

# The Group Law for Elliptic Curves

Stefan Berghofer

December 7, 2022

## Abstract

We prove the group law for elliptic curves in Weierstrass form over fields of characteristic greater than 2. In addition to affine coordinates, we also formalize projective coordinates, which allow for more efficient computations. By specializing the abstract formalization to prime fields, we can apply the curve operations to parameters used in standard security protocols.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Formalization using Axiomatic Type Classes</b>	<b>2</b>
2.1	Affine Coordinates . . . . .	2
2.2	Projective Coordinates . . . . .	38
<b>3</b>	<b>Formalization using Locales</b>	<b>45</b>
3.1	Affine Coordinates . . . . .	45
3.2	Projective Coordinates . . . . .	85
<b>4</b>	<b>Validating the Specification</b>	<b>93</b>
4.1	Specialized Definitions for Prime Fields . . . . .	93
4.2	The NIST Curve P-521 . . . . .	98

## 1 Introduction

Elliptic curves play an important role in cryptography, since they allow to achieve a security level that is comparable to that of RSA, while requiring a smaller key size and less computation time. The primitive operation on elliptic curves is *point addition*. To ensure the proper functioning of cryptographic algorithms based on elliptic curves, such as Diffie-Hellman key exchange (ECDH) or digital signatures (ECDSA), it is important that the points on the curve form a group with respect to point addition.

Our formalization of elliptic curves is based on earlier work by Laurent Théry in Coq [4]. Like its Coq counterpart, the Isabelle formalization uses decision procedures for rings and fields based on reflection, which are executed using Isabelle’s code generator for efficiency reasons. The decision procedure for rings is due to Grégoire and Mahboubi [3] and was ported from Coq to Isabelle by Bernhard Haeupler.

The formalization exists in two flavours: one based on axiomatic type classes, and another one based on locales. While the axiomatic type class version is more concise, the locale version is more suitable for working with concrete rings or fields like prime fields.

## 2 Formalization using Axiomatic Type Classes

```
theory Elliptic-Axclass
imports HOL-Decision-Procs.Reflective-Field
begin
```

### 2.1 Affine Coordinates

```
datatype 'a point = Infinity | Point 'a 'a
```

```
class ell-field = field +
  assumes two-not-zero:  $2 \neq 0$ 
begin
```

```
definition nonsingular :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool where
  nonsingular a b =  $(4 * a^3 + 27 * b^2 \neq 0)$ 
```

```
definition on-curve :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a point  $\Rightarrow$  bool where
  on-curve a b p = (case p of
    Infinity  $\Rightarrow$  True
  | Point x y  $\Rightarrow$   $y^2 = x^3 + a * x + b$ )
```

```
definition add :: 'a  $\Rightarrow$  'a point  $\Rightarrow$  'a point  $\Rightarrow$  'a point where
  add a p1 p2 = (case p1 of
    Infinity  $\Rightarrow$  p2
  | Point x1 y1  $\Rightarrow$  (case p2 of
    Infinity  $\Rightarrow$  p1
  | Point x2 y2  $\Rightarrow$ 
    if x1 = x2 then
      if y1 = - y2 then Infinity
    else
      let
        l =  $(3 * x1^2 + a) / (2 * y1)$ ;
        x3 =  $l^2 - 2 * x1$ 
      in
        Point x3  $(- y1 - l * (x3 - x1))$ 
    else
```

```

let
  l = (y2 - y1) / (x2 - x1);
  x3 = l ^ 2 - x1 - x2
in
  Point x3 (- y1 - l * (x3 - x1)))

```

**definition** *opp* :: 'a point  $\Rightarrow$  'a point **where**

```

opp p = (case p of
  Infinity  $\Rightarrow$  Infinity
  | Point x y  $\Rightarrow$  Point x (- y))

```

**end**

**lemma** *on-curve-infinity* [*simp*]: *on-curve a b Infinity*  
**by** (*simp add: on-curve-def*)

**lemma** *opp-Infinity* [*simp*]: *opp Infinity = Infinity*  
**by** (*simp add: opp-def*)

**lemma** *opp-Point*: *opp (Point x y) = Point x (- y)*  
**by** (*simp add: opp-def*)

**lemma** *opp-opp*: *opp (opp p) = p*  
**by** (*simp add: opp-def split: point.split*)

**lemma** *opp-closed*:  
*on-curve a b p  $\implies$  on-curve a b (opp p)*  
**by** (*auto simp add: on-curve-def opp-def power2-eq-square split: point.split*)

**lemma** *curve-elt-opp*:  
**assumes**  $p_1 = \text{Point } x_1 \ y_1$   
**and**  $p_2 = \text{Point } x_2 \ y_2$   
**and** *on-curve a b p<sub>1</sub>*  
**and** *on-curve a b p<sub>2</sub>*  
**and**  $x_1 = x_2$   
**shows**  $p_1 = p_2 \vee p_1 = \text{opp } p_2$   
**proof** –  
**from**  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \langle \text{on-curve a b p}_1 \rangle$   
**have**  $y_1^2 = x_1^3 + a * x_1 + b$   
**by** (*simp-all add: on-curve-def*)  
**moreover from**  $\langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle \text{on-curve a b p}_2 \rangle \langle x_1 = x_2 \rangle$   
**have**  $x_1^3 + a * x_1 + b = y_2^2$   
**by** (*simp-all add: on-curve-def*)  
**ultimately have**  $y_1 = y_2 \vee y_1 = - y_2$   
**by** (*simp add: square-eq-iff power2-eq-square*)  
**with**  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle x_1 = x_2 \rangle$  **show** *?thesis*  
**by** (*auto simp add: opp-def*)  
**qed**

```

lemma add-closed:
  assumes on-curve a b p1 and on-curve a b p2
  shows on-curve a b (add a p1 p2)
proof (cases p1)
  case (Point x1 y1)
  note Point' = this
  show ?thesis
proof (cases p2)
  case (Point x2 y2)
  show ?thesis
proof (cases x1 = x2)
  case True
  note True' = this
  show ?thesis
proof (cases y1 = - y2)
  case True
  with True' Point Point'
  show ?thesis
  by (simp add: on-curve-def add-def)
next
  case False
  note on-curve1 = ⟨on-curve a b p1⟩ [simplified Point' on-curve-def True',
simplified]
  from False True' Point Point' assms
  have y1 ≠ 0 by (auto simp add: on-curve-def)
  with False True' Point Point' assms
  show ?thesis
  apply (simp add: on-curve-def add-def Let-def)
  apply (field on-curve1)
  apply (simp add: two-not-zero)
  done
qed
next
  case False
  note on-curve1 = ⟨on-curve a b p1⟩ [simplified Point' on-curve-def, simplified]
  note on-curve2 = ⟨on-curve a b p2⟩ [simplified Point on-curve-def, simplified]
  from assms show ?thesis
  apply (simp add: on-curve-def add-def Let-def False Point Point')
  apply (field on-curve1 on-curve2)
  apply (simp add: False [symmetric])
  done
qed
next
  case Infinity
  with Point ⟨on-curve a b p1⟩ show ?thesis
  by (simp add: add-def)
qed
next

