Cryptography

Lecture 19

Announcements

- HW7 due 4/22/19
- Sign up for Scholarly Paper EC

Agenda

- More Number Theory!

Extended Euclidean Algorithm Example #1

Find: X, Y such that $9X + 23Y = \gcd(9,23) = 1$.

$$23 = 2 \cdot 9 + 5$$
$$9 = 1 \cdot 5 + 4$$
$$5 = 1 \cdot 4 + 1$$
$$4 = 4 \cdot 1 + 0$$

$$1 = 5 - 1 \cdot 4$$
$$1 = 5 - 1 \cdot (9 - 1 \cdot 5)$$
$$1 = (23 - 2 \cdot 9) - \big(9 - (23 - 2 \cdot 9)\big)$$
$$1 = 2 \cdot 23 - 5 \cdot 9$$

$-5 = 18 \; mod \; 23$ is the multiplicative inverse of $9 \; mod \; 23$.

Extended Euclidean Algorithm Example #2

Find: X, Y such that $5X + 33Y = \gcd(5,33) = 1$.

$$33 = 6 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

$$1 = 3 - 1 \cdot 2$$
$$1 = 3 - (5 - 3)$$
$$1 = (33 - 6 \cdot 5) - \big(5 - (33 - 6 \cdot 5)\big)$$
$$1 = 2 \cdot 33 - 13 \cdot 5$$

$-13 = 20 \ mod \ 33$ is the multiplicative inverse of $5 \ mod \ 33$.

Chinese Remainder Theorem

Going from $(a, b) \in Z_p \times Z_q$ to $x \in Z_N$

Find the unique $x \bmod N$ such that

$$x \equiv a \bmod p$$
$$x \equiv b \bmod q$$

Recall since $\gcd(p, q) = 1$ we can write

$$Xp + Yq = 1$$

Note that

$$Xp \equiv 0 \bmod p$$
$$Xp \equiv 1 \bmod q$$

Whereas

$$Yq \equiv 1 \bmod p$$
$$Yq \equiv 0 \bmod p$$

Going from $(a, b) \in Z_p \times Z_q$ to $x \in Z_N$

Find the unique $x \bmod N$ such that

$$x \equiv a \bmod p$$
$$x \equiv b \bmod q$$

Claim:

$$b \cdot Xp + a \cdot Yq \equiv a \bmod p$$
$$b \cdot Xp + a \cdot Yq \equiv b \bmod q$$

Therefore, $x \equiv b \cdot Xp + a \cdot Yq \bmod N$

Modular Exponentiation

Modular Exponentiation

Is the following algorithm efficient (i.e. poly-time)?

ModExp(a, m, N) //computes $a^m \ mod\ N$

Set $temp := 1$

For $i = 1$ to m

Set $temp := (temp \cdot a) mod\ N$

return $temp$;

No—the run time is $O(m)$. m can be on the order of N. This means that the runtime is on the order of $O(N)$, while to be efficient it must be on the order of $O(\log N)$.

Modular Exponentiation

We can obtain an efficient algorithm via “repeated squaring.”

ModExp(a, m, N) //computes $a^m \ mod\ N$, where $m = m_{n-1}m_{n-2}\cdots m_1m_0$ are the bits of m.

Set $s \coloneqq a$

Set $temp \coloneqq 1$

For $i = 0$ to $n - 1$

    If $m_i = 1$

        Set $temp \coloneqq (temp \cdot s) mod\ N$

    Set $s \coloneqq s^2\ mod\ N$

return $temp$;

This is clearly efficient since the loop runs for n iterations, where $n = \log_2 m$.

Modular Exponentiation

Why does it work?

$$m = \sum_{i=0}^{n-1} m_i \cdot 2^i$$

Consider $a^m = a^{\sum_{i=0}^{n-1} m_i \cdot 2^i} = \prod_{i=0}^{n-1} a^{m_i \cdot 2^i}$.

In the efficient algorithm:

s values are precomputations of a^{2^i}, for $i = 0\ to\ n-1$ (this is the "repeated squaring" part since $a^{2^i} = (a^{2^{i-1}})^2$).

If $m_i = 1$, we multiply in the corresponding s-value.

If $m_i = 0$, then $a^{m_i \cdot 2^i} = a^0 = 1$ and so we skip the multiplication step.