Chapter 3
Modular Arithmetic

February 15, 2010

3

In computers information is represented digitally, and nearly always in binary. It can be considered to be integers and manipulated as such. Arithmetical operations often change the length of the quantities, for example multiplication doubles the length. Using modular arithmetic with all quantities modulo n (so that $x \ mod \ n$ is the remainder on dividing x by n) imposes a fixed length.

Modular arithmetic also imposes a structure. If n is a prime, p say, then all integers $mod \ p$ form a field with addition/subtraction, multiplication and inverses.

Example mod 7 we have $1 * 1 = 1, \ 2 * 4 = 1, \ 3 * 5 = 1 mod \ p$

If n is not prime but composite the integers $mod \ n$ do not form a field but those less than and prime (ltpt) to n form a multiplicative group of $\phi(n)$ members. $\phi(n)$ is called Euler's Totient Function.

Example $\phi(15) = 8$ and the members of the group are 1, 2, 4, 7, 8, 11, 13 and 14. For example $14 * 8 = 112 = 7 \ mod(15)$ which is another member.

A fundamental theorem used in cryptography is:

Theorem 0.1 $\alpha^{\phi(n)} = 1 \ mod \ n$ *provided* α *is prime to* n.

Proof Consider the $\phi(n)$ integers a_i ltpt n, and the $\phi(n)$ products $(a_i, \ \alpha)$. There are all distinct $mod \ n$, because $a_i\alpha = a_j\alpha$ implies $\alpha(a_i - a_j) = 0 \ mod \ n$ which is impossible since $\alpha, \ a_i, a_j$ have no common factor with n and $|(a_i - a_j)| < n$.

Therefore the $a_i\alpha$ over i are simply the a_j in a different order and

$$\prod_i (a_i\alpha) = \alpha^{\phi(n)} \prod_i a_i = \prod_j a_j \ mod\ n$$

from which we may cancel the products being prime to n and get

$$\alpha^{\phi(n)} = 1 \ mod\ n$$

When n is a prime p, $\phi(n) = \phi(p) = p-1$ and we get Fermat's Theorem. ∎

Theorem 0.2 (Fermat's Theorem)

$$\alpha^{p-1} = 1 \ mod\ p$$

Note that if $c = m^r mod\ n$ with $c,\ r$ and n known we can find m from $m = c^s mod\ n$ with $s = r^{-1} mod\ \phi(n)$ or $rs = 1\ mod\ \phi(n)$. This requires the ability to calculate $\phi(n)$ which in turn requirs the factors of n (see theorem **??**).

To find modular inverses, we use Euclid's Algorithm, which essentially finds the highest common factor (HCF) between two integers, and can be extended to find the inverse of one with respect to the other provided that $HCF = 1$.

3.1 Euclid's Algorithm

Suppose integers $a > b$. We divide a by b to find quotient q_0 and remainder r_0.

$$\mathrm{a} = \mathrm{q}_0 b + r_o$$

Repeat for $b,\ r_0 : b = q_1 r_0 + r_1$
Repeat for $r_0,\ r_1 : r_0 = q_2 r_1 + r_2$ etc.

At each steep the remainder is less than the divisor, the remainder from the previous step. Therefore $r_0 > r_1 > r_2 > \cdots > r_n$ and the process stops with $r_n > 0$ and $r_{n+1} = 0$.

$$\begin{aligned} \mathrm{r}_{n-3} &= \mathrm{q}_{n-1} r_{n-2} + r_{n-1} \\ \mathrm{r}_{n-2} &= \mathrm{q}_n r_{n-1} + r_n \\ \mathrm{r}_{n-1} &= \mathrm{q}_{n+1} r_n \end{aligned}$$

Since r_n divides r_{n-1} it must divide r_{n-2} etc. until we see r_n divides b and a. Conversely, if $h = HCF(a,\ b)$, h divides $a,\ b,\ r_0,\ r_1, \cdots,\ r_n$. Therefore $r_n = h = HCF(a,\ b)$. If $r_n = 1$ we may write

$$\begin{aligned} 1 &= \mathrm{r}_{n-2} - q_n r_{n-1} \\ &= \mathrm{r}_{n-2} - q_n(r_{n-3} - q_{n-1}r_{n-2}) \\ &= \mathrm{r}_{n-2}(1 + q_n q_{n-1}) - q_n r_{n-3} \\ &\quad \textit{etc. working up the ladder to get} \\ \pm 1 &= \mathrm{aB\text{-}Ab} \end{aligned}$$

