Public Key Cryptography

Terminology

- Asymmetric cryptography
- Public key (known to entire world)
- Private key (not secret key)
- Encryption process (P to C with public key)
- Decryption Process (C to P with private key)
- Digital signature (P signed with private key)
  - Only holder of private key can sign, so can't be forged
  - But, can be recognized!

Uses

- Orders of magnitude slower than symmetric key crypto, so usually used to initiate symmetric key session
- Much easier to configure, so used widely in network protocols to establish temporary shared key that is used to transmit secret (symmetric) key

Uses

- Transmitting over insecure channel
- Alice <A_{pu}, A_{pr}> , Bob <B_{pu}, B_{pr}>
- Alice to Bob encrypt m with B_{pu}
- Bob to alice encrypt m with A_{pu}
- *Accurately knowing public key of other person is one of biggest challenges of using public key crypto.*

Uses

- Secure storage on insecure media
  - Encrypt not whole file, but a randomly generated secret key with public key. Then encrypt file using secret key.
  - Note if lose private key, you're out of luck. To backup, encrypt secret key with public key of a trusted friend (lawyer).
- Important advantage: Alice can enrypt a message for Bob without knowing Bob's decryption key

Uses

- Authentication
  - If Bob wants to prove his identity with symmetric key crypto, he needs a different symmetric key shared with each potential correspondent (otherwise friends can impersonate him)
  - Alice can verify she's talking to Bob (assuming she knows his public key) by sending a message r to Bob encrypted with Bob's public key. Bob sends back the cleartext message r (which only he could have decrypted).
  - Note Alice need not keep any secret information in order to verify Bob. (Unlike symmetric key crypto, in which a backup tape with a copy of the symmetric key might be used to impersonate Bob)

Uses

- Digital Signatures: prove message generated by particular individual
  - "Forged in USA" (engraved on screwdriver claiming to be of brand Craftsman)
  - If Bob encrypts a message with his private key, this proves both
    - Bob generated the message
    - The message has not been modified (to do so requires Bob's private key)
      - Without Bob's private key, creating ciphertext that decrypts to something meaningful with Bob's public key is computationally infeasible

Uses

- Digital Signatures: prove message generated by particular individual
  - Non-repudiation: Bob cannot deny having generated the message, since Alice could not have generated the proper signature without knowledge of Bob's private key.
  - Note that this can't be done with symmetric key. If Bob tries to claim he didn't send the message, Alice would know he's lying (because no one but herself and Bob would have the secret key), but Alice could not prove this to anyone else (since she herself could have generated the authentication code).

Modular Arithmetic

- Addition
  - Can be used as scheme to encrypt digits, since it maps each digit to different digit in a reversible way (decryption is addition by additive inverse)
    - Actually a Caeser cipher (and not good)
- Multiplication
  - Look at mod 10. Multiplication by 1,3,7, or 9 works, but not any of the others. Decryption done by multiplying by multiplicative inverse.
  - Multiplicative inverses can be found by using Euclid's Algorithm. Given x and n, Euclid's algorithm finds y such that $xy = 1 \bmod n$ (if there is such a y)

Modular Arithmetic

- Why 1,3,7,9? These are the numbers that are relatively prime to 10. All numbers that are relatively prime to 10 will have multiplicative inverses, others won't (so we can use these as ciphers, though not good ones).

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

x^y	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

Totient Function

- Allegedly from total and quotient
- How many numbers less than n are relatively prime to n?
- Totient function, $\phi(n)$ gives this.
- If n is prime, $\phi(n) = n-1$ (1,2,…n-1)
- If p and q are prime, $\phi(pq) = (p-1)(q-1)$
  - p, 2p, … (q-1)p q, 2q, … (p-1)q not rel. prime so have
  - pq – 1 – [(p-1) + (q-1)] = (p-1)(q-1)

Modular Exponentiation

- Note exponentiation by 3 acts as encryption of digits. Is there an inverse to this operation? Sometimes.
- Fact:

$$x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$$

  - Not true for all n, but for all any square free n (any n that doesn't have p^2 as a factor for any prime p)
- Note that if $y = 1 \bmod \phi(n)$, then $x^y \bmod n = x \bmod n$.

RSA

- Key length variable (but should now be at least 1024 bits)
- Plaintext block must be smaller than key length
- Ciphertext block will be length of key

RSA

- Choose two large primes (around 256 bits each) p and q. Let n = pq (very difficult to factor)
- Choose number e that is relatively prime to $\phi(n)$. Can do this since you know p and q and thus $\phi(pq)$ and from the derivation know exactly which numbers are relatively prime!
- Public key is <e, n>
- To make private key, find d that is the multiplicative inverse of e mod $\phi(n)$ (so ed = 1 mod $\phi(n)$) (use Euclid's algorithm)
- Private key is <d,n>
- To encrypt a number m, compute $c = m^e \bmod n$.
- To decrypt: $m = c^d \bmod n$.

