

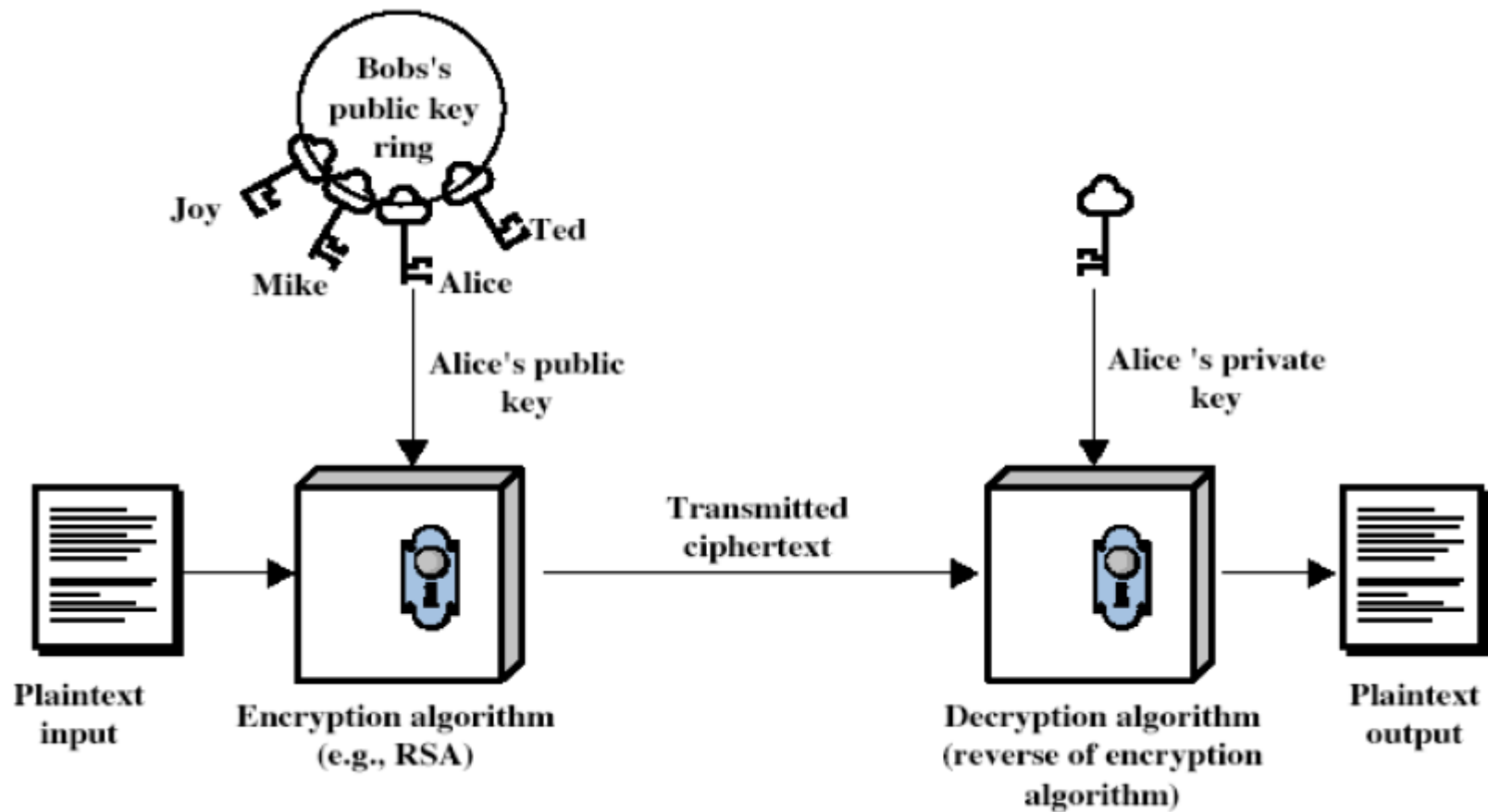
Public Key Cryptography and Cryptographic Hashes

CS461/ECE422

Reading

- Computer Security: Art and Science – Chapter 9
- Handbook of Applied Cryptography
<http://www.cacr.math.uwaterloo.ca/hac/>

Public-Key Cryptography



Public Key Cryptography

- Two keys
 - *Private key* known only to individual
 - *Public key* available to anyone
- Idea
 - Confidentiality: encipher using public key, decipher using private key
 - Integrity/authentication: encipher using private key, decipher using public one