If -1 applies we have $b^{-1} = A\ mod\ a$. If $+1$ applies we have $-Ab = 1 - aB$ and adding ab to both sides gives $b(a - A) = 1 + a(b - B)$ so $b^{-1} = (a - A)\ mod\ a$. Then we can calculate modular inverses using Euclid to find the HCF $(= 1)$ and working backwards up the ladder to find $b^{-1} mod\ a$

3.2 Continued Fraction Algorithm

There is a direct relationship between Euclid and continued fractions. This is illustrated below. It enables a faster one-pass algorithm for finding a modular inverse. At each stage as one finds the new q_j one evaluates $A_j,\ B_j$ (where $\frac{A_j}{B_j}$ is an approximation to $\frac{a}{b}$) using iterative formulae. The process ends with $aB_{n-1} - A_{n-1}b = (-1)^{n-1}$. For more background on continued fractions see Davenport "The Higher Arithmetic".

Remark If the quotients are all unity in a continued fraction expansion the ratio $\frac{A_j}{B_j}$ is that of adjacent Fiboncacci numbers.

Euclid	Example: $a = 67$ and $b = 24$
$a = q_0 b + r_0$	67=2∗24 + 19
$b = q_1 r_0 + r_1$	24=1∗19 + 5
$r_0 = q_2 r_1 + r_2$	19=3∗5 + 4
etc$\cdots$	5=1∗ 4 + 1
$r_{n-3} = q_{n-1} r_{n-2} + r_{n-1}$	4=4∗1
$r_{n-2} = q_n r_{n-1} + r_n$	
$r_{n-1} = q_{n+1} r_n$	

Continued Fractions	Example: $a = 67$ and $b = 24$
$\frac{a}{b} = q_0 + \frac{1}{\frac{b}{r_0}}$	$\frac{67}{24} = 2 + \frac{1}{\frac{24}{19}}$
$= q_0 + \frac{1}{q_1 + \frac{1}{\frac{r_0}{r_1}}}$	$= 2 + \frac{1}{1 + \frac{1}{\frac{19}{5}}}$
$= q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{1}{\frac{r_1}{r_2}}}}$	$= 2 + \frac{1}{1 + \frac{1}{3 + \frac{1}{\frac{5}{4}}}}$
	$= 2 + \frac{1}{1 + \frac{1}{3 + \frac{1}{1 + \frac{1}{\frac{4}{1}}}}}$
	$q_0 = 2\ q_1 = 1\ q_2 = 3\ q_3 = 1\ q_4 = 4$

Successive Approximations	to $\frac{a}{b} \longleftarrow \frac{A_i}{B_i}$
$A_0 = q_0$	$B_0 = 1$
$A_1 = q_0 q_1 + 1$	$B_1 = q_1$
$A_2 = q_0 q_1 q_2 + q_0 + q_2$	$B_2 = q_1 q_2 + 1$
$\cdots$	$\cdots$
easily shown that:	
$A_{m-1} = q_{m-1} A_{m-2} + A_{m-3}$	$B_{m-1} = q_{m-1} B_{m-2} + B_{m-3}$
$A_m = q_m A_{m-1} + A_{m-2}$	$B_m = q_m B_{m-1} + B_{m-2}$
Can prove that:	
$A_m B_{m-1} - A_{m-1} B_m = (-1)^{m-1}$	
At finish $(r_n = 1,\ r_{n+1} = 0)$	
$A_n = a\ B_n = b$	
So $aB_{n-1} - A_{n-1} b = (-1)^{n-1}$	
which gives inverse of $b\ mod\ a = \pm A_{n-1}$	

Example	
$A_0 = 2$	$B_0 = 1$
$A_1 = 3$	$B_1 = 1$
(use iteration formula)	
$A_2 = 3.3 + 2 = 11$	$B_2 = 4$
$A_3 = 1.11 + 3 = 14$	$B_3 = 1.4 + 1 = 5$
$A_4 = 4.14 + 11 = 67 = a$	$B_4 = 4.5 + 4 = 24 = b$
So $aB_3 - bA_3 = (-1)^3$	
$67.5 - 24.14 = -1$	
Therefore $24^{-1} mod\ 67 = 14$	