```

```

case Infinity
with ⟨on-curve a b p₂⟩ show ?thesis
  by (simp add: add-def)
qed

lemma add-case [consumes 2, case-names InfL InfR Opp Tan Gen]:
  assumes p: on-curve a b p
  and q: on-curve a b q
  and R1:  $\bigwedge p. P \text{ Infinity } p \ p$ 
  and R2:  $\bigwedge p. P \ p \ \text{Infinity } p$ 
  and R3:  $\bigwedge p. \text{on-curve } a \ b \ p \implies P \ p \ (\text{opp } p) \ \text{Infinity}$ 
  and R4:  $\bigwedge p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ l.$ 
     $p_1 = \text{Point } x_1 \ y_1 \implies p_2 = \text{Point } x_2 \ y_2 \implies$ 
     $p_2 = \text{add } a \ p_1 \ p_1 \implies y_1 \neq 0 \implies$ 
     $l = (3 * x_1^2 + a) / (2 * y_1) \implies$ 
     $x_2 = l^2 - 2 * x_1 \implies$ 
     $y_2 = -y_1 - l * (x_2 - x_1) \implies$ 
     $P \ p_1 \ p_1 \ p_2$ 
  and R5:  $\bigwedge p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ p_3 \ x_3 \ y_3 \ l.$ 
     $p_1 = \text{Point } x_1 \ y_1 \implies p_2 = \text{Point } x_2 \ y_2 \implies p_3 = \text{Point } x_3 \ y_3 \implies$ 
     $p_3 = \text{add } a \ p_1 \ p_2 \implies x_1 \neq x_2 \implies$ 
     $l = (y_2 - y_1) / (x_2 - x_1) \implies$ 
     $x_3 = l^2 - x_1 - x_2 \implies$ 
     $y_3 = -y_1 - l * (x_3 - x_1) \implies$ 
     $P \ p_1 \ p_2 \ p_3$ 
  shows  $P \ p \ q \ (\text{add } a \ p \ q)$ 
proof (cases p)
  case Infinity
  then show ?thesis
    by (simp add: add-def R1)
next
case (Point x₁ y₁)
note Point' = this
show ?thesis
proof (cases q)
  case Infinity
  with Point show ?thesis
    by (simp add: add-def R2)
next
case (Point x₂ y₂)
show ?thesis
proof (cases x₁ = x₂)
  case True
  note True' = this
  show ?thesis
  proof (cases y₁ = - y₂)
    case True
    with p Point Point' True' R3 [of p] show ?thesis
      by (simp add: add-def opp-def)

```

```

next
  case False
  from True' Point Point' p q have  $(y_1 - y_2) * (y_1 + y_2) = 0$ 
    by (simp add: on-curve-def ring-distrib power2-eq-square)
  with False have  $y_1 = y_2$ 
    by (simp add: eq-neg-iff-add-eq-0)
  with False True' Point Point' show ?thesis
    apply simp
    apply (rule R4)
    apply (auto simp add: add-def Let-def)
    done
qed
next
  case False
  with Point Point' show ?thesis
    apply -
    apply (rule R5)
    apply (auto simp add: add-def Let-def)
    done
qed
qed
qed

```

**lemma** *eq-opp-is-zero*:  $((x::'a::ell-field) = - x) = (x = 0)$

**proof**

```

  assume  $x = - x$ 
  have  $2 * x = x + x$  by simp
  also from  $\langle x = - x \rangle$ 
  have  $\dots = - x + x$  by simp
  also have  $\dots = 0$  by simp
  finally have  $2 * x = 0$  .
  with two-not-zero [where 'a='a] show  $x = 0$ 
    by simp
qed simp

```

**lemma** *add-casew* [*consumes 2, case-names InfL InfR Opp Gen*]:

```

  assumes p: on-curve a b p
  and q: on-curve a b q
  and R1:  $\bigwedge p. P \text{ Infinity } p$ 
  and R2:  $\bigwedge p. P p \text{ Infinity } p$ 
  and R3:  $\bigwedge p. \text{on-curve } a \ b \ p \implies P \ p \ (\text{opp } p) \ \text{Infinity}$ 
  and R4:  $\bigwedge p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ p_3 \ x_3 \ y_3 \ l.$ 
     $p_1 = \text{Point } x_1 \ y_1 \implies p_2 = \text{Point } x_2 \ y_2 \implies p_3 = \text{Point } x_3 \ y_3 \implies$ 
     $p_3 = \text{add } a \ p_1 \ p_2 \implies p_1 \neq \text{opp } p_2 \implies$ 
     $x_1 = x_2 \wedge y_1 = y_2 \wedge l = (3 * x_1^2 + a) / (2 * y_1) \vee$ 
     $x_1 \neq x_2 \wedge l = (y_2 - y_1) / (x_2 - x_1) \implies$ 
     $x_3 = l^2 - x_1 - x_2 \implies$ 
     $y_3 = - y_1 - l * (x_3 - x_1) \implies$ 
     $P \ p_1 \ p_2 \ p_3$ 

```

```

shows  $P p q$  (add a p q)
using p q
apply (rule add-case)
apply (rule R1)
apply (rule R2)
apply (rule R3)
apply assumption
apply (rule R4)
apply assumption+
apply (simp add: opp-def eq-opp-is-zero)
apply simp
apply simp
apply simp
apply (rule R4)
apply assumption+
apply (simp add: opp-def)
apply simp
apply assumption+
done

```

**definition**

*is-tangent*  $p q = (p \neq \text{Infinity} \wedge p = q \wedge p \neq \text{opp } q)$

**definition**

*is-generic*  $p q =$   
 $(p \neq \text{Infinity} \wedge q \neq \text{Infinity} \wedge$   
 $p \neq q \wedge p \neq \text{opp } q)$

**lemma** *diff-neq0*:

$(a::'a::\text{ring}) \neq b \implies a - b \neq 0$   
 $a \neq b \implies b - a \neq 0$   
**by** *simp-all*