Requirements

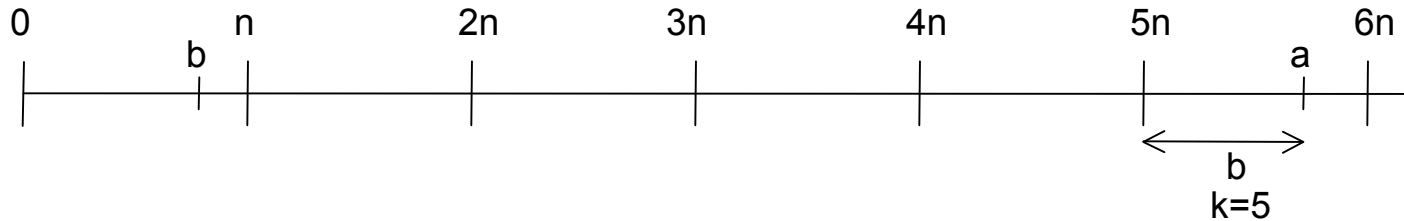
1. It must be computationally easy to encipher or decipher a message given the appropriate key
2. It must be computationally infeasible to derive the private key from the public key
3. It must be computationally infeasible to determine the private key from a chosen plaintext attack

General Facts about Public Key Systems

- Public Key Systems are much slower than Symmetric Key Systems
 - RSA 100 to 1000 times slower than DES. 10,000 times slower than AES?
 - Generally used in conjunction with a symmetric system for bulk encryption
- Public Key Systems are based on “hard” problems
 - Factoring large numbers, discrete logarithms, elliptic curves
- Only a handful of public key systems perform both encryption and signatures

Modular Arithmetic

- $a \bmod n = b$ if for some $k \geq 0$, $(k*n) + b = a$



- Associativity, Commutativity, and Distributivity hold in Modular Arithmetic
- Inverses also exist in modular arithmetic
 - For every a there exists another element (denoted -a) such that $a + (-a) \bmod n = 0$
 - For every a there exists another element (denoted a')
 - such that $(a*a') \bmod n = 1$
 - Note that - does not mean 'negative' in the usual sense

Modular Arithmetic

- Reducibility also holds
 - $(a + b) \bmod n = (a \bmod n + b \bmod n) \bmod n$
 - $(a * b) \bmod n = ((a \bmod n) * b \bmod n) \bmod n$
- Fermat's Thm: if p is any prime integer and a ($a < p$) is an integer, then $a^p \bmod p = a$
 - Corollary: $a^{p-1} \bmod p = 1$ if $a \neq 0$

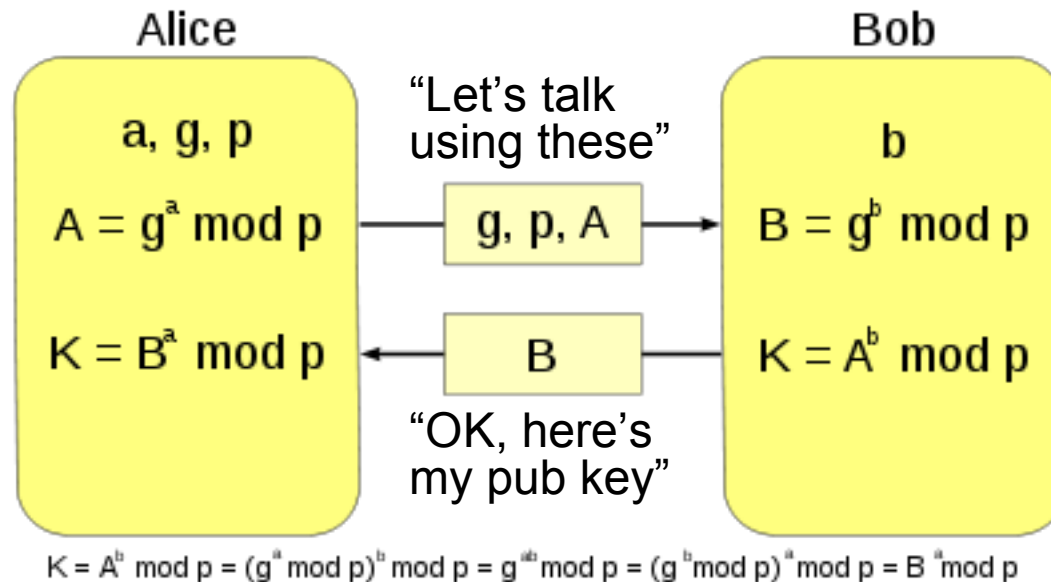
Diffie-Hellman

- The first public key cryptosystem proposed
 - [Crypto:How the Code Rebels Beat the Governemnt, Saving Privacy in the Digital Age...](#)”
- Usually used for exchanging keys securely
- Compute a common, shared key
 - Called a *symmetric key exchange protocol*
- Security based on discrete logarithm problem
 - Given integers n and g and prime number p , compute k such that $n = g^k \bmod p$
 - Solutions known for small p
 - Solutions computationally infeasible as p grows large