3.3 The Chinese Remainder Theorem (CRT)

The theorem states that if $a,\ b$ are coprime (and $a > b$) then there exists a unique solution $mod\ (ab)$ to the two equations:

$$x = \alpha\ mod\ a$$
$$x = \beta\ mod\ b$$

The theorem is fundamental in handling situations where the modulus is composite. We 'prove' the theorem by solving the two equations as follows:
$x = \beta + jb$ for some j, so
$x - \beta = jb$ and taking modulo a gives
$\alpha - \beta = jb\ mod\ a$. Solving for j gives
$j = b^{-1}(mod\ a)(\alpha - \beta)$.
Substitute in $x = \beta + jb$.

Example

$$\begin{array}{lcll} \text{x} & = & 3 \bmod 17 & (a) \\ \text{x} & = & 5 \bmod 11 & (b) \\ \text{j} & = & 11^{-1}(mod\ 17)(3-5) & \end{array}$$

By Euclid $11^{-1} mod\ 17 = 14$
Therefore $j = 15 * 14 mod\ 17 = 6$
So $x = 5 + 6 * 11 = 71\ mod\ 187$

Usually CRT is used when a and b are primes but the procedure is readily extended to serve when they are composite provided that $(a,\ b)$ are co-prime.

3.4 Theorems about Euler's Totient Function $\phi()$

An immediate consequence of CRT is:

Theorem 0.3 $\phi(ab) = \phi(a)\phi(b)$ *provided that* $(a,\ b)$ *are coprime.*

Proof By CRT there are $\phi(a) * \phi(b)$ distinct solutions $mod\ (ab)$ to

$$x = \alpha\ mod\ a,$$
$$x = \beta\ mod\ b$$

where $\alpha,\ \beta$ are less than and prime to (ltpt) $(a,\ b)$ respectively.

The solution x is in turn ltpt (ab). Therefore $\phi(ab) \geq \phi(a) * \phi(b)$. But any x ltpt (ab) taken $mod\ a$ is ltpt a and similarly for b, therefore it has already been considered. Therefore $\phi(ab) = \phi(a) * \phi(b)$ ∎

This leads to:

Theorem 0.4 *If $n = \prod_i p_i^{e_i}$ where the p_i are the prime factors of n then*

$$\phi(n) = n \prod_i (1 - \tfrac{1}{p_i})$$

Proof $\phi(p^e) = p^e - p^{e-1}$ if p is prime, because this is the number of integers less than p^e less those divisible by p. By Theorem**??**

$$\phi(n) = \prod_i (p_i^e - p_i^{e_i - 1}) = n \prod_i (1 - \tfrac{1}{p_i}) \qquad \blacksquare$$

Example $\phi(36) = \phi(3^2 * 2^2) = 36(a - \frac{1}{3})(1 - \frac{1}{2}) = 12$
Namely 1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35.

Theorem 0.5 *If d_i is a divisor of m then $m = \sum_i \phi(d_i)$ the sum being over all dividers including m itself and 1.*

Example $m = 12;\ d_i = 1,\ 2,\ 3,\ 4,\ 6,\ 12\ \ \phi(d_i) = 1,\ 1,\ 2,\ 2,\ 2,\ 4$

Proof Suppose $m = \sum_j p_j^{e_j}$ with p_j prime then the dividers d_i have the form $\prod_j p_j^{f_j}\ 1 \leq f_j \leq e_j$.

Consider all the dividers of form $p_1^{f_{1i} x_1}$ where x_1 is composed of all the other terms $p_j^{f_j}$ except those containing p_1 and $1 \leq f_{1i} \leq e_1$

$$\begin{array}{rcl} \sum_i \phi(x_1 p_1^{f_{1i}}) & = & \phi(x_1 \sum_i \phi(p_1^{f_{1i}}) \\ & = & \phi(x_1)(1 + (p_1 - 1) + p_1(p_1 - 1) + \ \ldots\ + p_1^{e_1 - 1}(p_1 - 1)) \\ & = & \phi(x_1) p_1^{e_1} \text{ using Theorem } \textbf{??}. \end{array}$$

Repeat the process extracting p_2 from x_1 to get

$$\sum(alldividers) = \phi(x_2) p_2^{e_2} p_1^{e_1}$$

where x_2 contains no terms with p_1 or p_2. The procedure finishes with

$$\sum(alldividers) = \prod_j p_j^{e_j} = m \qquad \blacksquare$$

3.5 Theorems about the Order of Elements mod(n)

The elements α ltpt n form a group (closed under multiplication, with inverses $\alpha^{\phi(n)-1}$) of order $\phi(n)$. We define the order of α $(\theta(\alpha)) = thelowestintegerj$ such that $\alpha^j = 1 mod\ n$.

