Elliptic Curve Cryptography

- Public key and D-H algorithms, but based on more complex math
- Considerably more security per key bit; allows for shorter keys
- More importantly, allows for much more efficient computation
- Recently endorsed by NSA
- Many patents in this space

Moore's Law and Public Key Cryptography

- For RSA, doubling the modulus length increases encryption time by $4\times$ and decryption time by $8\times$
- Attack time against RSA is based on factoring algorithms, not brute force: there are far too many possible primes for brute force to be ever be possible
- For number field sieve, complexity is approximately proportional to

$$.02e^{1.92\sqrt[3]{\ln n \cdot (\ln \ln n)^2}}$$

- Sub-linear, but space complexity goes up much faster
- There is a paper design for a $10M machine (TWIRL) to factor a single 1024-bit number in one year

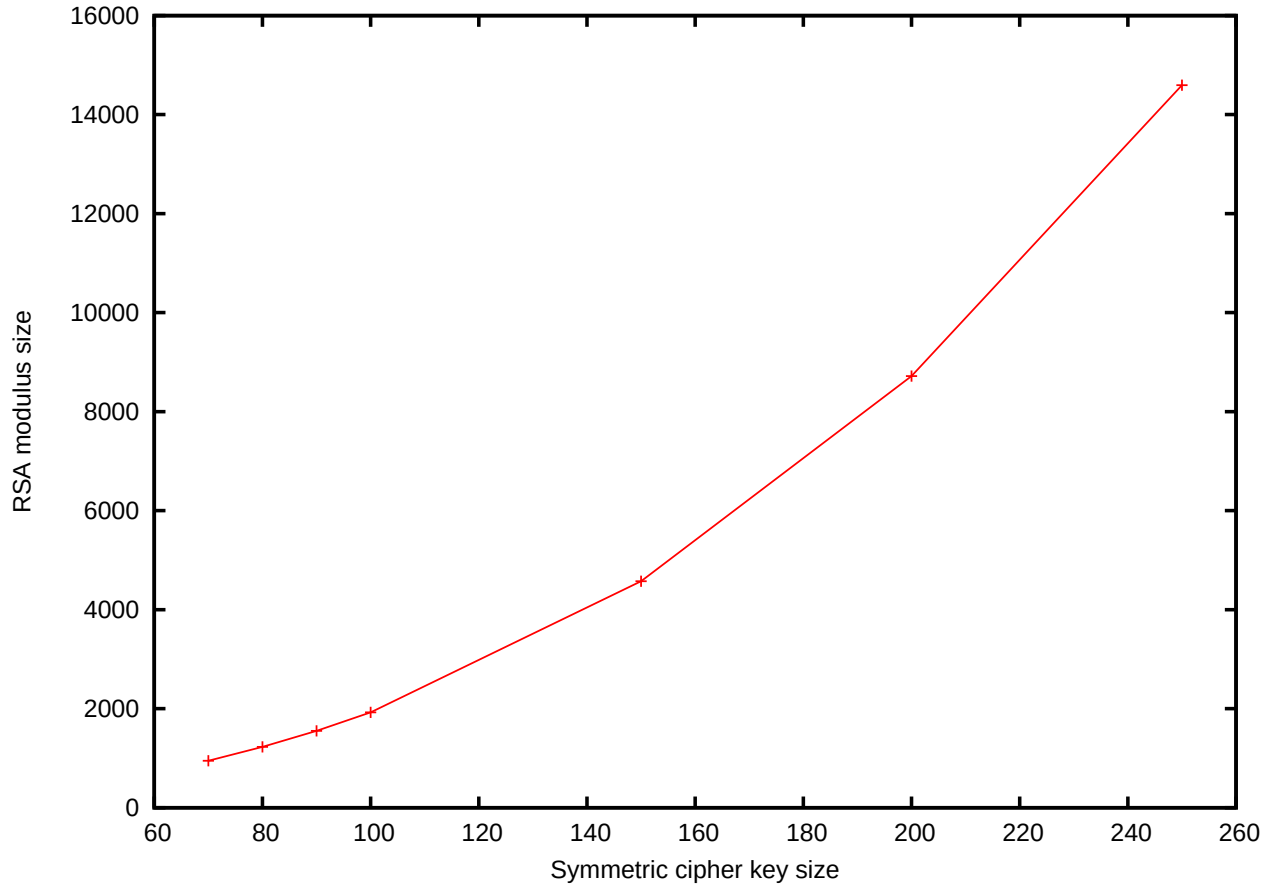
Rough Table of Key Length Equivalences

Symmetric Key Size (bits)	*RSA or DH Modulus Size (bits)*
70	947
80	1228
90	1553
100	1926
150	4575
200	8719
250	14596

