open nat

variables one x n p q d e : nat

variable gcd (a b : nat) : nat

variable phi (n : nat) : nat

variable prime (a : nat) : Prop

variable congruent (a b modulus : nat) : Prop

premise nDef : n = p \* q

premise pPhi : phi p = p - one

premise qPhi : phi q = q - one

premise nPhi : phi n = (q - one) \* (p - one)

premise xLess : x < n

premise pIsPrime : prime p

premise qIsPrime : prime q

premise de\_Inverse : (congruent (d \* e) one (phi n))

premise OneMul : ∀ n : nat, one \* n = n

premise OneExp : ∀ n : nat, one ^ n = one

premise ExpOne : ∀ n : nat, n ^ one = n

premise ExpSum (a b c : nat) : (a ^ b) \* (a ^ c) = a ^ (b + c)

premise ExpMul (a b c : nat) : (a ^ b) ^ c = a ^ (b \* c)

premise ExpSwap (a b c : nat) : (a ^ b) ^ c = (a ^ c) ^ b

premise ModuloDef (a b n : nat) : congruent a b n ↔ ∃ c, a = b + (c \* n)

premise CongruenceReflexivity (a b n : nat) : congruent a b n → congruent b a n

premise CongruenceScaling (a b n k : nat) : congruent a b n → congruent (a \* k) (b \* k) n

premise CongruenceExponentiation (a b n k : nat) : congruent a b n → congruent (a ^ k) (b ^ k) n

premise EqMul (a b k : nat) : a = b → k \* a = k \* b

premise MulDistrib (a b c : nat) : a \* (b + c) = (a \* b) + (a \* c)

premise Euler (a b : nat) : gcd a b = one → congruent (a ^ (phi b)) one b

premise GCDProperty (a b c : nat) : gcd a (b \* c) ≠ one → prime b → prime c → a < b \* c → (((∃ n, a = n \* b) ∧ (gcd a c = one)) ∨ ((∃ n, a = n \* c) ∧ (gcd a b = one)))

theorem ProofCoprime (t : nat) : gcd x n = one → congruent (x \* ((x ^ (phi n)) ^ t)) x n :=

assume H1 : gcd x n = one,

have H2 : congruent (x ^ (phi n)) one n, from Euler x n H1,

have Hexpt : congruent (x ^ (phi n)) one n → congruent ((x ^ (phi n)) ^ t) (one ^ t) n, from CongruenceExponentiation (x ^ (phi n)) one n t,

have H3 : congruent ((x ^ (phi n)) ^ t) (one ^ t) n, from Hexpt H2,

have Honet : one ^ t = one, from OneExp t,

have H4 : congruent ((x ^ (phi n)) ^ t) one n, from eq.subst Honet H3,

have Hmulx : congruent ((x ^ (phi n)) ^ t) one n → congruent (((x ^ (phi n)) ^ t) \* x) (one \* x) n, from CongruenceScaling ((x ^ (phi n)) ^ t) one n x,

have H5 : congruent (((x ^ (phi n)) ^ t) \* x) (one \* x) n, from Hmulx H4,

have Honex : one \* x = x, from OneMul x,

have H6 : congruent (((x ^ (phi n)) ^ t) \* x) x n, from eq.subst Honex H5,

show congruent (x \* ((x ^ (phi n)) ^ t)) x n, from eq.subst (mul.comm ((x ^ (phi n)) ^ t) x) H6

theorem ProofNotCoprime\_Part1 : gcd x n ≠ one → (((∃ n, x = n \* p) ∧ (gcd x q = one)) ∨ ((∃ n, x = n \* q) ∧ (gcd x p = one))) :=

assume H1 : gcd x n ≠ one,

have H2 : x < p \* q, from eq.subst nDef xLess,

have H3 : gcd x (p \* q) ≠ one, from eq.subst nDef H1,

show (((∃ n, x = n \* p) ∧ (gcd x q = one)) ∨ ((∃ n, x = n \* q) ∧ (gcd x p = one))), from GCDProperty x p q H3 pIsPrime qIsPrime H2

theorem ProofNotCoprime\_Part2 (t : nat) : (∃ n, x = n \* p) ∧ (gcd x q = one) → congruent (x \* ((x ^ (phi n)) ^ t)) x n :=

