authority	Dec 19, 2034 6:31:27 PM
Equifax Secu...rtificate Authority	Aug 22, 2018 9:41:51 AM
Equifax Secure eBusiness CA-1	Jun 20, 2020 9:00:00 PM
Equifax Secure eBusiness CA-2	Jun 23, 2019 5:14:45 AM
Equifax Secu...l eBusiness CA-1	Jun 20, 2020 9:00:00 PM
Federal Common Policy CA	Dec 1, 2030 8:45:27 AM
FNMT Clase 2 CA	Mar 18, 2019 8:26:19 AM
GeoTrust Global CA	May 20, 2022 9:00:00 PM
GeoTrust Pri...ification Authority	Jul 16, 2036 4:59:59 PM
Global Chambersign Root	Sep 30, 2037 9:14:18 AM

⋮

Famous Attacks

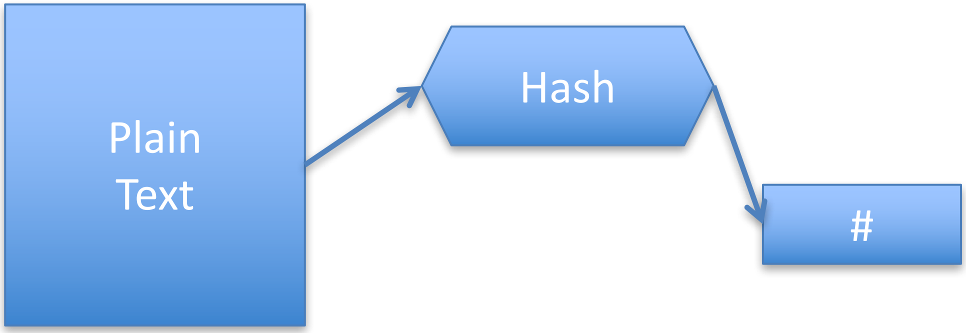
- BEAST
- CRIME
- BREACH
- RC4 Attacks
- Renegotiation Attacks
- Heartbleed

Cryptographic Hash Functions

- Takes a long string (message) and computes a smaller fixed length string (message digest)
- Must have the following properties:
    - Given the message digest, it should be computationally impossible to determine the message in a time period shorter than a brute-force mechanism
    - Given a message m1, it should be computationally infeasible to find a message m2 such that both have the same message digest. i.e. No collisions.
    - Small changes in the message should result in large differences in the message digest

Hash Operation

- Hash (digest) function are very simple. They process the data in blocks, reducing it down and adding the next block.
- The end result is a number.
- Specifically designed so that a tiny change in input results in a large change in output.



md5('The quick brown fox jumps over the lazy dog.') = e4d909c290d0fb1ca068ffaddf22cbd0

md5('The quick brawn fox jumps over the lazy dog.') = 11705d9963455e725bfef4b9c7fa33da

Hash Functions

- Classic Digests: (old & busted)
  - CRC/CRC-32
- Common Hashes: (common design)
  - MD5 – 128-bit
  - SHA1 – 160-bit
  - SHA2 – 192/224/256/384/512-bit
- Newest Hash: (differing design)
  - SHA3 – 224/256/384/512-bit

Hash Limitations

- Have a finite number of outputs for an infinite number of inputs
  - Collisions are inevitable
  - MD5 (and SHA1 / SHA2) suffer from known collision attacks
- Are by design…**extremely** fast
  - Which is a strength
  - And a Weakness

MD5 Hash

- Cryptographic hash function with a 128-bit hash value
- Input message is broken-up into 512-bit blocks
- Widely used to verify the integrity of downloaded files
- Example:
  - Input String: “Cisco Self Defending Networks”
  - MD5 = 096ccf99a07f4314758455d6259747f1
  - Input String: “Cisco Self Defending Network”
  - MD5 = 2377b33fd876d08671d150a4ddfc5734

An extra ***s*** causes the entire hash to be different

- In August 2004 collisions were found:
  - In March 2005 scientists reached a complete collision in 8 hours on a 1.6GHz notebook (http://cryptography.hyperlink.cz/MD5_collisions.html)
  - Does not mean that collisions can be achieved on demand for different size data streams

SHA-2

- SHA (Secure Hash Algorithm)
- Use SHA 256 or above
- Cryptographic hash function with large hash value
- Uses similar principles as MD5 hash function
- In February 2005, a group of Chinese researchers broke the SHA-1 algorithm
  - The entropy was brought down from 280 to 269
- Algorithms will be broken, you should:
  - Design code to allow replacement of crypto algorithms

(H)MAC

- Digital Signatures are great, but RSA is a relatively expensive function
- If you have to perform a large number of signatures (say on a stream of Network Packets), you should consider HMAC
- Combination of Hash + Symmetric
  - Pre-Share or Exchange a symmetric key
  - Hash message
  - Encrypt Hash with symmetric key

Cryptographic Attacks

- Symmetric Attacks
  - Known Plaintext
  - Side-Channel
  - Chosen Plaintext
- Asymmetric Attacks
  - Factorization
  - Common Primes
  - Quantum
- Hash Attacks
  - Rainbow Tables (Pre-Computation or Time/Space Tradeoff)
  - Accelerated Brute-Force (FPGA & GPU)

Hash Cracking

- Simple Hashes > Rainbow Tables
  - Pre-compute the MD5 for all passwords
  - Cracking hashes is now a disk access time problem
    - And not a big one at that
- Salted Hashes > Brute-Force
  - Rainbow tables defeated
  - But modern CPU/GPU/FPGA chips are faaaaassst.
  - CloudCracker.com
- Iterative Hashes
  - Many rounds of Hashing
  - Solely to cost CPU time
  - PBKDF2

Crypto Recommendations

- Algorithms:
  - Should use RSA 2048 for Asymmetric Operations
  - Should use AES 128 or 256 for Symmetric Operations
    - Should use non ECB (default) modes
    - CBC, CTR, XTS & GCM are all considered best-practice
  - Should use SHA2 family (256/512 hash lengths)

Common Crypto Mistakes

- Most common ways to misuse cryptography:
  - Believe that once crypto is added, your implementation is secure
  - Use bad random number generation
  - Poor management of key information
  - Trying to write a custom cryptographic implementation

Standards

- See FIPS-140 & SP-800 documents for currently acceptable encryption for government use.
  - Is referenced by many other standards
  - Consider it your baseline for good crypto
- ISO19790:2112 is the international standard.
  - Derived from FIPS-140-2.
  - http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=52906

Crypto-Libraries

- OpenSSL
  - Everywhere
- Schannel/CryptoAPI (CAPI)
  - Windows
- JCE
  - Java
- CommonCrypto/CDSA
  - OSX and iOS
- All do cryptography well, All have their issues
  - Be careful with defaults:
    - Example: Java happily ignores an IV passed into AES and defaults to ECB mode unless you explicitly ask for CBC.

Sources

- https://en.wikipedia.org/wiki/Transport_Layer_Security