Theorem 0.6 *If $r = \theta(\alpha) mod\ n$ then $r|\phi(n)$ (r divides $\phi(n)$)*

Proof Suppose $\phi(n) = ir + k$ then $k < r$

$$1 = \alpha^{\phi(n)} = \alpha^{ir+k} = \alpha^k mod\ n$$

which is a contradiction because r is the lowest integer. Therefore $k = 0$. ∎

Definition An element is called *primitive* if it's order is the order of the group $(\theta(\alpha) = \phi(n))$.

Theorem 0.7 *If the modulus is a prime p there exists a primitive element in the group $GF(p)$.*

Proof Firstly we show that if elements $a,\ b$ in the group have orders $e,\ f$ which are coprime then $\theta(ab) = ef$ because

$$(\text{ab})^{ef} = (a^e)^f.b^f)^e = 1\ mod\ p \text{ therefore } Order(ab)|ef$$

and if $x = Order(ab)$ $(ab)^{xe} = b^{xe} = 1 mod\ p$ so $f|x$ and similarly $e|x$. Therefore $ef|x$ or $ef|Order(ab)$. ∎

Now suppose $\phi(p) = p - 1 = \prod p_i^{e_i}$. p_i prime Let $q = \frac{p-1}{p_1}$. There exist elements in the group $(GF(p))$ which do not satisfy $x^q = 1\ mod\ p$ because there are at most q solutions to this equation in a field. Let a_1 be such an element with $Order(a_1) = \prod p_i^{f_i}$. But $f_1 = e_1$ otherwise a_1 would satisfy $x^q = 1\ mod\ p$.

Consequently $\alpha_1 = a_1^{p_2^{f_2} p_3^{f_3}} \ldots$ has order $p_1^{f_1} = p_1^{e_1}$. Similarly we can find α_2 with order $p_2^{e_2}$ etc. Therefore $\prod \alpha_i$ has order $\prod p_i^{e_i} = p - 1$ and $\prod \alpha_i$ is primitive.

Example $p = 13$. 2 is primitive. $1,\ 2,\ 2^2 = 4,\ 2^3 = 8,\ 2^4 = 3,\ 2^5 = 6,\ 2^6 = 12\ (= -1),\ 2^7 = 11,\ 2^8 = 9,\ 2^9 = 5,\ 2^{10} = 10,\ 2^{11} = 7,\ 2^{12} = 1$

Theorem 0.8 *If α has order d mod n then $\theta(\alpha^s = \frac{d}{h})$ where $h = HCF(d, s)$.*

Proof Suppose $d = ih,\ s = jh\ (i,\ j)$ coprime then $(\alpha^s)^{\frac{d}{h}} = \alpha^{jd} = (\alpha^d)^j = 1$ therefore $Order(\alpha^s)|\frac{d}{h}$. If $\theta(\alpha^s) = x$ then $1 = (\alpha^s)^x = \alpha^{jhx}$ $d|jhx$, or $\frac{d}{h} = i|x = \theta(\alpha^s)$. Therefore $\theta(\alpha^s) = \frac{d}{h}$. ∎

Corollary 0.9 *If $n = p$ prime and α is primitive we have $\theta(\alpha^{d_i}) = \frac{p-1}{d_i} = d'_i$ where d'_i (also a divisor of (p-1)) is the 'complement' of d_i. The number of elements of order $= d'_i$ is $\phi(d'_i)$. See theorem* **??**.