Add 11 bits to the public key size if TWIRL can be built

(Numbers by Orman and Hoffman, RFC 3766)

Public versus Symmetric Key Sizes



Message Integrity

Message Integrity

- We need a way to prevent cut-and-paste attacks
- We can use a key and a cryptographic hash to generate a *Message Authentication Code* (MAC).
- Simpler solutions don't work
- One bad idea: append a cryptgraphic hash to some plaintext, and encrypt the whole thing with, say, CBC mode

$$\{P, H(P)\}_K$$

- This can fall victim to a *chosen plaintext attack*

Chosen Plaintext Attack

- The enemy picks some plaintext P and tricks you into encrypting it
- This has happened in the real world!
- You transmit

$$\{\text{prefix}, P, \text{suffix}, H(\text{prefix}, P, \text{suffix})\}_K$$

- But P is of the form

$$\text{prefix}, P', \text{suffix}, H(\text{prefix}, P', \text{suffix})$$

- An ordinary CBC subset will have the checksum!

HMAC

- Build a MAC from a cryptographic hash function
- Best-known construct is HMAC — provably secure under minimal assumptions
- HMAC$(m, k) = H(k, H(k, m))$ where H is a cryptographic hash function
- Note: authentication key *must* be distinct from the confidentiality key
- Frequently, the output of HMAC is truncated

CBC MAC

- Recall that for CBC encryption, the last block of ciphertext depends on all of the plaintext
- Do a second encryption (using a different key), but only send the last block
- If you use the same key, a CBC cut-and-paste attack will work
- Query: what IV should you use?

Order of Encryption and MACing

- If we want to encrypt and MAC a message, what order do we do it in?
- Three choices:

$$\{P\}_K, M_{K'}(P)$$

$$\{P, M_{K'}(P)\}_K$$

$$\{P\}_K, M_{K'}(\{P\}_K)$$

- The last is the most secure (provably so) — always calculate a MAC on the ciphertext
- Besides, since MACs are often cheaper than decryption, we can verify the integrity of ciphertext first, and discard the message if bogus

What to MAC?

- Obviously, the MAC includes all of the ciphertext
- Frequently, the MAC should include plaintext metadata
- Example: suppose you supply a plaintext message length, in plaintext, to go along with CBC encryption of a padded message
- The MAC should cover that length

Key Lifetimes

- A confidentiality key is useful as long as the data is sensitive; that may be many years
- A digital signature private key is useful as long as you need to prove authorship — think of a digitally-signed, 30-year mortgage
- A MAC key is useful only while the session is alive; once the session is over, the key is useless

Authentication

With Whom Are You Communicating?

- Identification — who they claim to be
- Authentication — proof
- Authorization — what they can do

Current Focus

- Authentication
- Network-oriented (biometrics aren't much good over the net)
- Many different styles

Authentication

- Something you know (i.e., passwords)
- Something you have (i.e., some sort of token or smart card)
- Something you are (biometrics)

Shared Secrets

- Can be something you know or something you have
- Password
- Cryptographic key

Passwords

- Very guessable — 10-40%, according to several studies
- Forgettable — people write them down (though that's generally *not* a networked threat)
- Susceptible to eavesdropping on the network (often called *sniffing*, though that's actually the brand name of a very reputable network diagnostic device)
- Can be given away, social-engineered, phished, etc.

Guessing Passwords

- Explained well by Morris and Thompson (see the reading)
- Basic strategy: start with a good dictionary
- Match your dictionary to the target population: science fiction terms, other languages, etc.
- Include names: boyfriends, girlfriends, spouses, pets, etc.
- Try variations on this word list: add a period or a digit, change “o” to “0”, etc.