Algorithm

- Public Constants: prime p , integer $g \neq 0, 1, \text{ or } p-1$
- Choose private keys (lowercase k) and compute public keys (uppercase K)
 - Alice chooses private key k_{Alice} ,
computes public key $K_{Alice} = g^{k_{Alice}} \text{ mod } p$
 - Similarly Bob chooses k_{Bob} ,
computes $K_{Bob} = g^{k_{Bob}} \text{ mod } p$
- Exchange public keys and compute shared information
 - To communicate with Bob, Alice computes $k_{shared} = K_{Bob}^{k_{Alice}} \text{ mod } p$
 - To communicate with Alice, Bob computes $k_{shared} = K_{Alice}^{k_{Bob}} \text{ mod } p$

Diffie-Hellman Key Exchange



g is typically small, in practice $g=2$ or $g=5$

Alice and Bob compute the same shared key

K_{Bob} and K_{Alice} are viewable by anyone

Alice computes Bob's public key raised to her secret key, mod p

$$\begin{aligned} K_{Bob}^{k_{Alice}} \bmod p &= (g^{k_{Bob}} \bmod p)^{k_{Alice}} \bmod p \\ &= ((g^{k_{Bob}})^{k_{Alice}} \bmod p) \bmod p && \text{by law } a * b \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n \\ &= ((g^{k_{Bob} * k_{Alice}}) \bmod p) && (a^b)^c = a^{b * c} \\ &= ((g^{k_{Alice} * k_{Bob}}) \bmod p) \\ &= ((g^{k_{Alice}})^{k_{Bob}} \bmod p \\ &= ((g^{k_{Alice}})^{k_{Bob}} \bmod p) \bmod p \\ &= (g^{k_{Alice}} \bmod p)^{k_{Bob}} \bmod p \\ &= K_{Alice}^{k_{Bob}} \bmod p \end{aligned}$$

Which is what Bob computes --- Alice's public key raised to his secret key. Note $0 \leq key < p$

Both Bob and Alice now use this key for encryption, using some (other) algorithm

An attacker can see K_{Alice} , K_{Bob} , and messages encrypted using the shared key

Without knowledge of either k_{Alice} or k_{Bob} , he works exhaustively through all possible keys with values $0 \leq key < p$

DH also protects Alice from Bob, and vice versa. For Alice to discover k_{Bob} in equation

$$K_{Alice}^{k_{Bob}} \bmod p$$

She must solve the "discrete logarithm" problem. No efficient solutions known for large keys

Example : Computing Key

- Assume $p = 53$ and $g = 17$
- Alice chooses $k_{Alice} = 5$
 - Then $K_{Alice} = 17^5 \bmod 53 = 40$
- Bob chooses $k_{Bob} = 7$
 - Then $K_{Bob} = 17^7 \bmod 53 = 6$
- Shared key:
 - $K_{Bob}^{k_{Alice}} \bmod p = 6^5 \bmod 53 = 38$
 - $K_{Alice}^{k_{Bob}} \bmod p = 40^7 \bmod 53 = 38$

Real public DH values

- For IPsec and SSL, there are a small set of g's and p's published that all standard implementations support.
 - Group 1 and 2
 - <http://tools.ietf.org/html/rfc2409>
 - Group 5 and newer proposed values
 - <http://tools.ietf.org/html/draft-ietf-ipsec-ike-modp-groups-00>

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
 - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (e.g. 1024 bits)
- security due to cost of factoring large numbers into prime components

Background

- Totient function $\phi(n)$
 - Number of positive integers less than n and relatively prime to n
 - *Relatively prime* means with no factors in common with n
- Example: $\phi(10) = ?$
 - 4 because 1, 3, 7, 9 are relatively prime to 10
- Example: $\phi(p) = ?$ where p is a prime
 - $p-1$ because all lower numbers are relatively prime

Background

- Euler generalized Fermat's Thm for composite numbers.
 - Recall Fermat's Thm $(a^{p-1} \bmod p) = 1$ if $a \neq 0$
- Euler's Thm: $(x^{\phi(n)} \bmod n) = 1$
 - Where q and p are primes
 - $n = pq$
 - then $\phi(n) = (p-1)(q-1)$

RSA Algorithm

- Choose two large prime numbers p, q
 - Let $n = p * q$; then $\phi(n) = (p-1)(q-1)$
 - Choose $e < n$ such that e is relatively prime to $\phi(n)$.
 - Compute d such that $e * d \bmod \phi(n) = 1$
- Public key: (e, n) ; private key: d
- Encipher message ‘ m ’ : $c = m^e \bmod n$
- Decipher: $m = c^d \bmod n$

The RSA problem (how strong is RSA?)

