CSci 5271
Introduction to Computer Security
Day 15: Cryptography part 2: public-key

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

Public-key crypto basics

Announcements

Public key encryption and signatures

Pre-history of public-key crypto

- First invented in secret at GCHQ
- Proposed by Ralph Merkle for UC Berkeley grad. security class project
  - First attempt only barely practical
  - Professor didn't like it
- Merkle then found more sympathetic Stanford collaborators named Diffie and Hellman

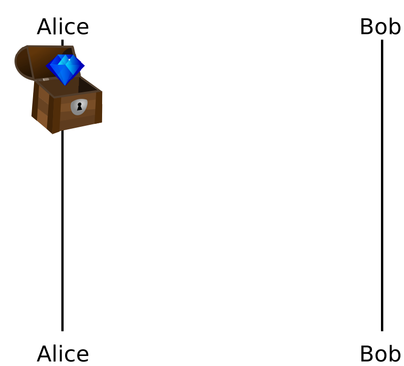
Box and locks analogy

- Alice wants to send Bob a gift in a locked box
  - They don't share a key
  - Can't send key separately, don't trust UPS
  - Box locked by Alice can't be opened by Bob, or vice-versa

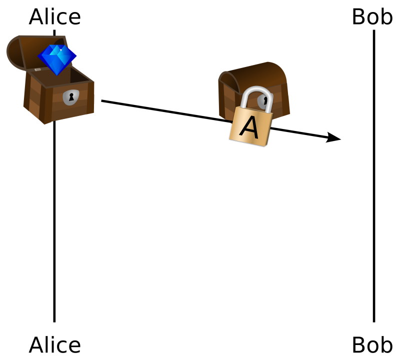
Box and locks analogy

- Alice wants to send Bob a gift in a locked box
  - They don't share a key
  - Can't send key separately, don't trust UPS
  - Box locked by Alice can't be opened by Bob, or vice-versa
- Math perspective: physical locks commute

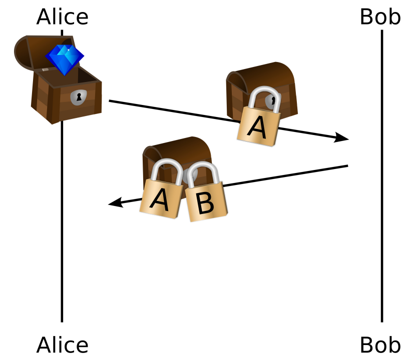
Protocol with clip art



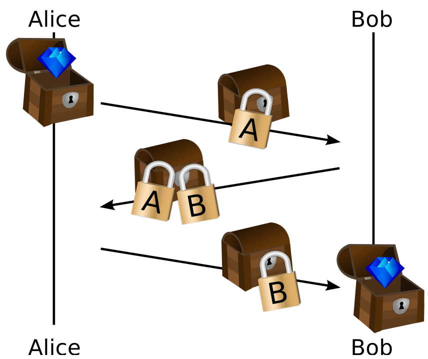
Protocol with clip art



Protocol with clip art



Protocol with clip art



Public key primitives

- Public-key encryption (generalizes block cipher)
  - Separate encryption key EK (public) and decryption key DK (secret)
- Signature scheme (generalizes MAC)
  - Separate signing key SK (secret) and verification key VK (public)

Modular arithmetic

- Fix *modulus* n, keep only remainders mod n
  - mod 12: clock face; mod 2^{32}: `unsigned int`
- $+$, $-$, and $\times$ work mostly the same
- Division: see Exercise Set 1
- Exponentiation: efficient by square and multiply

Generators and discrete log

- Modulo a prime p, non-zero values and $\times$ have a nice ("group") structure
- g is a *generator* if $g^0, g, g^2, g^3, \ldots$ cover all elements
- Easy to compute $x \mapsto g^x$
- Inverse, *discrete logarithm*, hard for large p

