Public Key Cryptography

- Ciphers such as AES and DES are known as *conventional*, *symmetric* algorithms, or *secret key* algorithms
- In such algorithms, $K = K^{-1}$, i.e., the encryption key and the decryption key are the same
- In *public key* or *asymmetric* cryptography, $K \neq K^{-1}$. Furthermore, given K it is infeasible to find K^{-1}

The History of Public Key Cryptography

- Generally credited to Diffie and Hellman's paper "New Directions in Cryptography" (1976)
- Remarkable paper — created the academic field of cryptography
- However — public key crypto was actually invented by the British in 1970, under the name "Non-Secret Encryption"
- Some claim that it was actually invented by the Americans in the mid-1960s to control nuclear weapons
- See the reading list for today

The Purpose of Public Key Cryptography

- If Alice and Bob want to exchange secret messages, they first have to share a key
- What if they've never met?
- What if they have exchanged keys, but run out?
- Key-handling is hard

Key-Handling

. . . the judge asked the prosecution's expert witness: "Why is it necessary to destroy yesterday's . . . [key] . . . list if it's never going to be used again?" The witness responded in shock: A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications."

The Problem of Key-Handling

- Reusing keys is dangerous — many cryptanalytic attacks work by looking for key reuse
- Friedman's "Index of Coincidence" detects overlap from just the ciphertext of conventional ciphers.
- One of the ways Enigma was attacked: the British captured a German weather observation ship that had the next several months of keys
  ☞ Note the other mistake: putting general-purpose keys in a vulnerable place
- The "Venona" project: the U.S. read years of Soviet communications when they discovered that the Soviets had reused one-time pads

One-Time Pads

- As noted last time for stream ciphers, must never be reused
- Producing so much true-random keying material is a strain
- During war-time, the Soviets couldn't keep up
- *Sometimes* usable for point-to-point communication
- Doesn't work well in groups: n^2 keying problem. Worse yet, every set of keys for a one-time pad must be long enough to handle the maximum length of messages you'll ever send
- Theoretically unbreakable but practically useless

The Solution: Public-Key Cryptography

- Alice publishes her *encryption* key K
- This isn't secret; anyone can know it
- Glaring example: the Mossad—Israel's Secret Intelligence Service—has a web page you can use to talk to them. The server uses public key cryptography

A First Approximation

- Alice has a public key K_A, which she publishes, and a private key K_A^{-1}, which she keeps secret
- Bob wants to send her a message M
- Bob looks up her key and sends $\{M\}_{K_A}$
- Alice uses K_A^{-1} to calculate $\{\{M\}_{K_A}\}_{K_A{}^{-1}} = M$

That’s Too Expensive

- All known public key algorithms are far more expensive than symmetric algorithms
- The most common ones rely on exponentiation of very large numbers
- New ones (*elliptic curve cryptography*) is cheaper, but still expensive

A Better (But Not Good) Approach

- Alice has a public key K_A, which she publishes, and a private key K_A^{-1}, which she keeps secret
- Bob wants to send her a message M
- Bob looks up her key
- Bob generates a random symmetric *session key* K_S and sends $\{K_S\}_{K_A}$, $\{M\}_{K_S}$
- That is, you use public key cryptography *only* to encrypt the session key. The session key is used for all bulk data.
- Alice uses K_A^{-1} to calculate $\{\{K_S\}_{K_A}\}_{K_A{}^{-1}} = K_S$
- Alice uses K_S to calculate $\{\{M\}_{K_S}\}_{K_S{}^{-1}} = M$

Why Isn't it Good?

- Bob doesn't know who sent the message
- Bob doesn't know that K_S is *fresh*, i.e., not previously used
- (Actually doing public key encryption is tricky)

RSA

- Pick two large primes, p and q
- Let $n = pq$
- Pick two keys, e and d, such that $ed \equiv 1 \bmod (p-1)(q-1)$
- e is the encryption (or *public*) key; d is the decryption (or *private*) key
- Encryption: $C \equiv M^e \bmod n$
- Decryption: $M \equiv C^d \bmod n$
- That is, $(M^e)^d \equiv M \bmod n$
- Strength rests on difficulty of factoring n

Huh?

- Remarkably, checking the primality of a large number can be done efficiently
- However, there are no known efficient algorithms for factoring large numbers
- For efficiency, usually $e = 3$
- Given e, p, q, calcuating d is easy via Euclid's Algorithm
- If we could factor n, it is therefore easy to find d
- It is unknown if there is a way to recover d without factoring n
- All of this follows from (reasonably) elementary number theory

Turning it Around

- What if we *encrypt* with d?
- Why not? The equations are symmetric
- Only the possesor of the private key d can calculate $M^d \bmod n$
- But e is public, so anyone can calculate $(M^d)^e \bmod n \equiv M$
- This is known as a *digital signature*

Digital Signatures

- Only the key owner can calculate them
- Anyone can verify them
- Any change to the message will result in a different signature value

History of Digital Signatures

- The British did not invent digital signatures, only public key encryption
- There is reason to suspect that the Americans invented digital signatures but not public key encryption
- Diffie and Hellman invented both, but failed in an attempt to design suitable algorithms
- They came agonizingly close — they had the equation, but with a prime modulus
- It took Rivest, Shamir, and Adleman to solve both problems

Non-Repudiation

- Digital signatures provide *non-repudiation*
- “protection against false denial of involvement in a communication” [RFC 2828]
- Since anyone can verify the signature, a judge can, too

Digital versus Physical Signatures

- Physical signatures are strongly bound to the signer, and weakly bound to the message
- Digital signatures are strongly bound to the message, and weakly bound to the signer
- What if the private key leaks? What if the signer *deliberately* leaks the private key, to provide deniability?