- Given public key (e, n) and cipher text $c = m^e \bmod n$, find m
- $n = p * q$, both unknown, both primes
 - But suppose you could *factor* n quickly, and discover p and q
 - The RSA algorithm for computing d (the private portion of (e, n)) is
“ Compute d such that $e * d \bmod \phi(n) = 1$ “
and was executed using the same known set of inputs : n, p, q, e
- So by factoring n into p and q we can compute d , and thus use the RSA decryption formula to compute m

Working through the equations

$$C = F(m, e) = m^e \bmod n \quad (C \text{ is ciphertext})$$

Now decipher...

- $F(F(m, e), d)$ (given C , this is the decipher step)
 - $= (m^e \bmod n)^d \bmod n$ (symbol substitution)
 - $= m^{e*d} \bmod n$ (rules of modular arithmetic)
 - $e*d \bmod \phi(n) = 1$ (by construction of d)
 - $k*\phi(n) + 1 = e*d$ (k exists by definition of mod function)
 - $= (m^1 \bmod n * m^{k*\phi(n)} \bmod n) \bmod n$ (by substitution)
 - By Euler's theorem $X^{\phi(n)} \bmod n = 1$
 - $= m \bmod n$
 - $= m$

Example: Confidentiality

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
 - e is her public key, d is her private key
- Bob wants to send Alice secret message HELLO (07 04 11 11 14) --- encodes each character separately
 - $07^{17} \bmod 77 = 28$ (notice the encoding uses public key)
 - $04^{17} \bmod 77 = 16$
 - $11^{17} \bmod 77 = 44$
 - $11^{17} \bmod 77 = 44$
 - $14^{17} \bmod 77 = 42$
- Bob sends 28 16 44 44 42

Security Services

- Confidentiality
 - Only the owner of the private key knows it, so text enciphered with public key cannot be read by anyone except the owner of the private key
- Authentication
 - Only the owner of the private key knows it, so text enciphered with private key must have been generated by the owner

More Security Services

- Integrity
 - Enciphered letters cannot be changed undetectably without knowing private key
- Non-Repudiation
 - Message enciphered with private key came from someone who knew it

Example

- Alice receives 28 16 44 44 42
- Alice uses private key, $d = 53$, to decrypt message:
 - $28^{53} \bmod 77 = 07$
 - $16^{53} \bmod 77 = 04$
 - $44^{53} \bmod 77 = 11$
 - $44^{53} \bmod 77 = 11$
 - $42^{53} \bmod 77 = 14$
- Alice translates message to letters to read HELLO
 - No one else could read it, as only Alice knows her private key and that is needed for decryption

Example:

Integrity/Authentication

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
 - e is her public key, d is her private key
- Alice wants to send Bob message HELLO (07 04 11 11 14) so Bob knows it is what Alice sent (no changes in transit, and authenticated)
 - $07^{53} \bmod 77 = 35$ (the encryption uses **her private key**---authentication)
 - $04^{53} \bmod 77 = 09$
 - $11^{53} \bmod 77 = 44$
 - $11^{53} \bmod 77 = 44$
 - $14^{53} \bmod 77 = 49$
- Alice sends 35 09 44 44 49
- Notice that anyone can intercept this...can only Bob decode?

Example

- Bob receives 35 09 44 44 49
- Bob uses Alice's public key, $e = 17$, $n = 77$, to decrypt message:
 - $35^{17} \bmod 77 = 07$
 - $09^{17} \bmod 77 = 04$
 - $44^{17} \bmod 77 = 11$
 - $44^{17} \bmod 77 = 11$
 - $49^{17} \bmod 77 = 14$
- Bob translates message to letters to read HELLO
 - Alice sent it as only she knows her private key, so no one else could have enciphered it
 - If (enciphered) message's blocks (letters) altered in transit, would not decrypt properly
 - Coding is not tied to Bob----anyone could intercept and decode. But whoever does knows that the message had to have been encoded with Alice's private key

Example: Both

- Alice wants to send Bob message HELLO both enciphered and authenticated (integrity-checked)
 - Alice's keys: public (17, 77); private: 53
 - Bob's keys: public: (37, 77); private: 13
- Alice enciphers HELLO (07 04 11 11 14): authenticate **first**, code to recipient **last**
 - $(07^{53} \bmod 77)^{37} \bmod 77 = 07$
 - $(04^{53} \bmod 77)^{37} \bmod 77 = 37$
 - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
 - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
 - $(14^{53} \bmod 77)^{37} \bmod 77 = 14$
- Alice sends 07 37 44 44 14
- Bob needs additional information to know that Alice claims to have sent the message

Warnings

- Encipher message in blocks considerably larger than the examples here
 - If 1 character per block, RSA can be broken using statistical attacks (just like classical cryptosystems)
 - Attacker cannot alter letters, but can rearrange them and alter message meaning
 - Example: reverse enciphered message of text ON to get NO

