1 Outline:

- ElGamal Signatures
- Digital Signature Algorithm (DSA)

2 El Gamal Signature Scheme

**Keygen:** generate a prime $p$ (1024 bits)

generator $g$ of $Z_p^*$

$x \in_R \{0, 1, \ldots, p - 2\}$

$y = g^x \pmod{p}$

$PK = (p, g, y)$

$SK = (x)$

**Question:** Is it okay if we take the first $n$ primes, multiply them all together and add or subtract 1 to get a prime number?

**Answer:** Those primes are bad for cryptography. If all the prime factors of $p - 1$ are relatively small, lots of cryptographic attacks are possible. Generally, primes $p$ such that $p - 1$ has a big prime factor are much better.

**Sign($M$):** (using $SK$ & $PK$)

$m = h(M)$

$h$ is a collision-resistant hash function

$k \in_R \{1, 2, \ldots, p - 2\}$ s.t. $gcd(k, p - 1) = 1$

($\in_R$ means choose at random $\rightarrow$ randomized signature scheme)

$r = g^k \pmod{p}$

$s = (m - rx)k \pmod{p - 1}$

output: $\sigma = (r, s)$

**Note:** $k, r$ can be computed before the message is seen. In addition, you need a new $k$ and $r$ everytime you sign a message. Otherwise, it will not be secure.

---

*May be freely reproduced for educational or personal use.*
3 Digital Signature Algorithm (DSA)

DSA is a public key signature algorithm and is specified by the NIST’s Digital Signature Standard\(^1\) (DSS). DSA is used to create a small\(^2\), publicly verifiable signature \(\sigma\) for a given message \(M\).

The DSA has three components, key generation, signature creation, and signature verification.

\(^1\)http://www.nist.gov/public_affairs/releases/digsigst.htm
\(^2\)A small signature is good.
Key Generation:  
\[ q = \text{some random 160-bit prime} \]
\[ p = qt + 1 \quad (1024 \text{ bits}) \]
\[ g \text{ with order } q, g \text{ not a generator (order } q, \text{ not } p - 1) \]
\[ x \in_R \{0, 1, \ldots, q - 1\} \]
\[ y = g^x \pmod{p} \]

public key : \((p, q, g, y)\)
secret key : \(x\)

Signature Creation:  
for message \(M\) calculate \(m = h(M)\)
\[ k \in_R \{1, 2, \ldots, q - 1\} \]
\[ r = (g^k \pmod{p}) \pmod{q} \]
\[ s = (m + rx) / k \pmod{q} \]
\[ \sigma = (r, s), \text{ where } r \text{ and } s \text{ are each 160 bits} \]

Signature Verification:  
given \(M, p, y, r, s, q, g, h\)
check if \(0 < r < q\) and \(0 < s < q\)
compute \(w = s^{-1} \pmod{q}\)
check if \(r = g^{wm}y^w \pmod{p} \pmod{q}\)

Question :  
Doesn’t DSA specify SHA-1 as the hash function \(h\)?
Answer :  
Yes, \(h\) is SHA-1 in DSA, so \(\text{length}(m)\) is 160 bits.

Question :  
I’ve heard that DSA has a subliminal channel...
Answer :  
Yes, it is true that a malicious manufacturer could use a guess and check algorithm to have some control over the signature output by manipulating \(k\). In this way, a DSA implementation could leak some bits of the secret key in each signature. The malicious manufacturer would only need to know the special encoding format to extract leaked bits of the key.\(^4\)

Question :  
What about the tightness or provability of the security of DSA?
Answer :  
These schemes used in practice are not provably tight or anything like the theory. In essence, one can only reduce such claims of provability or tightness to “hard” problems like factoring or discrete logarithms.

\(^3\)160-bit numbers are chosen because square root attacks, like the solution to Problem 1-1, would still require brute-forcing through \(2^{80}\) possibilities.

\(^4\)Note that any implementation that allowed a malicious manufacturer to guess \(k\) (for instance using a poor pseudorandom number generator) would allow the manufacturer to extract the secret key \(x\) from just one signature. Also of note is Karl’s clarification of this question and his answer to his own question, emailed out to the class list by the staff.