Diffie-Hellman key exchange

- Goal: anonymous key exchange
- Public parameters p, g; Alice and Bob have resp. secrets a, b
- Alice→Bob: $A = g^a \pmod{p}$
- Bob→Alice: $B = g^b \pmod{p}$
- Alice computes $B^a = g^{ba} = k$
- Bob computes $A^b = g^{ab} = k$

Relationship to a hard problem

- We're not sure discrete log is hard (likely not even NP-complete), but it's been unsolved for a long time
- If discrete log is easy (e.g., in P), DH is insecure
- Converse might not be true: DH might have other problems

Categorizing assumptions

- Math assumptions unavoidable, but can categorize
- E.g., build more complex scheme, shows it's "as secure" as DH because it has the same underlying assumption
- Commonly "decisional" (DDH) and "computational" (CDH) variants

Key size, elliptic curves

- Need key sizes ~10 times larger then security level
  - Attacks shown up to about 768 bits
- Elliptic curves: objects from higher math with analogous group structure
  - (Only tenuously connected to ellipses)
- Elliptic curve algorithms have smaller keys, about $2\times$ security level

Outline

Note to early readers

- This is the section of the slides most likely to change in the final version
- If class has already happened, make sure you have the latest slides for announcements

Outline

Public-key crypto basics

Announcements

Public key encryption and signatures

General description

- Public-key encryption (generalizes block cipher)
  - Separate encryption key EK (public) and decryption key DK (secret)
- Signature scheme (generalizes MAC)
  - Separate signing key SK (secret) and verification key VK (public)

RSA setup

- Choose $n = pq$, product of two large primes, as modulus
- n is public, but p and q are secret
- Compute encryption and decryption exponents e and d such that

$$M^{ed} = M \pmod{n}$$

RSA encryption

- Public key is (n, e)
- Encryption of M is $C = M^e \pmod{n}$
- Private key is (n, d)
- Decryption of C is $C^d = M^{ed} = M \pmod{n}$

RSA signature

- Signing key is (n, d)
- Signature of M is $S = M^d \pmod{n}$
- Verification key is (n, e)
- Check signature by $S^e = M^{de} = M \pmod{n}$
- Note: symmetry is a nice feature of RSA, not shared by other systems

RSA and factoring

- We're not sure factoring is hard (likely not even NP-complete), but it's been unsolved for a long time
- If factoring is easy (e.g., in P), RSA is insecure
- Converse might not be true: RSA might have other problems

Homomorphism

- Multiply RSA ciphertexts ⇒ multiply plaintexts
- This *homomorphism* is useful for some interesting applications
- Even more powerful: fully homomorphic encryption (e.g., both $+$ and $\times$)
  - First demonstrated in 2009; still very inefficient

Problems with vanilla RSA

- Homomorphism leads to chosen-ciphertext attacks
- If message and e are both small compared to n, can compute $M^{1/e}$ over the integers
- Many more complex attacks too

Hybrid encryption

- Public-key operations are slow
- In practice, use them just to set up symmetric session keys

\+ Only pay RSA costs at setup time

– Breaks at either level are fatal

Padding, try #1

- Need to expand message (e.g., AES key) size to match modulus
- PKCS#1 v. 1.5 scheme: prepend 00 01 FF FF .. FF
- Surprising discovery (Bleichenbacher'98): allows adaptive chosen ciphertext attacks on SSL

Modern "padding"

- Much more complicated encoding schemes using hashing, random salts, Feistel-like structures, etc.
- Common examples: OAEP for encryption, PSS for signing
- Progress driven largely by improvement in random oracle proofs

Simpler padding alternative

- "Key encapsulation mechanism" (KEM)
- For common case of public-key crypto used for symmetric-key setup
  - Also applies to DH
- Choose RSA message r at random mod n, symmetric key is $H(r)$

– Hard to retrofit, RSA-KEM insecure if e and r reused with different n

Box and locks revisited

- Alice and Bob's box scheme fails if an intermediary can set up two sets of boxes
  - Man-in-the-middle (or middleperson) attack
- Real world analogue: challenges of protocol design and public key distribution

Next time

- Building crypto into more complex protocols