Direct Digital Signature

- Involve only sender & receiver
- Assumed receiver has sender's public-key
- Digital signature made by sender signing entire message (or hash) with private-key
- Can encrypt using receivers public-key
- Important that sign first then encrypt message & signature
- Security depends on sender's private-key

Sign-Encrypt vs. Encrypt-Sign

- Is Sign-Encrypt Enough?
 - Recipient knows who wrote the message
 - But who encrypted it?
 - Surreptitious forwarding : signer and encrypter are different entities
- Does Encrypt-Sign make sense?
 - Signature can be easily replaced
 - RSA Signatures

Problems / Solutions

There is not a complete binding of sender/receiver to the message

- Naming repairs
 - Include Senders name
 - Include Recipients name
- Sign/Encrypt/Sign
 - Add recipient's name,
 - Sign to prove authorship
 - Encrypt for confidentiality
 - Sign (again) to prove authorship of encryption

Hash or Checksums

- Mathematical function to generate a set of k bits from a set of n bits (where $k \leq n$).
 - k is smaller than n except in unusual circumstances
- Example: ASCII parity bit
 - ASCII has 7 bits; 8th bit is “parity”
 - Even parity: even number of 1 bits
 - Odd parity: odd number of 1 bits

Example Use

- Bob receives “10111101” as bits.
 - Sender is using even parity; 6 1 bits, so character was received correctly
 - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
 - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Another Example

- 8-bit Cyclic Redundancy Check (CRC)
 - XOR all bytes in the file/message
 - Good for detecting accidental errors
 - But easy for malicious user to “fix up” to match altered message
- For example, change the 4th bit in one of the bytes
 - Fix up by flipping the 4th bit in the CRC
- Easy to find a M' that has the same CRC

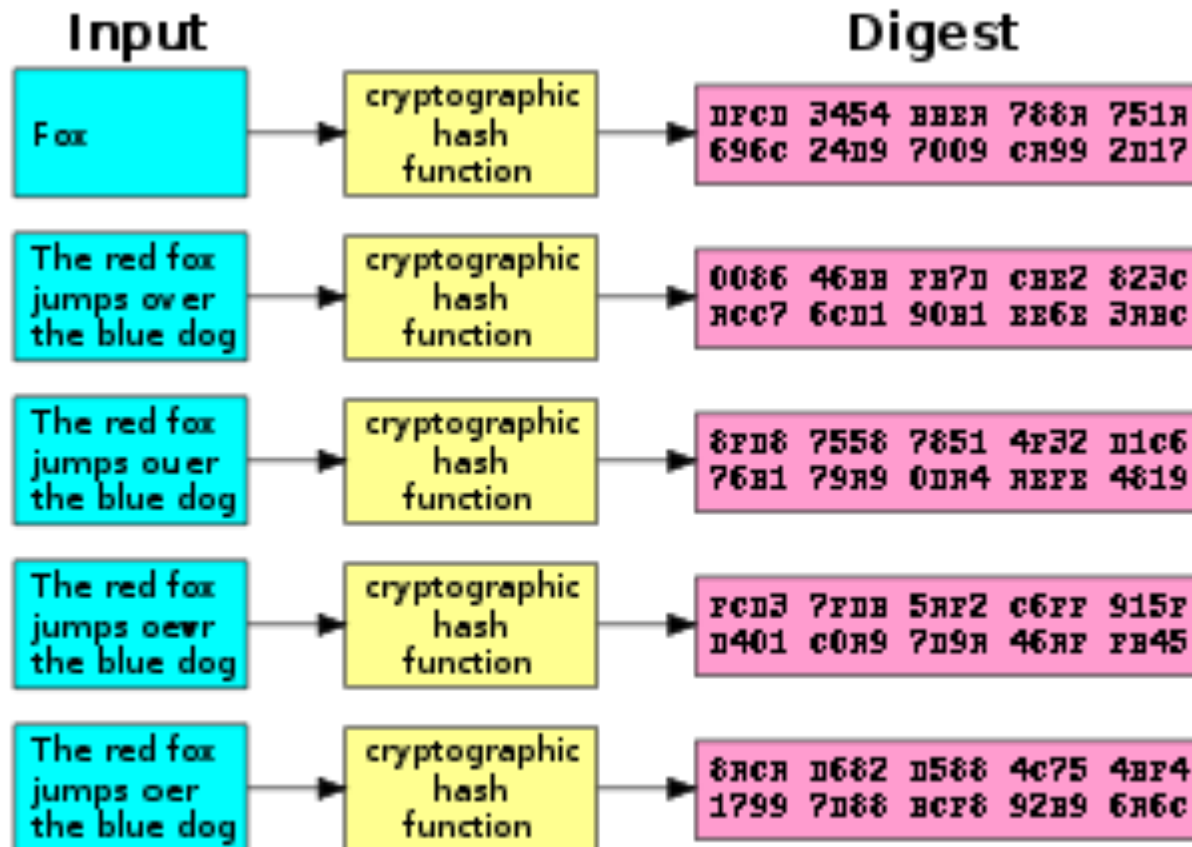
Cryptographic Hash or Checksum or Message Digest