Example $p = 13,\ p-1 = 12,\ d_1 = 1,\ d_2 = 2,\ d_3 = 3,\ d_4 = 4,\ d_5 = 6,\ d_6 = 12,\ \alpha = 2$

d_i	=	1	2	3	4	6	12
2^{d_i}	=	2	4	8	3	12	1
$\mathrm{d}'_i = Order(2^{d_i})$	=	12	6	4	3	2	1
Number of elements of this order$=\phi(d'_1)$	=	4	2	2	2	1	1

$$
\begin{aligned}
Elements\ of\ Order &= 12\ are\ 2, 6, 11, 7\\
Elements\ of\ Order &= 6\ are\ 4, 10\\
Elements\ of\ Order &= 4\ are\ 5, 8\\
Elements\ of\ Order &= 3\ are\ 3, 9\\
Elements\ of\ Order &= 2\ are\ 12\\
Elements\ of\ Order &= 1\ are\ 1
\end{aligned}
$$

Theorem 0.10 *If $n = n_1 n_2$ (n_1, n_2) coprime and*

$$
\begin{aligned}
\theta(\alpha) &= k_1 mod\ n_1\\
\theta(\alpha) &= k_2 mod\ n_2
\end{aligned}
$$

then $\theta(\alpha) mod\ (n_1 n_2) = LCM(k_1,\ k_2)$. Where LCM is the least common multiple.

Proof Let LCM $= \lambda$ so $\lambda = s_1 k_1 = s_2 k_2$.

$$
\begin{aligned}
&\alpha^{k_1} = 1\ mod\ n_1 \quad therefore\ \alpha^{\lambda} = 1\ mod\ n_1 \quad therefore\ \mathrm{n}_1|(\alpha^{\lambda} - 1)\\
&\alpha^{k_2} = 1\ mod\ n_2 \quad therefore\ \alpha^{\lambda} = 1\ mod\ n_2 \quad therefore\ \mathrm{n}_2|(\alpha^{\lambda} - 1)
\end{aligned}
$$

But $n_1,\ n_2$ coprime therefore $(n_1 n_2)|(\alpha^{\lambda} - 1)$, therefore $Order(\alpha) mod\ n_1 n_2 | \lambda$. Let $x = Order(\alpha) mod (n_1 n_2)$ $\alpha^x = 1\ mod\ n_1 n_2$ so

$$
\begin{aligned}
\alpha^x &= 1 \bmod \mathrm{n}_1 \quad so\ \mathrm{k}_1|x\\
\alpha^x &= 1 \bmod \mathrm{n}_2 \quad so\ \mathrm{k}_2|x
\end{aligned}
$$

Therefore $LCM(k_1,\ k_2) = \lambda | x = Order(\alpha)$, Therefore $Order(\alpha) = \lambda\ mod (n_1 n_2)$ ∎

Corollary 0.11 *There exists no primitive element for a composite modulus (except when n_1 or $n_2 = 2$) because the maximum values for k_1, k_2 are $\phi(n_1)$, $\phi(_2)$ and $\lambda < \phi(n_1)\phi(n_2)$ because they have a common factor of two. Now $\phi(n_1 n_2) = \phi(n_1)\phi(n_2)$.*
Therefore the max order of element $< \phi(n_1 n_2)$.

Example $n_1 = 3,\ n_2 = 7,\ \ \phi(n_1) = 2,\ \phi(n_2) = 6,\ \ max\ \lambda = 6,\ \phi(21) = 12.$

Appendix X
Some superficial notes on Multi-precision Arithmetic

Because the numbers (integers) used in cryptology are large - or at best very much larger than those handled by current computer hardware (32 or 64 bits) - multi-precision arithmetic using special software packages is essential. The following notes discuss firstly the general ideas involved in multi-precision addition/subtraction and multiplication/division. (Exponentiation is just repeated multiplication.) Computers have hardware-implemented arithmetic of their own word-length (example: 32 or 64 bits) which can be invoked by a multi-precision software package if useful.

Cryptographic arithmetic is frequently modular (*mod n*) as well as involving large integers. Two approaches to speeding up such calculations (Karatsuba and montgomery) are appended.

1. We assume a word-length of s bits - in positive (i.e. not "two's complement") form. All integers are positive.

2. The "base" is then $B = 2^s$, and a multi-precision integer is of the form $a_n B^{n-1} + a_{n-1} B^{n-2} + ... + a_2 B + a_1$ where a_i is a positive s-bit integer. (Example: $S = 32$)

3. We are concerned with modular arithmetic in which quantities are *mod* N, and so if $B^{n-1} < N \leq B^n$ we have at most n words in a multi-precision integer.