**lemma** *minus2-not0*:  $(-2::'a::\text{ell-field}) \neq 0$

**using** *two-not-zero* [**where** 'a='a]  
**by** *simp*

**lemmas** [*simp*] = *minus2-not0* [*simplified*]

**declare** *two-not-zero* [*simplified*, *simp add*]

**lemma** *spec1-assoc*:

**assumes**  $p_1$ : *on-curve* a b  $p_1$   
**and**  $p_2$ : *on-curve* a b  $p_2$   
**and**  $p_3$ : *on-curve* a b  $p_3$   
**and** *is-generic*  $p_1 p_2$   
**and** *is-generic*  $p_2 p_3$   
**and** *is-generic* (add a  $p_1 p_2$ )  $p_3$   
**and** *is-generic*  $p_1$  (add a  $p_2 p_3$ )

```

shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
using p1 p2 assms
proof (induct rule: add-case)
  case InfL
  show ?case by (simp add: add-def)
next
  case InfR
  show ?case by (simp add: add-def)
next
  case Opp
  then show ?case by (simp add: is-generic-def)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p1 x1 y1 p2 x2 y2 p4 x4 y4 l)
  with ⟨on-curve a b p2⟩ ⟨on-curve a b p3⟩
  show ?case
  proof (induct rule: add-case)
    case InfL
    then show ?case by (simp add: is-generic-def)
  next
    case InfR
    then show ?case by (simp add: is-generic-def)
  next
    case Opp
    then show ?case by (simp add: is-generic-def)
  next
    case Tan
    then show ?case by (simp add: is-generic-def)
  next
    case (Gen p2 x2' y2' p3 x3 y3 p5 x5 y5 l1)
    from ⟨on-curve a b p2⟩ ⟨on-curve a b p3⟩ ⟨p5 = add a p2 p3⟩
    have on-curve a b p5 by (simp add: add-closed)
    with ⟨on-curve a b p1⟩ show ?case using Gen [simplified ⟨p2 = Point x2' y2'⟩]
    proof (induct rule: add-case)
      case InfL
      then show ?case by (simp add: is-generic-def)
    next
      case InfR
      then show ?case by (simp add: is-generic-def)
    next
      case (Opp p)
      from ⟨is-generic p (opp p)⟩
      show ?case by (simp add: is-generic-def opp-opp)
    next
      case Tan
      then show ?case by (simp add: is-generic-def)
    next

```



```

case (Gen p1 x1' y1' p5' x5' y5' p6 x6 y6 l2)
from ⟨on-curve a b p1⟩ ⟨on-curve a b (Point x2' y2')⟩
  ⟨p4 = add a p1 (Point x2' y2')⟩
have on-curve a b p4 by (simp add: add-closed)
then show ?case using ⟨on-curve a b p3⟩ Gen
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
  case (Opp p)
  from ⟨is-generic p (opp p)⟩
  show ?case by (simp add: is-generic-def opp-opp)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p4' x4' y4' p3' x3' y3' p7 x7 y7 l3)
  from ⟨p4' = Point x4' y4'⟩ ⟨p4' = Point x4 y4⟩
  have p4: x4' = x4 y4' = y4 by simp-all
  from ⟨p3' = Point x3' y3'⟩ ⟨p3' = Point x3 y3⟩
  have p3: x3' = x3 y3' = y3 by simp-all
  from ⟨p1 = Point x1' y1'⟩ ⟨p1 = Point x1 y1⟩
  have p1: x1' = x1 y1' = y1 by simp-all
  from ⟨p5' = Point x5' y5'⟩ ⟨p5' = Point x5 y5⟩
  have p5: x5' = x5 y5' = y5 by simp-all
  from ⟨Point x2' y2' = Point x2 y2⟩
  have p2: x2' = x2 y2' = y2 by simp-all
  note ps = p1 p2 p3 p4 p5
  note ps' =
    ⟨on-curve a b p1⟩ [simplified ⟨p1 = Point x1 y1⟩ on-curve-def, simplified]
    ⟨on-curve a b p2⟩ [simplified ⟨p2 = Point x2 y2⟩ on-curve-def, simplified]
    ⟨on-curve a b p3⟩ [simplified ⟨p3 = Point x3 y3⟩ on-curve-def, simplified]
show ?case
  apply (simp add: ⟨p6 = Point x6 y6⟩ ⟨p7 = Point x7 y7⟩)
  apply (simp only: ps
    ⟨x6 = l22 - x1' - x5'⟩ ⟨x7 = l32 - x4' - x3'⟩
    ⟨y6 = - y1' - l2 * (x6 - x1')⟩ ⟨y7 = - y4' - l3 * (x7 - x4')⟩
    ⟨l2 = (y5' - y1') / (x5' - x1')⟩ ⟨l3 = (y3' - y4') / (x3' - x4')⟩
    ⟨l1 = (y3 - y2') / (x3 - x2')⟩ ⟨l = (y2 - y1) / (x2 - x1)⟩
    ⟨x5 = l12 - x2' - x3⟩ ⟨y5 = - y2' - l1 * (x5 - x2')⟩
    ⟨x4 = l2 - x1 - x2⟩ ⟨y4 = - y1 - l * (x4 - x1)⟩)
  apply (rule conjI)
  apply (field ps')
  apply (rule conjI)
  apply (simp add: ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩, symmetric])
  apply (rule conjI)

```

```

    apply (rule notI)
    apply (ring (prems) ps'(1-2))
    apply (cut-tac ⟨x1' ≠ x5'⟩ [simplified ⟨x5' = x5⟩ ⟨x1' = x1⟩ ⟨x5 = l12 -
x2' - x3⟩
      ⟨l1 = (y3 - y2') / (x3 - x2')⟩ ⟨y2' = y2⟩ ⟨x2' = x2⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field ps'(1-2))
    apply (simp add: ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩, symmetric])
    apply (rule conjI)
    apply (simp add: ⟨x1 ≠ x2⟩ [symmetric])
    apply (rule notI)
    apply (ring (prems) ps'(1-2))
    apply (cut-tac ⟨x4' ≠ x3'⟩ [simplified ⟨x4' = x4⟩ ⟨x3' = x3⟩ ⟨x4 = l2 - x1
- x2⟩
      ⟨l = (y2 - y1) / (x2 - x1)⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field ps'(1-2))
    apply (simp add: ⟨x1 ≠ x2⟩ [symmetric])
    apply (field ps')
    apply (rule conjI)
    apply (rule notI)
    apply (ring (prems) ps'(1-2))
    apply (cut-tac ⟨x1' ≠ x5'⟩ [simplified ⟨x5' = x5⟩ ⟨x1' = x1⟩ ⟨x5 = l12 -
x2' - x3⟩
      ⟨l1 = (y3 - y2') / (x3 - x2')⟩ ⟨y2' = y2⟩ ⟨x2' = x2⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field ps'(1-2))
    apply (simp add: ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩, symmetric])
    apply (rule conjI)
    apply (simp add: ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩, symmetric])
    apply (rule conjI)
    apply (rule notI)
    apply (ring (prems) ps'(1-2))
    apply (cut-tac ⟨x4' ≠ x3'⟩ [simplified ⟨x4' = x4⟩ ⟨x3' = x3⟩ ⟨x4 = l2 - x1
- x2⟩
      ⟨l = (y2 - y1) / (x2 - x1)⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field ps'(1-2))
    apply (simp-all add: ⟨x1 ≠ x2⟩ [symmetric])
  done
qed
qed
qed
qed