RSA Example

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de = 1 \bmod 160$ and $d < 160$
   Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $KU=\{7,187\}$
7. Keep secret private key $KR=\{23,17,11\}$

RSA Example cont

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$

Questions

- Why does it work?
- Why is it secure?
- Are operations sufficiently efficient?
- How do we find big primes?

Why Does It Work?

- We chose d and e so that $de = 1 \bmod \phi(n)$, so for any x,
- $x^{(ed)} \bmod n = x^{(ed \bmod \phi(n))} \bmod n = x^1 \bmod n = x \bmod n$.
- And $(x^e)^d = x^{(ed)}$

Why Is It Secure?

- We're not sure it is, but it seems to be
- Based on premise that factoring a big number is difficult.
  - Semiprimes, the product of two (not necessarily distinct) primes, are most difficult numbers to factor.
  - Largest such semiprime yet factored is RSA-768, 768 bits, 232 decimal digits.
    - Took two years, hundreds of machines, several research institutions, and highly optimized code.
    - Equivalent of 2000 CPU years on a single-core 2.2 GHz AMD Opteron

Why Is It Secure?

- If you can factor n, you're golden:
  - Problem is one of finding modular log (i.e. inverse of exponential)
  - Why? Adversary knows <e,n>. So for message m, knows ciphertext is $c = m^e \bmod n$.
  - So if adversary can reverse the exponentiation (that is, find the number x s.t. $x^e \bmod n = c$), she's got the original message m!
  - Remember how we originally find this inverse: By knowing $\phi(n)$. Which is difficult to know if you can't factor n

Why Is It Secure?

- We don't know that there are not easier ways to break it (we do know that breaking it is no harder than factoring)

- We do know that it can be broken with a quantum computer using Shor's Algorithm (1994) which has cubic time and linear space complexity in the number of bits of the number being factored
  - So if quantum computers become practical...

This Can be Broken if Used Carelessly!

- Suppose Bob knows I'm going to send Alice a message with the identity of a congressperson who is crooked
  - So I'll encrypt name with Alice's public key and send resulting ciphertext
  - Bob knows Alice's public key, so he can encrypt all possible names and identify the crook.
  - A way around this: pad the name with a large random number, say 128 bits long. Then Bob has to try all of a space that is huge. (I.e. 535 x (2^{128}))

RSA Security

- three approaches to attacking RSA:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
  - timing attacks (on running of decryption)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor $N=p.q$, hence find $\varnothing(N)$ and then d
  - determine $\varnothing(N)$ directly and find d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - Recall factorization of RSA-768 in 2009
  - biggest improvement comes from improved algorithm
    - cf "Quadratic Sieve" to "Generalized Number Field Sieve"
  - barring dramatic breakthrough 1024+ bit RSA secure
    - ensure p, q of similar size and matching other constraints

Progress in Factorization

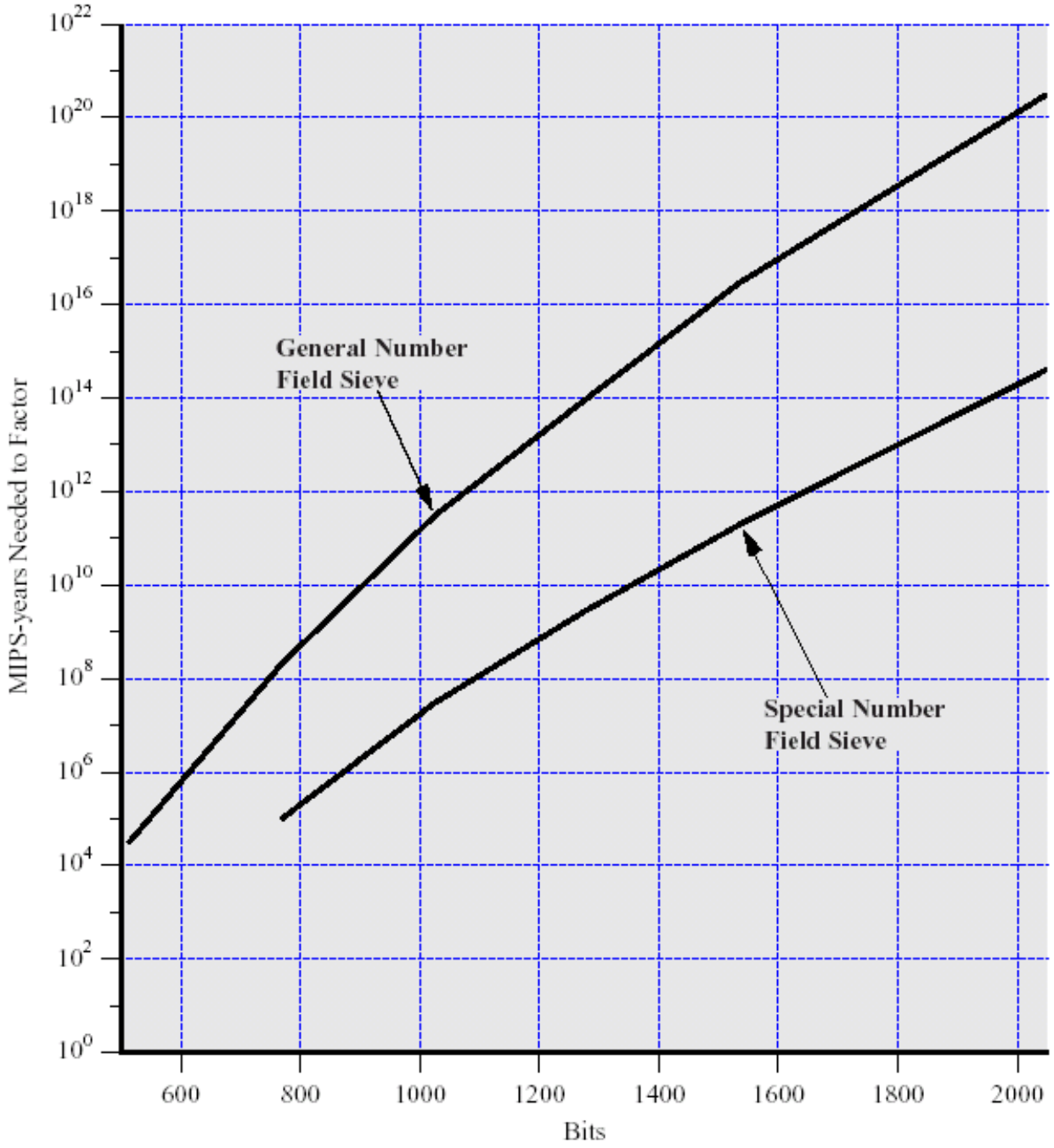
- In 1977, RSA inventors dare Scientific American readers to decode a cipher printed in Martin Gardner's column.
  - Reward of $100
  - Predicted it would take 40 quadrillion years
  - Challenge used a public key size of 129 decimal digits (about 428 bits)
- In 1994, a group working over the Internet solved the problem in 8 months.