- $h: A \rightarrow B$:
 - For any $x \in A$, $h(x)$ is easy to compute
 - For every $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$ (*pre-image attack*)
 - One way function
 - It is computationally infeasible to find two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$ (*collision resistance*)
 - Alternate form : **Given** any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Usually the case that input has arbitrary size, output has fixed size

Example

- SHA-1
 - Note fixed sized output
 - SHA-1 has avalanche effect



Collisions

- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
 - Pigeonhole principle: if there are n containers for $n+1$ objects, then at least one container will have 2 objects in it.
 - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files

Birthday Paradox

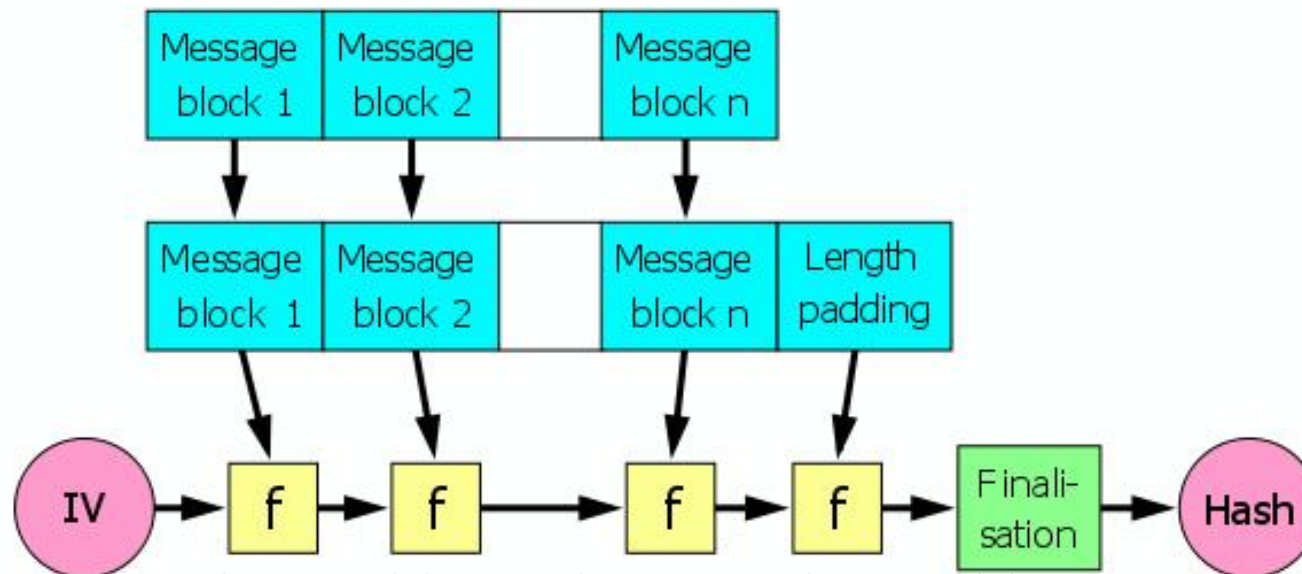
- What is the probability that someone in the room has the same birthday as me?
- What is the probability that two people in the room have the same birthday?
 - Think of it as each of n people randomly choosing numbers and requiring each draw to be unique
 - 1st draw unique from others with probability 1
 - 2nd draw unique from first with probability $(1-1/365)$
 - 3rd draw unique from first 2 with probability $(1-2/365)$
 - Etc
 - $\text{Prob}\{n \text{ unique draws}\} = 1 (1-1/365)(1- 2/356)(1 - 3/365) \dots (1- (n-1)/365)$
 - $\text{Prob}\{\text{at least 1 match}\} = 1 - (365!/(365^n*(365-n)!)$
 - Turns out that probability of at least one match is > 0.5 if $n \geq 23$.
 - In generalization of N outcomes, chance of a match in n tries exceeds 0.5 if n exceeds $1.17 \sqrt{N}$
 - http://en.wikipedia.org/wiki/Birthday_paradox

Another View of Collisions

- **Birthday attack** works thus:
 - m bits in hash function output
 - opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
 - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
 - two sets of messages are compared to find pair with same hash (probability close to 0.5 by birthday paradox)
 - have user sign the valid message, then substitute the forgery which will have a valid signature
- Need to use larger message authentication code (MAC)