```

```

lemma spec2-assoc:
  assumes p1: on-curve a b p1
  and p2: on-curve a b p2
  and p3: on-curve a b p3
  and is-generic p1 p2
  and is-tangent p2 p3
  and is-generic (add a p1 p2) p3
  and is-generic p1 (add a p2 p3)
  shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
  using p1 p2 assms
proof (induct rule: add-case)
  case InfL
  show ?case by (simp add: add-def)
next
  case InfR
  show ?case by (simp add: add-def)
next
  case Opp
  then show ?case by (simp add: is-generic-def)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p1 x1 y1 p2 x2 y2 p4 x4 y4 l)
  with ⟨on-curve a b p2⟩ ⟨on-curve a b p3⟩
  show ?case
  proof (induct rule: add-case)
    case InfL
    then show ?case by (simp add: is-generic-def)
  next
    case InfR
    then show ?case by (simp add: is-generic-def)
  next
    case Opp
    then show ?case by (simp add: is-generic-def)
  next
    case (Tan p2 x2' y2' p5 x5 y5 l1)
    from ⟨on-curve a b p2⟩ ⟨p5 = add a p2 p2⟩
    have on-curve a b p5 by (simp add: add-closed)
    with ⟨on-curve a b p1⟩ show ?case using Tan
  proof (induct rule: add-case)
    case InfL
    then show ?case by (simp add: is-generic-def)
  next
    case InfR
    then show ?case by (simp add: is-generic-def)
  next
    case (Opp p)
    from ⟨is-generic p (opp p)⟩ ⟨on-curve a b p⟩

```

```

  show ?case by (simp add: is-generic-def opp-opp)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p1 x1' y1' p5' x5' y5' p6 x6 y6 l2)
  from ⟨on-curve a b p1⟩ ⟨on-curve a b p2⟩ ⟨p4 = add a p1 p2⟩
  have on-curve a b p4 by (simp add: add-closed)
  then show ?case using ⟨on-curve a b p2⟩ Gen
  proof (induct rule: add-case)
    case Inl
    then show ?case by (simp add: is-generic-def)
  next
    case Inr
    then show ?case by (simp add: is-generic-def)
  next
  case (Opp p)
  from ⟨is-generic p (opp p)⟩
  show ?case by (simp add: is-generic-def opp-opp)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p4' x4' y4' p3' x3' y3' p7 x7 y7 l3)
  from
    ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
    ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
  have
    y1:  $y_1^2 = x_1^3 + a * x_1 + b$  and
    y2:  $y_2^2 = x_2^3 + a * x_2 + b$ 
    by (simp-all add: on-curve-def)
  from
    ⟨p5' = Point x5' y5'⟩
    ⟨p5' = Point x5 y5⟩
    ⟨p4' = Point x4' y4'⟩
    ⟨p4' = Point x4 y4⟩
    ⟨p3' = Point x2' y2'⟩
    ⟨p3' = Point x2 y2⟩
    ⟨p3' = Point x3' y3'⟩
    ⟨p1 = Point x1' y1'⟩
    ⟨p1 = Point x1 y1⟩
  have ps:
    x5' = x5 y5' = y5
    x4' = x4 y4' = y4 x3' = x2 y3' = y2 x2' = x2 y2' = y2
    x1' = x1 y1' = y1
  by simp-all
  show ?case
  apply (simp add: ⟨p6 = Point x6 y6⟩ ⟨p7 = Point x7 y7⟩)
  apply (simp only: ps)

```

$\langle x_7 = l_3^2 - x_4' - x_3' \rangle$   
 $\langle y_7 = -y_4' - l_3 * (x_7 - x_4') \rangle$   
 $\langle l_3 = (y_3' - y_4') / (x_3' - x_4') \rangle$   
 $\langle x_6 = l_2^2 - x_1' - x_5' \rangle$   
 $\langle y_6 = -y_1' - l_2 * (x_6 - x_1') \rangle$   
 $\langle l_2 = (y_5' - y_1') / (x_5' - x_1') \rangle$   
 $\langle x_5 = l_1^2 - 2 * x_2' \rangle$   
 $\langle y_5 = -y_2' - l_1 * (x_5 - x_2') \rangle$   
 $\langle l_1 = (3 * x_2'^2 + a) / (2 * y_2') \rangle$   
 $\langle x_4 = l^2 - x_1 - x_2 \rangle$   
 $\langle y_4 = -y_1 - l * (x_4 - x_1) \rangle$   
 $\langle l = (y_2 - y_1) / (x_2 - x_1) \rangle$   
**apply** (*rule conjI*)  
**apply** (*field y1 y2*)  
**apply** (*intro conjI*)  
**apply** (*simp add:  $\langle y_2' \neq 0 \rangle$  [simplified  $\langle y_2' = y_2 \rangle$ ]*)  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE [OF  $\langle x_1' \neq x_5' \rangle$  [simplified*  
 $\langle x_5 = l_1^2 - 2 * x_2' \rangle$   
 $\langle l_1 = (3 * x_2'^2 + a) / (2 * y_2') \rangle$   
 $\langle x_1' = x_1 \rangle \langle x_2' = x_2 \rangle \langle y_2' = y_2 \rangle \langle x_5' = x_5 \rangle]$ )  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add:  $\langle y_2' \neq 0 \rangle$  [simplified  $\langle y_2' = y_2 \rangle$ ]*)  
**apply** (*simp add:  $\langle x_1 \neq x_2 \rangle$  [symmetric]*)  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE [OF  $\langle x_4' \neq x_3' \rangle$  [simplified*  
 $\langle x_4 = l^2 - x_1 - x_2 \rangle$   
 $\langle l = (y_2 - y_1) / (x_2 - x_1) \rangle$   
 $\langle x_4' = x_4 \rangle \langle x_3' = x_2 \rangle]$ )  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add:  $\langle x_1 \neq x_2 \rangle$  [symmetric]*)  
**apply** (*field y1 y2*)  
**apply** (*intro conjI*)  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE [OF  $\langle x_1' \neq x_5' \rangle$  [simplified*  
 $\langle x_5 = l_1^2 - 2 * x_2' \rangle$   
 $\langle l_1 = (3 * x_2'^2 + a) / (2 * y_2') \rangle$   
 $\langle x_1' = x_1 \rangle \langle x_2' = x_2 \rangle \langle y_2' = y_2 \rangle \langle x_5' = x_5 \rangle]$ )  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add:  $\langle y_2' \neq 0 \rangle$  [simplified  $\langle y_2' = y_2 \rangle$ ]*)  
**apply** (*simp add:  $\langle y_2' \neq 0 \rangle$  [simplified  $\langle y_2' = y_2 \rangle$ ]*)  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)

```

apply (rule notE [OF ⟨ $x_4' \neq x_3'$ ⟩ [simplified
  ⟨ $x_4 = l^2 - x_1 - x_2$ ⟩
  ⟨ $l = (y_2 - y_1) / (x_2 - x_1)$ ⟩
  ⟨ $x_4' = x_4$ ⟩ ⟨ $x_3' = x_2$ ⟩]])
apply (rule sym)
apply (field y1 y2)
apply (simp-all add: ⟨ $x_1 \neq x_2$ ⟩ [symmetric])
done
qed
qed
next
  case (Gen  $p_3 x_3 y_3 p_5 x_5 y_5 p_6 x_6 y_6 l_1$ )
  then show ?case by (simp add: is-tangent-def)
qed
qed