4. In practice this means that when adding two integers $a,\ b(< N$ of course) we may need to subtract N from the result; when subtracting b from a we need to test if $b > a$ and if so add in N; and when multiplying we need to find the remainder on dividing the product of two integers by N. If $N = B^n$ all this is easy, we simply discard all the words to

the left of B^n. (In reality $N \neq B^n$, in particular because we require N to be a prime or a product of two primes, but for the following timings let us ignore this and suppose $N = B^n$).

Assume single precision addition takes time t, and single precision multiplication takes time m - note: the multiplication of two single precision integers produces a double-precision result.

Integer length m bits	Time for Addition	Time for Multiplication
B	t	m
2B	2t	3m+2t
4B	4t	10m+9t
8B	8t	36m+35t

Very roughly: Double the length of integers results in doubling the time for addition, and quadrupling the time for multiplication.

What of exponentiation? Let us assume the size of exponents is the size of the other integers. An exponent of m bits needs m multiplications so we have:

Integer length m bits	Time for exponentiation
B	2(s-1)m
2B	2(2s-1)(3m+2t)
4B	2(4s-1)(10m+9t)
8B	2(8s-1)(36m+35t)

Double the length of integers results in multiplying the time taken by approximately eight. Exponentiation is of cubic complexity.

Note: Example: $(s = 4\,bits)\; x^{15} = x^8 * x^4 * x^2 * 1 = 3\,squarings\,and\,3\,multiplications = 6\,operations = 2(s-1)$

In the preceding we have assumed modularity is achieved by discarding - i.e. $N = B^n$. This is not so usually. Modularity has to be achieved by division. Division is a complex, trial and error procedure. To divide a by b, assuming a is significantly greater than b, we

1. Shift b left $(b * 2^i)$ until $b2^i \leq a$ but $b2^{i+1} > a$, call $b2^i$ now b.

2. Subtract b from a to form $a' = a - b$.

3. Shift b right to get $b' = \frac{b}{2}$.

4. Is $b' < a'$? If yes the go to step two (with $b = b',\ a = a'$), if no then go to step 3.

If the quotient is of interest, record a '1′ for each subtraction in step 2 in the appropriate quotient register. If we are only interested in the remainder stop the process when b can't be shifted further right, and then the remainder is equal to a'.

The number of subtractions is (very roughly) ns, for integers of length n s-bits words. (The situation is complicated by working across word boundaries, the possibility of using some hardware division, the fact that many reductions modulo-N take place on the results (DOUBLE-precision) of multiplications, etc). Assuming this is correct we have for example:

Integer length	Time for Multiplication Modular	Time for Exponentiation Modular
$8B$	$36m + t(35 + 8s)$	$2(8(s-1))(36m + t(35 + 8s))$

If $m = t$ and $s = 32$ for example, time for 256-bit modular exponentiation $2 * 8 * 31(36 + 35 + 256)t = 1.6 * 10^5 t$

3X.1 Karatsuba speed up of Multi-precision Multiplication

Suppose we work with multi-precision integers of length $2^k s$ bits (example: $k = 5$, $m = 32$ for 1024-bit arithmetic). We start with the fact

$$\text{a}*b = ((a+b)^2 - (a-b)^2)/(4) \tag{0.2}$$

Thus multiplication can be reduced to two squarings (plus a trivial right shift).

Squaring can be made to take $\frac{3}{4}$ of the time that direct multiplication takes. As follows:

$$
\begin{array}{rcl}
\textit{Let} & \mathrm{a} & =(\mathrm{a}_1, a_2) \text{ and} \\
\textit{let} & 2^k s & =2\mathrm{j}=\text{length of a} \\
\textit{then} & \mathrm{a} & =\mathrm{a}_1 2^j + a_2 \\
\textit{so} & \mathrm{a}^2 & =\mathrm{a}_1 2^{2j} + a_1 a_2 2^{j+1} + a_2^2 \\
 & & =2^j(a_1+a_2)^2 + (2^{2j}-2^j)a_1^2 + (1-2^j)a_2^2 \\
 & & = 3 \textit{ squarings of integers of length}\mathrm{j}=2^{k-1}s \text{ (with some other trivial operations)}
\end{array}
$$

So the time is $\frac{3}{4}$ of the time to square (a_1, a_2) of length $2^k s$.

