Quadratic Sieve

- Runtime is similar to index calculus
  - tune { # of primes in factor base
           # of k's to test } (smoothness)
  - result: subexponential algorithm

- Next in class:
  - Euclidean algorithm
    - take gcd
    - modular inversion

Signature message m (document to be signed)

Alice (signer) Bob (verifier)

private key a
public key A

Signing

creates signature s as a function of (message m, private key a)

s →

Verifying

as a function of s, m, A

returns: valid or invalid

Properties:

1) verifying should return "valid" on a well-formed signature
2) it should be computationally infeasible to forge a signature on a given document for a given public key, i.e. create s so that s, m, A returns "valid".

Gives: authenticity: only Alice could have signed

integrity: document can't be changed after signing

non-repudiation: signer can't claim they didn't sign

RSA Signature

Idea: pretend document is a "ciphertext" & "decrypt" it.

Alice

RSA private key: d (dec. exp.)

RSA public key: $\begin{cases} n = pq & \text{(modulus)} \\ e & \text{(enc. exp)} \end{cases}$

message m

Signing: $S \equiv m^d \pmod n$ $\quad S \rightarrow$

Bob

Verifying:

$V \equiv S^e \pmod n$

If $V \equiv m$ then return valid

o/w return invalid

Hardness: Forging: Find S, given m, n, e

RSA problem

"plaintext" "signature" $\quad$ "ciphertext" "message"

Existential forgery

Start with s, compute $m = s^e \pmod n$

Then $s, m, (e, n)$ that returns valid.

BUT m is gobbledygook.

Hash functions (in signatures, hash the message to make it shorter for signing.)

A cryptographic hash function:

$$h : \{0,1\}^* \longrightarrow \{0,1\}^k$$

bit strings of any length → bit strings of length k

"message" $\longmapsto$ "digest"

Warning!

∞-to-1 function (not injection)

w/ properties:

1) efficient to compute
2) preimage resistance: Given digest d, it should

be computationally infeasible to find m w/ $h(m) = d$.

3) collision resistance: computationally infeasible to find $m_1 \neq m_2$ s.t. $h(m_1) = h(m_2)$.

4) avalanche effect: small changes in m should result in big changes in hash of m.

Ex. ① checksum: given message m, $h(m) = \Sigma$ bits of $m \pmod 2$

eg. $n = 1101$, $h(m) = 1$

Terrible cryptographic hash!

MD5	(90's)	— seconds to find a collision
SHA-1	(95 ish)	— broken also 2017
SHA-2	NIST 2001	— length extension attacks
SHA-3	NIST 2015	