```

lemma spec3-assoc:

```

assumes  $p_1$ : on-curve a b  $p_1$ 
and  $p_2$ : on-curve a b  $p_2$ 
and  $p_3$ : on-curve a b  $p_3$ 
and is-generic  $p_1 p_2$ 
and is-tangent  $p_2 p_3$ 
and is-generic (add a  $p_1 p_2$ )  $p_3$ 
and is-tangent  $p_1$  (add a  $p_2 p_3$ )
shows add a  $p_1$  (add a  $p_2 p_3$ ) = add a (add a  $p_1 p_2$ )  $p_3$ 
using  $p_1 p_2$  assms
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
  case Opp
  then show ?case by (simp add: is-generic-def)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen  $p_1 x_1 y_1 p_2 x_2 y_2 p_4 x_4 y_4 l$ )
  with ⟨on-curve a b  $p_2$ ⟩ ⟨on-curve a b  $p_3$ ⟩
  show ?case
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next

```

```

case Opp
then show ?case by (simp add: is-tangent-def opp-opp)
next
case (Tan p2 x2' y2' p5 x5 y5 l1)
from ⟨on-curve a b p2⟩ ⟨p5 = add a p2 p2⟩
have on-curve a b p5 by (simp add: add-closed)
with ⟨on-curve a b p1⟩ show ?case using Tan
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
case Opp
then show ?case by (simp add: is-tangent-def opp-opp)
next
case (Tan p1 x1' y1' p6 x6 y6 l2)
from ⟨on-curve a b p1⟩ ⟨on-curve a b p2⟩ ⟨p4 = add a p1 p2⟩
have on-curve a b p4 by (simp add: add-closed)
then show ?case using ⟨on-curve a b p2⟩ Tan
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
case (Opp p)
from ⟨is-generic p (opp p)⟩
show ?case by (simp add: is-generic-def opp-opp)
next
case Tan
then show ?case by (simp add: is-generic-def)
next
case (Gen p4' x4' y4' p2' x2'' y2'' p7 x7 y7 l3)
from
  ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
  ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
have
  y1:  $y_1^2 = x_1^3 + a * x_1 + b$  and
  y2:  $y_2^2 = x_2^3 + a * x_2 + b$ 
by (simp-all add: on-curve-def)
from
  ⟨p4' = Point x4' y4'⟩
  ⟨p4' = Point x4 y4⟩
  ⟨p2' = Point x2' y2'⟩
  ⟨p2' = Point x2 y2⟩
  ⟨p2' = Point x2'' y2''⟩

```

```

  ⟨p1 = Point x1' y1'⟩
  ⟨p1 = Point x1 y1⟩
  ⟨p1 = Point x5 y5⟩
have ps:
  x4' = x4 y4' = y4 x2' = x2 y2' = y2 x2'' = x2 y2'' = y2
  x1' = x5 y1' = y5 x1 = x5 y1 = y5
  by simp-all
note qs =
  ⟨x7 = l3 ^ 2 - x4' - x2''⟩
  ⟨y7 = - y4' - l3 * (x7 - x4')⟩
  ⟨l3 = (y2'' - y4') / (x2'' - x4')⟩
  ⟨x6 = l2 ^ 2 - 2 * x1'⟩
  ⟨y6 = - y1' - l2 * (x6 - x1')⟩
  ⟨x5 = l1 ^ 2 - 2 * x2'⟩
  ⟨y5 = - y2' - l1 * (x5 - x2')⟩
  ⟨l1 = (3 * x2' ^ 2 + a) / (2 * y2')⟩
  ⟨l2 = (3 * x1' ^ 2 + a) / (2 * y1')⟩
  ⟨x4 = l ^ 2 - x1 - x2⟩
  ⟨y4 = - y1 - l * (x4 - x1)⟩
  ⟨l = (y2 - y1) / (x2 - x1)⟩
from ⟨y2' ≠ 0⟩ ⟨y2' = y2⟩
have 2 * y2 ≠ 0 by simp
show ?case
  apply (simp add: ⟨p6 = Point x6 y6⟩ ⟨p7 = Point x7 y7⟩)
  apply (simp only: ps qs)
  apply (rule conjI)
  apply (field y2)
  apply (intro conjI)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨y1' ≠ 0⟩])
  apply (simp only: ps qs)
  apply field
  apply (rule ⟨2 * y2 ≠ 0⟩)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨x1 ≠ x2⟩])
  apply (rule sym)
  apply (simp only: ps qs)
  apply field
  apply (rule ⟨2 * y2 ≠ 0⟩)
  apply (rule ⟨2 * y2 ≠ 0⟩)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨x4' ≠ x2''⟩])
  apply (rule sym)
  apply (simp only: ps qs)
  apply field
  apply (intro conjI)

```



```

    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (erule thin-rl)
    apply (rule notI)
    apply (ring (prems))
    apply (rule notE [OF ⟨x1 ≠ x2⟩])
    apply (rule sym)
    apply (simp only: ps qs)
    apply field
    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (field y2)
    apply (intro conjI)
    apply (rule notI)
    apply (ring (prems))
    apply (rule notE [OF ⟨y1' ≠ 0⟩])
    apply (simp only: ps qs)
    apply field
    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (rule notI)
    apply (ring (prems))
    apply (rule notE [OF ⟨x4' ≠ x2''⟩])
    apply (rule sym)
    apply (simp only: ps qs)
    apply field
    apply (erule thin-rl)
    apply (rule conjI)
    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (rule notI)
    apply (ring (prems))
    apply (rule notE [OF ⟨x1 ≠ x2⟩])
    apply (rule sym)
    apply (simp only: ps qs)
    apply field
    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (rule ⟨2 * y2 ≠ 0⟩)
    apply (rule notI)
    apply (ring (prems))
    apply (rule notE [OF ⟨x1 ≠ x2⟩])
    apply (rule sym)
    apply (simp only: ps qs)
    apply field
    apply (rule ⟨2 * y2 ≠ 0⟩)
done
qed
next
case Gen
  then show ?case by (simp add: is-tangent-def)
qed
next
case Gen