assume H1 : (∃ n, x = n \* p) ∧ (gcd x q = one),

have gcd x q = one, from and.elim\_right H1,

have congruent (x ^ (phi q)) one q, from Euler x q this,

have congruent ((x ^ (phi q)) ^ t) (one ^ t) q, from CongruenceExponentiation (x ^ (phi q)) one q t this,

have congruent ((x ^ (phi q)) ^ t) one q, from eq.subst (OneExp t) this,

have congruent (((x ^ (phi q)) ^ t) ^ (p - one)) (one ^ (p - one)) q, from CongruenceExponentiation ((x ^ (phi q)) ^ t) one q (p - one) this,

have congruent (((x ^ (phi q)) ^ t) ^ (p - one)) one q, from eq.subst (OneExp (p - one)) this,

have congruent (((x ^ (phi q)) ^ (p - one)) ^ t) one q, from eq.subst (ExpSwap (x ^ (phi q)) t (p - one)) this,

have congruent ((x ^ ((phi q) \* (p - one))) ^ t) one q, from eq.subst (ExpMul x (phi q) (p - one)) this,

have congruent ((x ^ ((q - one) \* (p - one))) ^ t) one q, from eq.subst qPhi this,

have congruent ((x ^ (phi n)) ^ t) one q, from eq.subst (eq.symm nPhi) this,

have ∃ c, ((x ^ (phi n)) ^ t) = one + (c \* q), from (iff.elim\_left (ModuloDef ((x ^ (phi n)) ^ t) one q)) this,

exists.elim this (fun (v : nat) (Hv : ((x ^ (phi n)) ^ t) = one + (v \* q)),

have x \* ((x ^ (phi n)) ^ t) = x \* (one + (v \* q)), from EqMul ((x ^ (phi n)) ^ t) (one + (v \* q)) x Hv,

have x \* ((x ^ (phi n)) ^ t) = (x \* one) + (x \* (v \* q)), from eq.trans this (MulDistrib x one (v \* q)),

have x \* ((x ^ (phi n)) ^ t) = (one \* x) + (x \* (v \* q)), from eq.subst (mul.comm x one) this,

have Hxvq : x \* ((x ^ (phi n)) ^ t) = x + (x \* (v \* q)), from eq.subst (OneMul x) this,

have ∃ n, x = n \* p, from and.elim\_left H1,

exists.elim this (fun (w : nat) (Hw : x = w \* p),

have x \* ((x ^ (phi n)) ^ t) = x + ((w \* p) \* (v \* q)), from eq.subst Hw Hxvq,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* (p \* (v \* q))), from eq.subst (mul.assoc w p (v \* q)) this,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* (p \* (q \* v))), from eq.subst (mul.comm v q) this,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* ((p \* q) \* v)), from eq.subst (eq.symm (mul.assoc p q v)) this,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* (n \* v)), from eq.subst (eq.symm nDef) this,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* (v \* n)), from eq.subst (mul.comm n v) this,

have x \* ((x ^ (phi n)) ^ t) = x + (w \* v) \* n, from eq.subst (eq.symm (mul.assoc w v n)) this,

have ∃ i, x \* ((x ^ (phi n)) ^ t) = x + i \* n, from exists.intro (w \* v) this,

show congruent (x \* ((x ^ (phi n)) ^ t)) x n, from (iff.elim\_right (ModuloDef (x \* ((x ^ (phi n)) ^ t)) x n)) this))

theorem ProofFinal (t : nat) : congruent (x \* ((x ^ (phi n)) ^ t)) x n → congruent (x ^ (one + ((phi n) \* t))) x n :=

assume H1 : congruent (x \* ((x ^ (phi n)) ^ t)) x n,

have H2 : congruent (x \* (x ^ ((phi n) \* t))) x n, from eq.subst (ExpMul x (phi n) t) H1,

have Honex : x = x ^ one, from eq.symm (ExpOne x),

have H3 : congruent ((x ^ one) \* (x ^ ((phi n) \* t))) x n, from eq.subst Honex H2,

show congruent (x ^ (one + ((phi n) \* t))) x n, from eq.subst (ExpSum x one ((phi n) \* t)) H3