Eavesdropping

- Trivial on wireless networks
- Not hard on wired Ethernets, either, especially if the switch misbehaves (or can be induced to misbehave)
- Active attack: send out fake ARP responses, to direct traffic to you
- Best defense: encryption
- Some people use various one-time passsword schemes

Tokens

- Generally have an embedded shared secret used as a cryptographic key
- Time-based: encrypted the time of day (i.e., RSA's SecurID)
- Challenge/response: server sends a random number; token encrypts it
- These are *one-time passwords* — never reused
- (Database is updated to prevent immediate reuse with SecurID)
- Frequently used together with a PIN, to guard against device loss or theft
- Other advantages beyond eavesdropping protection: can't be social-engineered; if lent out, the authorized owner can't use them
- But — someone put his on a webcam: `http://fob.webhop.net/`

A SecurID Token

Active Attacks

- Simple one-time passwords aren't enough against active attackers
- Two attacks (depending on attacker's powers): last-digit guessing attack and connection hijacking

Connection Hijacking

- Use ARP-spoofing or equivalent to route traffic through you
- Wait for victim to log in
- Imitate victim; don't let packets through to real user
- Demonstrated in the lab in 1995; now doable with off-the-shelf hacker code
- Defense: cryptographically protect *all* packets

Last-Digit Guessing Attacks

- Watch user start to log in with SecurID (or equivalent)
- Open up ten simultaneous login sessions; as the user types each digit of the authenticator, repeat that on each of the new login sessions
- Before the user types the last digit, guess at each possibility, once per new session
- The attacker will win. . .
- Defense (other than crypto): lock the authentication database entry for that user after the login name is entered, rather than when the authenticator is received

Cryptography and Authentication

- Some way to use a cryptographic key to prove who you are
- (Much more on that next class)
- Can go beyond simple schemes given above
- Can use symmetric or public key schemes
- Most public key schemes use *certificates*

What are Certificates

- How does Alice get Bob's public key?
- What if the enemy tampers with the phone book? Sends the phone company a false change-of-key notice? Interferes with Alice's query to the phone book server?
- A certificate is a digitally-signed message containing an identity and a public key — prevents tampering.

Why Trust a Certificate?

- Who signed it? Why do you trust them?
- How do you know the public key of the *Certificate Authority* (CA)?
- Some public key (known as the *trust anchor*) must be provided out-of-band — trust has to start somewhere.

Certificate Authorities

- Who picks CAs? No one and every one.
- Your browser has some CAs built-in — because the CA paid the browser vendor enough money. Is that grounds for trust?
- Matt Blaze: “A commercial certificate authority can be trusted to protect you from anyone from whom they won’t take money.”

Who Gets Certificates?

- How do you prove your identity to a CA?
- How good a job do they do verifying it?
- What warranties does the CA give if someone is fooled? (Most disclaim all liability. . .)

Another Trust Model

- Get certificates from parties whom you know.
- Issue certificates to your own users: *authorization certificates*
- Don’t rely on commercial identity-based CAs.

Certificate Hierarchy versus Web of Trust

- Most CAs are tree-structured
- Top-level CAs can use *bridge CAs* to cross-certify each other
- PGP uses a different style: a *web of trust*.
- Certificate signings can form an arbitrarily-complex graph — users can verify path to as many trust anchors as they wish.

Styles of Certification

- At least 3 major styles
- X.509/PKIX — traditional hierarchical CA (but can have “pki” instead of “PKI”)
- SPKI/SDSI — authorization certificates
- PGP web of trust (primarily for email)

What Else is in a Certificate?

- Technical information, such as algorithm identifiers
- More identification information — company, location, etc.
- Expiration date
- Logos
- Certificate role

Some Local Certificates

- CS dept web certificate at `http://www.cs.columbia.edu/~smb/classes/f06/cs-cert.txt`
- University web certificate at `http://www.cs.columbia.edu/~smb/classes/f06/cu-cert.txt`
- The CS department certificate uses MD5 — not good
- The CUIT certificate expires on October 7 — will they renew it in time?

Things to Notice About Certificates

- Signer (the university didn't issue the department's certificate)
- Validity dates
- Algorithms (RSA, SHA1, MD5)
- Certificate usage — encryption and authentication, but *not* for issuing other certificates
- Certificate Revocation List (CRL)

Not All Certificates are Alike

- An email certificate isn't the same as an ecommerce certificate.
- A CA certificate is even more different — can I use my att.com email certificate to issue more att.com certificates?
- What about a code-signing certificate for ActiveX?
- Usage-specific information, such as IP address range
- The certificate type is listed in the certificate. Beyond that, end systems can apply their own policies, i.e., don't accept code-signing certificates from `www.evilhackerdudez.org`

Revoking Certificates

- Keys associated with certificates can be compromised
- One choice – *certificate revocation list* (CRL)
- Can get large, which is one reason why certificates expire
- For connected hosts, possible to do online certificate status checking
- Can the attacker block connectivity to the status server?
- CRLs are the weak link of public key cryptography.