MD5 and SHA

- Most widely used *keyless* crypto hashes
 - Based on Merkle/Damgård-hash construction
 - Message padded to be multiple of block size
 - Note chain-forward from compression function



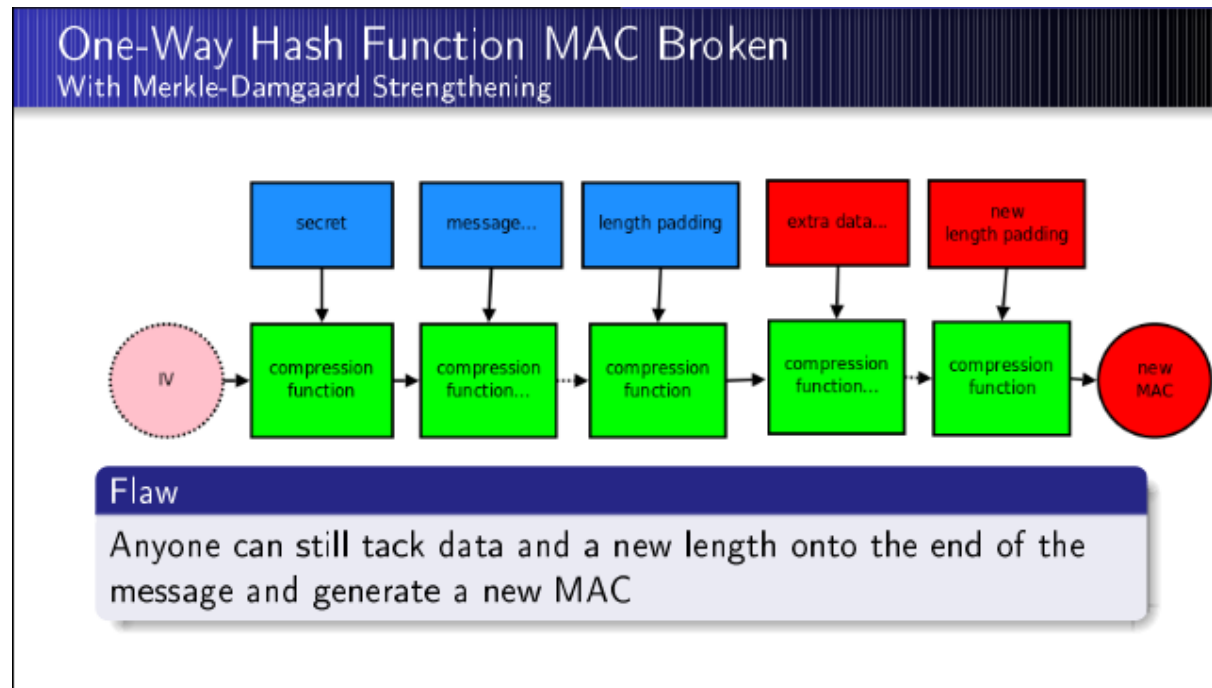
MD5 output is 128 bits and SHA-1 is 160 bits

Attack on Merkle/Damgård *Applications*

- Some ways of using hashes based on Merkle/Damgård-hash are vulnerable

Attacker has $h(m)$ and $\text{length}(m)$

- Choosing suitable m' attacker can calculate $h(m \parallel \text{pad}(m) \parallel m')$
- Attack involves intercepting message, appending to it, and changing hash of it



So What?

- Some simple authentication protocols
 - Establish a secret key K between parties (e.g. server and client)
 - Authenticate a message m by sending

$$h(K || m) || m$$

- Idea is that an attacker doesn't know K , so can't replace this with a different message and different hash
- Except he can...by appending m' to m , and send
$$h(K || m || \text{pad}(m) || m') || (m || \text{pad}(m) || m')$$
- Recipient authenticates $(m || \text{pad}(m) || m')$ *without knowing K !*
- [Attack](#) on Flickr API

Reversing key and plaintext doesn't help much for Merkle/Damgård hashing

- Establish a secret key K between parties (e.g. server and client)
- Authenticate a message m by sending

$$h(m \parallel K) \parallel m$$

- Suppose h is weak with respect to collisions
- Attacker finds m and m' with $h(m) = h(m')$. m is innocent, m' is not
 - Requests authenticator for m , i.e., receives $h(m \parallel K)$ but does not know K . $h(m' \parallel K) = h(m \parallel K)!!!$
 - Sends $h(m \parallel K) \parallel m'$. Receiver, knowing K , validates

MD5 Broken

- MD5 shown in 2007 to not be collision resistant
 - Researchers showed how to create a pair of files with same MD5 checksum
- In 2008 a group used this technique to fake SSL certificate validity
 - i.e., a real attack on real-world use of MD5
- US-CERT (in Dept. of Homeland Security) declared
 - “MD5 should be considered cryptographically broken and unsuitable for further use”
- Most U.S. government applications required to move to SHA-2 family