```

```

    then show ?case by (simp add: is-tangent-def)
  qed
qed

lemma add-0-l: add a Infinity p = p
  by (simp add: add-def)

lemma add-0-r: add a p Infinity = p
  by (simp add: add-def split: point.split)

lemma add-opp: on-curve a b p  $\implies$  add a p (opp p) = Infinity
  by (simp add: add-def opp-def on-curve-def split: point.split-asm)

lemma add-comm:
  assumes on-curve a b p1 on-curve a b p2
  shows add a p1 p2 = add a p2 p1
proof (cases p1)
  case Infinity
  then show ?thesis by (simp add: add-0-l add-0-r)
next
  case (Point x1 y1)
  note Point' = this
  with ⟨on-curve a b p1⟩
  have y1: y12 = x13 + a * x1 + b
    by (simp add: on-curve-def)
  show ?thesis
  proof (cases p2)
    case Infinity
    then show ?thesis by (simp add: add-0-l add-0-r)
  next
    case (Point x2 y2)
    with ⟨on-curve a b p2⟩
    have y2: y22 = x23 + a * x2 + b
      by (simp add: on-curve-def)
    show ?thesis
    proof (cases x1 = x2)
      case True
      show ?thesis
      proof (cases y1 = - y2)
        case True
        with Point Point' ⟨x1 = x2⟩ show ?thesis
          by (simp add: add-def)
        next
          case False
          with y1 y2 [symmetric] ⟨x1 = x2⟩ Point Point'
          show ?thesis
            by (simp add: power2-eq-square square-eq-iff)
      qed
    qed
  next
    case False
    with y1 y2 [symmetric] ⟨x1 = x2⟩ Point Point'
    show ?thesis
      by (simp add: power2-eq-square square-eq-iff)
  qed
next

```

```

case False
with Point Point' show ?thesis
  apply (simp add: add-def Let-def)
  apply (rule conjI)
  apply field
  apply simp
  apply field
  apply simp
done
qed
qed
qed

lemma uniq-opp:
  assumes add a p1 p2 = Infinity
  shows p2 = opp p1
  using assms
  by (auto simp add: add-def opp-def Let-def
    split: point.split-asm if-split-asm)

lemma uniq-zero:
  assumes ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and add: add a p1 p2 = p2
  shows p1 = Infinity
  using p1 p2 assms
proof (induct rule: add-case)
  case InfL
  show ?case ..
next
  case InfR
  then show ?case by simp
next
  case Opp
  then show ?case by (simp add: opp-def split: point.split-asm)
next
  case (Tan p1 x1 y1 p2 x2 y2 l)
  from <p1 = Point x1 y1> <p2 = Point x2 y2> <p2 = p1>
  have x2 = x1 y2 = y1 by simp-all
  with <y2 = - y1 - l * (x2 - x1)> <y1 ≠ 0>
  have - y1 = y1 by simp
  with <y1 ≠ 0>
  show ?case by simp
next
  case (Gen p1 x1 y1 p2 x2 y2 p3 x3 y3 l)
  then have y1: y1 ^ 2 = x1 ^ 3 + a * x1 + b
  and y2: y2 ^ 2 = x2 ^ 3 + a * x2 + b
  by (simp-all add: on-curve-def)

```

```

from ⟨ $p_3 = p_2$ ⟩ ⟨ $p_2 = \text{Point } x_2 \ y_2$ ⟩ ⟨ $p_3 = \text{Point } x_3 \ y_3$ ⟩
have  $ps: x_3 = x_2 \ y_3 = y_2$  by simp-all
with ⟨ $y_3 = -y_1 - l * (x_3 - x_1)$ ⟩
have  $y_2 = -y_1 - l * (x_2 - x_1)$  by simp
also from ⟨ $l = (y_2 - y_1) / (x_2 - x_1)$ ⟩ ⟨ $x_1 \neq x_2$ ⟩
have  $l * (x_2 - x_1) = y_2 - y_1$ 
  by simp
also have  $-y_1 - (y_2 - y_1) = (-y_1 + y_1) + -y_2$ 
  by simp
finally have  $y_2 = 0$  by simp
with ⟨ $p_2 = \text{Point } x_2 \ y_2$ ⟩ ⟨on-curve a b p2⟩
have  $x_2^3 = -(a * x_2 + b)$ 
  by (simp add: on-curve-def eq-neg-iff-add-eq-0 add.assoc del: minus-add-distrib)
from ⟨ $x_3 = l^2 - x_1 - x_2$ ⟩ ⟨ $x_3 = x_2$ ⟩
have  $l^2 - x_1 - x_2 - x_2 = x_2 - x_2$  by simp
then have  $l^2 - x_1 - 2 * x_2 = 0$  by simp
then have  $x_2 * (l^2 - x_1 - 2 * x_2) = x_2 * 0$  by simp
then have  $(x_2 - x_1) * (2 * a * x_2 + 3 * b) = 0$ 
  apply (simp only: l = (y_2 - y_1) / (x_2 - x_1) y_2 = 0)
  apply (field (prems) y1 x2)
  apply (ring y1 x2)
  apply (simp add: x_1 \neq x_2 [symmetric])
  done
with ⟨ $x_1 \neq x_2$ ⟩ have  $2 * a * x_2 + 3 * b = 0$  by simp
then have  $2 * a * x_2 = -(3 * b)$ 
  by (simp add: eq-neg-iff-add-eq-0)
from  $y_2$  [symmetric] ⟨ $y_2 = 0$ ⟩
have  $(-(2 * a))^3 * (x_2^3 + a * x_2 + b) = 0$ 
  by simp
then have  $b * (4 * a^3 + 27 * b^2) = 0$ 
  apply (ring (prems) 2 * a * x_2 = -(3 * b))
  apply (ring 2 * a * x_2 = -(3 * b))
  done
with  $ab$  have  $b = 0$  by (simp add: nonsingular-def)
with ⟨ $2 * a * x_2 + 3 * b = 0$ ⟩  $ab$ 
have  $x_2 = 0$  by (simp add: nonsingular-def)
from ⟨ $l^2 - x_1 - 2 * x_2 = 0$ ⟩
show ?case
  apply (simp add: x_2 = 0 y_2 = 0 l = (y_2 - y_1) / (x_2 - x_1))
  apply (field (prems) y1 b = 0)
  apply (insert ab b = 0 x_1 \neq x_2 x_2 = 0)
  apply (simp add: nonsingular-def)
  apply simp
  done