Progress In Factorization

- Factoring is a hard problem, but not as hard as it used to be!
- MIPS year is a 1-MIPS machine running for a year
- For reference: a 3-GHz Pentium is about a 750-MIPS machine

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve

Figures are more than 10 years old. Check out the Wikipedia Integer Factorization Page for more recent figures.



How Efficient Is RSA?

- Need 5 efficient operations: encryption, decryption, generating a signature, verifying a signature, generating a key (less important since done less)
- These ops require taking large number, raising it to large number, and then finding remainder modulo large number. Way too slow if done straightforward way.
  - You can read about tricks to make this faster

Finding Big Primes

- By prime number theorem, probability of a number less than n being prime is about 1 / ln n.
- Thus, for example, a hundred digit number has about a 1 in 230 chance of being prime.
- No nice way of absolutely determining that a huge number is prime, but we can guess pretty accurately
- Fermat's Theorem: If p is prime, and 0 < a < p, then a^(p-1) mod p = 1 mod p.
  - Works because though it's possible for a^(n-1) = 1 mod n for a non-prime, it's not likely. For a randomly generated number of about 100 digits, probability that n is not prime but relation holds is about 1 in 10^(13).
  - Other similar probabilistic algorithms for finding large primes

UPDATE!!! The AKS Algorithm!

- The Agrawal-Keyal-Saxena Primality Test
  - Published in 2002 (after previous slide created)
  - A deterministic ***polynomial time*** primality-proving algorithm
  - Developed by three researchers at the Indian Institute of Technology Kanpur
  - Answered a centuries old question (and in a surprising way)!
  - Won 2006 Godel Prize and 2006 Fulkerson Prize
- Unfortunately the "constants" involved in the computational complexity estimates are very large
  - So not yet practical for identifying large primes (but making this competitive with probabilistic algorithms is a current research area)

Diffie-Hellman

- Oldest public key cryptosystem still in use
- Does neither encryption nor digital signatures.
- Used because it is fastest at what it does: allow two individuals to agree on a symmetric key even though they can only communicate over insecure channels.
- Remarkable because neither Alice nor Bob need any apriori information, yet after the exchange of two messages, they share a secret number.
- One bad thing: no authentication, so Alice may be setting up a key with Trudy!

The Process

- Alice and Bob agree on two primes, p and g, where p is a large prime and g is a number less than p (with some restrictions)
- Each chooses a random 1024 bit number (SA for Alice, SB for Bob).
- Alice computes TA = g^{SA} mod p. Bob computes TB = g^{SB} mod p.
- They exchange their T values
- Alice computes TB^{SA} mod p, Bob computes TA^{SB} mod p.
- Done: $TB^{SA} = (g^{SB})^{SA} = g^{(SB*SA)} = g^{(SA*SB)} = (g^{SA})^{SB} = TA^{SB}$ mod p.

Why It Is Secure

- Whole world knows g^{SA} and g^{SB}, but getting $g^{(SA*SB)}$ means having to do a modular logarithm
  - If can find y such that $g^{y} = g^{SA}$, then know SA.
- And well, it's not exactly secure -- it has that problem with a man-in-the-middle attack (the lack of authentication of endpoints)