More on SHA

- Standard put forth by NIST
- SHA spec
 - <http://csrc.nist.gov/CryptoToolkit/tkhash.html>
- Comes in different flavors that vary based on output size
 - SHA-1 outputs 160 bits
 - “SHA-2” is a family of SHA-X
(X=224,256,384,512) is #output bits

SHA-1 Broken

- Chinese researchers had a break through (2005)
 - http://www.schneier.com/blog/archives/2005/02/sha1_broken.html
 - Recent results show that you can find collisions in 2^{69} attempts which would be less than 2^{80} from brute force
 - Does not affect HMAC-SHA
 - Jon Callas (CTO PGP)
 - "It's time to walk, but not run, to the fire exits. You don't see smoke, but the fire alarms have gone off."
- NIST published standards promoting using of larger SHA's
 - <http://csrc.nist.gov/CryptoToolkit/Hash.html>
 - Is currently running a competition for SHA-3
 - Final round to be announced 2010, winner 2012

Keyed vs Keyless Hash

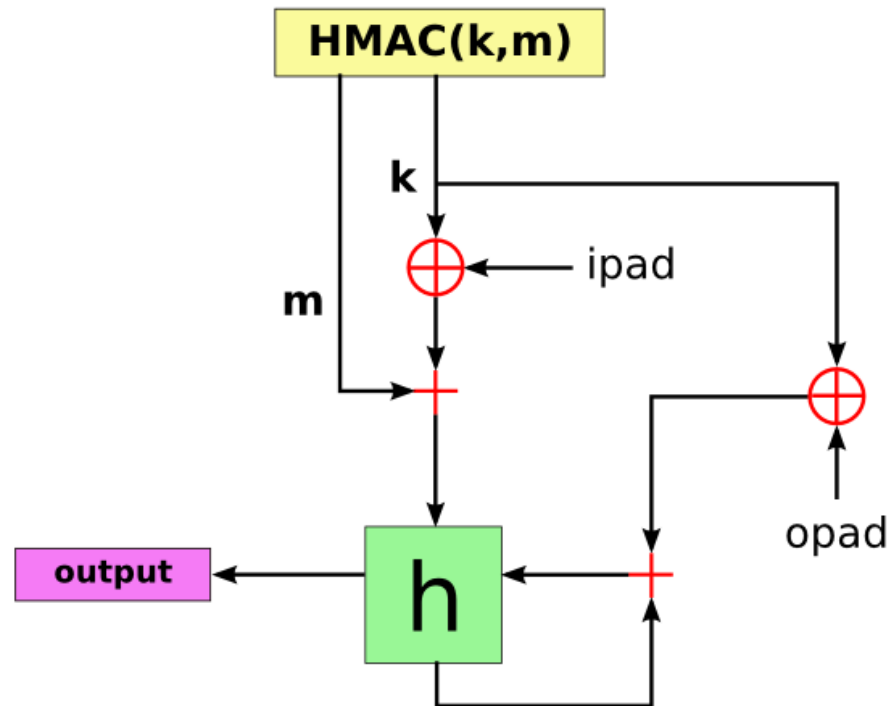
- Keyless hash – anyone can recompute given the message and indication of the algorithm
 - Example : check on software you download---hash and signed hash are posted
- Keyed hash – must have access to a key to recompute the hash

Message Authentication Codes

- MAC is a crypto hash that is a proof of a message's integrity
 - Important that adversary cannot fixup MAC if he changes message
- MAC's rely on keys to ensure integrity
 - Either the crypto hash is encrypted already
 - Or crypto hash must be augmented to take a key

HMAC Example

HMAC = “Hash-based Message Authentication Code”



$$\text{HMAC}_K(m) = h\left((K \oplus \text{opad}) \parallel h((K \oplus \text{ipad}) \parallel m) \right)$$

K is secret key known to sender and receiver

HMAC is immune to the earlier attacks

- Note difference between

$h(K || m)$ or $h(m || K)$ and

$h(K || h(K || m))$

- The “internal” hash step defeats concatenation and “external” collision attacks
- See [NIST document](#) specifying standard for HMAC
 - Note--- h is general, power of HMAC is in double hashing

Use Symmetric Ciphers for Keyed Hash

- In theory can use DES or AES in CBC mode
 - Last block is the hash
- DES with 64 bit block size is too small to be effective MAC
 - Birthday attack requires only $O(2^{\{32\}})$ samples be generated

Key Points

- Symmetric cryptosystems encipher and decipher using the same key
 - Or one key is easily derived from the other
- Public key cryptosystems encipher and decipher using different keys
 - Computationally infeasible to derive one from the other
- Cryptographic checksums provide a check on integrity