```

qed

lemma *opp-add*:

```

assumes  $p_1: \text{on-curve } a \ b \ p_1$ 
and  $p_2: \text{on-curve } a \ b \ p_2$ 

```

```

  shows  $opp (add a p_1 p_2) = add a (opp p_1) (opp p_2)$ 
proof (cases p1)
  case Infinity
  then show ?thesis by (simp add: add-def opp-def)
next
  case (Point x1 y1)
  show ?thesis
  proof (cases p2)
  case Infinity
  with ⟨p1 = Point x1 y1⟩ show ?thesis
  by (simp add: add-def opp-def)
  next
  case (Point x2 y2)
  with ⟨p1 = Point x1 y1⟩ p1 p2
  have x1 ^ 3 + a * x1 + b = y1 ^ 2
  x2 ^ 3 + a * x2 + b = y2 ^ 2
  by (simp-all add: on-curve-def)
  with Point ⟨p1 = Point x1 y1⟩ show ?thesis
  apply (cases x1 = x2)
  apply (cases y1 = - y2)
  apply (simp add: add-def opp-def Let-def)
  apply (simp add: add-def opp-def Let-def trans [OF minus-equation-iff eq-commute])
  apply (simp add: add-def opp-def Let-def)
  apply (rule conjI)
  apply field
  apply simp
  apply field
  apply simp
  done
qed
qed

```

```

lemma compat-add-opp:
  assumes p1: on-curve a b p1
  and p2: on-curve a b p2
  and add a p1 p2 = add a p1 (opp p2)
  and p1 ≠ opp p1
  shows p2 = opp p2
  using p1 p2 assms
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: add-0-l)
  next
  case InfR
  then show ?case by (simp add: opp-def add-0-r)
  next
  case (Opp p)
  then have add a p p = Infinity by (simp add: opp-opp)
  then have p = opp p by (rule uniq-opp)

```

```

with ⟨p ≠ opp p⟩ show ?case ..
next
case (Tan p1 x1 y1 p2 x2 y2 l)
then have add a p1 p1 = Infinity
  by (simp add: add-opp)
then have p1 = opp p1 by (rule uniq-opp)
with ⟨p1 ≠ opp p1⟩ show ?case ..
next
case (Gen p1 x1 y1 p2 x2 y2 p3 x3 y3 l)
have (2::'a) * 2 ≠ 0
  by (simp only: mult-eq-0-iff) simp
then have (4::'a) ≠ 0 by simp
from Gen have ((- y2 - y1) / (x2 - x1)) ^ 2 - x1 - x2 =
  ((y2 - y1) / (x2 - x1)) ^ 2 - x1 - x2
  by (simp add: add-def opp-def Let-def)
then show ?case
  apply (field (prems))
  apply (insert ⟨p1 ≠ opp p1⟩
    ⟨p1 = Point x1 y1⟩ ⟨p2 = Point x2 y2⟩ ⟨4 ≠ 0⟩)[1]
  apply (simp add: opp-def eq-neg-iff-add-eq-0)
  apply (simp add: ⟨x1 ≠ x2⟩ [symmetric])
done

```

qed

lemma compat-add-triple:

```

assumes ab: nonsingular a b
and p: on-curve a b p
and p ≠ opp p
and add a p p ≠ opp p
shows add a (add a p p) (opp p) = p
using add-closed [OF p p] opp-closed [OF p] assms
proof (induct add a p p opp p rule: add-case)
case InfL
from ⟨p ≠ opp p⟩ uniq-opp [OF ⟨Infinity = add a p p⟩ [symmetric]]
show ?case ..
next
case InfR
then show ?case by (simp add: opp-def split: point.split-asm)
next
case Opp
then have opp (opp (add a p p)) = opp (opp p) by simp
then have add a p p = p by (simp add: opp-opp)
with uniq-zero [OF ab p p] ⟨p ≠ opp p⟩
show ?case by (simp add: opp-def)
next
case Tan
then show ?case by simp
next
case (Gen x1 y1 x2 y2 p3 x3 y3 l)

```

```

from ⟨opp  $p = \text{Point } x_2 \ y_2$ ⟩
have  $p = \text{Point } x_2 \ (- \ y_2)$ 
  by (auto simp add: opp-def split: point.split-asm)
with ⟨add a p p = Point  $x_1 \ y_1$ ⟩ [symmetric]
obtain  $l'$  where  $l'$ :
   $l' = (3 * x_2^2 + a) / (2 * - \ y_2)$ 
  and  $xy: x_1 = l'^2 - 2 * x_2$ 
   $y_1 = - \ (- \ y_2) - l' * (x_1 - x_2)$ 
  and  $y_2: - \ y_2 \neq - \ (- \ y_2)$ 
  by (simp add: add-def Let-def split: if-split-asm)
have  $x_3 = x_2$ 
apply (simp add: xy
  ⟨ $l = (y_2 - y_1) / (x_2 - x_1)$ ⟩ ⟨ $x_3 = l^2 - x_1 - x_2$ ⟩)
apply field
apply (insert ⟨ $x_1 \neq x_2$ ⟩)
apply (simp add: xy)
done
then have  $p_3 = p \vee p_3 = \text{opp } p$ 
  by (rule curve-elt-opp [OF ⟨ $p_3 = \text{Point } x_3 \ y_3$ ⟩ ⟨ $p = \text{Point } x_2 \ (- \ y_2)$ ⟩]
  add-closed [OF add-closed [OF p p] opp-closed [OF p],
  folded ⟨ $p_3 = \text{add a (add a p p) (opp p)$ ⟩]
  ⟨on-curve a b p⟩])
then show ?case
proof
  assume  $p_3 = p$ 
  with ⟨ $p_3 = \text{add a (add a p p) (opp p)$ ⟩
  show ?thesis by simp
next
  assume  $p_3 = \text{opp } p$ 
  with ⟨ $p_3 = \text{add a (add a p p) (opp p)$ ⟩
  have  $\text{add a (add a p p) (opp p) = opp } p$  by simp
  with ab add-closed [OF p p] opp-closed [OF p]
  have  $\text{add a p p} = \text{Infinity}$  by (rule uniq-zero)
  with ⟨ $\text{add a p p} = \text{Point } x_1 \ y_1$ ⟩ show ?thesis by simp
qed
qed

lemma add-opp-double-opp:
  assumes ab: nonsingular a b
  and  $p_1: \text{on-curve } a \ b \ p_1$ 
  and  $p_2: \text{on-curve } a \ b \ p_2$ 
  and  $\text{add a } p_1 \ p_2 = \text{opp } p_1$ 
  shows  $p_2 = \text{add a (opp } p_1) (\text{opp } p_1)$ 
proof (cases  $p_1 = \text{opp } p_1$ )
  case True
  with assms have  $\text{add a } p_2 \ p_1 = p_1$  by (simp add: add-comm)
  with ab  $p_2 \ p_1$  have  $p_2 = \text{Infinity}$  by (rule uniq-zero)
  also from ⟨on-curve a b p⟩ have  $\dots = \text{add a } p_1 (\text{opp } p_1)$ 
  by (simp add: add-opp)