Large Primes

- How large is “large”?
- Today, people commonly use 1024-bit moduli
- There are published designs for a \$1,000,000 machine that can factor a 1024-bit key in a year
- As far as is known, no one has built such a thing, but. . .
- How long must the information remain secret? How long must a digital signature be verifiable? Mortgages commonly last for 30 years
- Prudence suggests 2048 or 3072-bit keys

The RSA Challenge

- A challenge encryption appeared in Scientific American in 1977
- The modulus was 129 digits, or 429 bits
- A large distributed effort solved in in 1993:
  THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

Actually Using RSA

- There are many traps here, both obvious and subtle
- Example: let “yes” $= 1$, “no” $= 0$
- Encrypt your answer with RSA
- Oops. . .
- Must use mathematically sound padding. (Possible approach: Encrypt 1023 random bits, plus one bit of message)

Timing Attacks

- 1-bits in the exponent take longer than 0-bits (can shift over the 0-bits)
- By having your target decrypt suitable RSA messages, you can learn where the 1-bits are
- Implemented in 2003 by Boneh and Brumley against web servers

Common Objections

- The NSA can factor RSA moduli
- Who knows? But they use RSA, too. Besides, factoring has been a subject of mathematical attention for > 350 years
- The NSA can build a catalog of primes
- By the Prime Number Theorem, there are $\approx n/\log n$ primes less than n. For 512-bit p and q, that is about 10^{151}. Even NSA doesn't have that much disk space.
- It's magic and can't work…

I Cheated

- For encryption, I said “use symmetric algorithms; use RSA for the session key”
- For digital signatures, I said “sign the message”
- It’s still too expensive to do that
- We need *cryptographic hash functions*
- We sign $H(M)$, not M

Cryptogrpaphic Hash Functions

- Must be reasonably cheap
- Must take an arbitrary-length message and produce a fixed-length output
- Must be impossible to forge signatures by attacking the hash function

Properties of Cryptogrpaphic Hash Functions

Collision resistance It is computationally infeasible to find $x, y, x \neq y$ such that $H(x) = H(y)$

Preimage resistance Given an output value y, it is computationally infeasible to find x such that $H(x) = y$

Second preimage resistance Given an input x, it is computationally infeasible to find x' such that $H(x) = H(x')$

Hash Function Failures

- Second preimage resistance: forge a new document or message to match any hash
- Preimage resistance: similar, but you don't get to see the input message
- Collision: trick someone into signing one document; show the other to the judge — see `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions`

Modern Hash Functions

- MD5 (128 bits) — Invented by Rivest
- SHA-1 (160 bits) — Invented by NSA; standardized by NIST
- ☞ SHA-0 wasn't as strong as it should have been; NSA made a mistake
- SHA-256, SHA-384, SHA-512 — Stronger variants of SHA-1
- Other, less common ones: RIPEMD160 (160-bit), Whirlpool (512 bits)

Status

- Only MD5 and SHA-1 are widely used
- SHA-256, SHA-384, SHA-512 are stronger (and slower) variants
- Last year, a collision-finding algorithm for MD5 was published by Wang et al.
- This year, she showed that SHA-1 is much weaker than it should be
- Can we switch? Should we?

Switching Hash Functions

- Do we need to switch now?
- Not quite — for many purposes, collision-resistance isn't crucial
- We should immediately stop using MD5 for secure email
- But we can't convert to anything stronger than SHA-1 — no one supports it, and the network protocols weren't properly designed for upgrades
- There is as yet no agreement on what hash function to switch to

Other Important Algorithms

- Diffie-Hellman — used for key management
- Relies for its strength on the *discrete logarithm* problem: Given a and $a^b \bmod p$, it is infeasible for find b
- DSA (Digital Signature Algorithm) — U.S. government standard for digital signatures; cannot be used for encryption
- Based on discrete log

Algorithm Strengths

Hash functions need to have output twice as long as the symmetric key size for proper collision resistance

Symmetric Key Size	Hash Output Size	RSA or DH Modulus Size
70	140	947
80	160	1228
90	180	1553
100	200	1926
150	300	4575
200	400	8719
250	500	14596

(Source: RFC 3766)

Sizes based on estimated computational equivalence

Cost of Increasing Modulus Size

For RSA, doubling the modulus length increases encryption time by $\sim 4\times$ and increases decryption time by $\sim 8\times$.

Modulus	CPU Time
256	1.5 ms
512	8.6
1024	55.4
2048	387.

(Source: RFC 3766)

Tests run years ago, on a 350 Mhz machine