But we can iterate to evaluate squarings of length $2^{k-1}s$ as three squarings of length $2^{k-2}s$; etc until $2^{k-i}=1$, i.e. k times. Each time we gain $\frac{3}{4}$, so final time (taking into account it has to be done twice see equation **??** above) is $2*(\frac{3}{4})^k$ of unoptimised time. If $k=5$ we have cut to 0.475 unoptimised time.

3X.1.1 Modular Multiplication Without Trial Division (Montgomery)

The idea: change the representation of integers *mod* N to another representation. Perform the operation - example: repeated multiplication as in squaring - in this new more convenient representation. When finished, convert the result back to original representation.

Of course, the initial conversion and final reconversion cost time. But, the intervening operation (if repeated) are much faster - hence the ultimate gain.

Let N be the modulus, $x,\ y$ two integers $< N$. Let R be a convenient base (example: 2^k for some k) and form $x' = Rx \ mod\ N$ and $y' = Ry \ mod\ N$. (Note: $R,\ N$ co-prime and $R > N$). In this representation $(x+y)' = x' + y'$. However $x'y' \neq (xy)'$ because $x'y' = R^2 xy \ mod\ N$. Therefore to get the correct form for $(xy)'$ we need to evaluate

$$
\begin{array}{rcl}
\mathrm{R}^{-1}x'y' \ mod\ N & = & \mathrm{R(xy)\ mod\ N} \\
\textit{Let } \mathrm{R}^{-1} \text{ be inverse of} & & \mathrm{R \ mod\ N} \textit{ so that} \\
\mathrm{RR}^{-1} & = & 1+\mathrm{NN}' \text{ for some } N'
\end{array}
$$

Since $R^{-1} < N$ we must have $N' < R$. To convert any T to $R^{-1}T \ mod\ N$ (for example $T = x'y'$) we use

$$
\begin{array}{rcl}
\mathrm{TRR}^{-1} & = & \mathrm{T}(1+\mathrm{NN}') \\
\textit{So } \mathrm{TR}^{-1} \ mod\ N & = & (\mathrm{T}(1+\mathrm{NN}'))/(R) \ mod\ N \\
 & = & (\mathrm{T}+(((\mathrm{T\ mod\ R})\mathrm{N}')mod\ R)N)/(R) \ mod\ N
\end{array}
$$

If $T < N^2$ (example: $T = x'y'$) then $T < RN$. So $TR^{-1} \ mod\ N < \frac{RN+RN}{R} =$

$2N \ mod \ N$ and the $mod \ N = 2N \ mod \ N$ means either subtract N once or not at all.

3X.2 Algorithm

1. Form $m = ((T \ mod \ R)N') \ mod \ R$ giving $m < R$.
2. $t = \frac{(T+mN)}{R}$.
   Note: t is an integer since $mN = TN'N \ mod \ R = -T \ mod \ R$
   Note: $tR + T \ mod \ N$, therefore $t = TR^{-1} \ mod \ R$
   Division is trivial
3. If $t \geq N$ then $t = t - N$

Answer: $t = R^{-1}T \ mod \ N$. That is in evaluating $x * y \ mod \ N$ instead of one multiplication and one division we have three multiplication ($x'y'$, TN', mN) (Plus initial conversions x y)

Example $x = 17, \ y = 24, \ N = 29, \ xy \ mod \ N =?$
Choose $R = 32(= 2^5) \ \ R^{-1} \ mod \ 29 = 10, \ N' = 11$

$$\begin{aligned} \text{x}' &= \text{Rx mod N}=22 \\ \text{y}' &= \text{Ry mod N}=14 \\ so \ \text{x}'y' &= 308 \end{aligned}$$

$$\begin{aligned} So(\text{xy})' &= \text{Rxy}=\text{R}^{-1}x'y' \ mod \ N \\ &= (308+((308(\text{mod } 32).11)\text{mod } 32)*29)/(32) \\ &= (308+(20*11) \ mod \ 32 * 29)/(32) \\ &= (308+28*29)/(32) \\ &= 35 \end{aligned}$$

To convert $(xy)' = Rxy \ mod \ N$ back to xy, multiply by R^{-1}

$$\begin{aligned} \text{xy}=\text{R}^{-1}(xy)' &= (35+(((35 \text{ mod } 32).11)\text{mod } 32)29)/(32) \\ &=(35+29)/(32) \\ &=2 \text{ mod } 29 \ (=17*24 \ mod \ 29) \end{aligned}$$