```

```

also from True have ... = add a (opp p1) (opp p1) by simp
finally show ?thesis .
next
case False
from p1 p2 False assms show ?thesis
proof (induct rule: add-case)
  case InfL
  then show ?case by simp
next
  case InfR
  then show ?case by simp
next
  case Opp
  then show ?case by (simp add: add-0-l)
next
  case (Tan p1 x1 y1 p2 x2 y2 l)
  from ⟨p2 = opp p1⟩ ⟨on-curve a b p1⟩
  have p1 = opp p2 by (simp add: opp-opp)
  also note ⟨p2 = add a p1 p1⟩
  finally show ?case using ⟨on-curve a b p1⟩
    by (simp add: opp-add)
next
  case (Gen p1 x1 y1 p2 x2 y2 p3 x3 y3 l)
  from ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
  have y1: y1 ^ 2 = x1 ^ 3 + a * x1 + b
    by (simp add: on-curve-def)
  from ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
  have y2: y2 ^ 2 = x2 ^ 3 + a * x2 + b
    by (simp add: on-curve-def)
  from ⟨p1 = Point x1 y1⟩ ⟨p1 ≠ opp p1⟩
  have y1 ≠ 0
    by (simp add: opp-Point)
  from Gen have x1 = ((y2 - y1) / (x2 - x1)) ^ 2 - x1 - x2
    by (simp add: opp-Point)
  then have 2 * y2 * y1 = a * x2 + 3 * x2 * x1 ^ 2 + a * x1 -
    x1 ^ 3 + 2 * b
    apply (field (prems) y1 y2)
    apply (field y1 y2)
    apply simp
    apply (simp add: ⟨x1 ≠ x2⟩ [symmetric])
  done
  then have (x2 - (((3 * x1 ^ 2 + a) / (2 * (- y1))) ^ 2 -
    2 * x1)) * (x2 - x1) ^ 2 = 0
    apply (drule-tac f=λx. x ^ 2 in arg-cong)
    apply (field (prems) y1 y2)
    apply (field y1 y2)
    apply (simp-all add: ⟨y1 ≠ 0⟩)
  done
with ⟨x1 ≠ x2⟩

```



```

have  $x_2 = ((3 * x_1 \wedge 2 + a) / (2 * (- y_1))) \wedge 2 - 2 * x_1$ 
  by simp
with  $\langle p_2 = \text{Point } x_2 \ y_2 \rangle - \langle \text{on-curve } a \ b \ p_2 \rangle$ 
  add-closed [OF
    opp-closed [OF  $\langle \text{on-curve } a \ b \ p_1 \rangle$ ] opp-closed [OF  $\langle \text{on-curve } a \ b \ p_1 \rangle$ ]]
have  $p_2 = \text{add } a \ (\text{opp } p_1) \ (\text{opp } p_1) \vee p_2 = \text{opp} \ (\text{add } a \ (\text{opp } p_1) \ (\text{opp } p_1))$ 
  apply (rule curve-elt-opp)
  apply (simp add: add-def opp-Point Let-def  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \langle y_1 \neq 0 \rangle$ )
  done
then show ?case
proof
  assume  $p_2 = \text{opp} \ (\text{add } a \ (\text{opp } p_1) \ (\text{opp } p_1))$ 
  with  $\langle \text{on-curve } a \ b \ p_1 \rangle$ 
  have  $p_2 = \text{add } a \ p_1 \ p_1$ 
    by (simp add: opp-add [of a b] opp-opp opp-closed)
  show ?case
  proof (cases add a p_1 p_1 = opp p_1)
    case True
    from  $\langle \text{on-curve } a \ b \ p_1 \rangle$ 
    show ?thesis
      apply (simp add: opp-add [symmetric]  $\langle p_2 = \text{add } a \ p_1 \ p_1 \rangle \text{True}$ )
      apply (simp add:  $\langle p_3 = \text{add } a \ p_1 \ p_2 \rangle$  [simplified  $\langle p_3 = \text{opp } p_1 \rangle$ ])
      apply (simp add:  $\langle p_2 = \text{add } a \ p_1 \ p_1 \rangle \text{True add-opp}$ )
      done
    next
    case False
    from  $\langle \text{on-curve } a \ b \ p_1 \rangle$ 
    have  $\text{add } a \ p_1 \ (\text{opp } p_2) = \text{opp} \ (\text{add } a \ (\text{add } a \ p_1 \ p_1) \ (\text{opp } p_1))$ 
      by (simp add:  $\langle p_2 = \text{add } a \ p_1 \ p_1 \rangle$ 
        opp-add [of a b] add-closed opp-closed opp-opp add-comm [of a b])
    with  $ab \ \langle \text{on-curve } a \ b \ p_1 \rangle \langle p_1 \neq \text{opp } p_1 \rangle \text{False}$ 
    have  $\text{add } a \ p_1 \ (\text{opp } p_2) = \text{opp } p_1$ 
      by (simp add: compat-add-triple)
    with  $\langle p_3 = \text{add } a \ p_1 \ p_2 \rangle \langle p_3 = \text{opp } p_1 \rangle$ 
    have  $\text{add } a \ p_1 \ p_2 = \text{add } a \ p_1 \ (\text{opp } p_2)$  by simp
    with  $\langle \text{on-curve } a \ b \ p_1 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle$ 
    have  $p_2 = \text{opp } p_2$  using  $\langle p_1 \neq \text{opp } p_1 \rangle$ 
      by (rule compat-add-opp)
    with  $\langle \text{on-curve } a \ b \ p_1 \rangle \langle p_2 = \text{add } a \ p_1 \ p_1 \rangle$ 
    show ?thesis by (simp add: opp-add)
  qed
qed
qed
qed

```

```

lemma cancel:
  assumes ab: nonsingular a b
  and  $p_1: \text{on-curve } a \ b \ p_1$ 
  and  $p_2: \text{on-curve } a \ b \ p_2$ 

```

```

and p3: on-curve a b p3
and eq: add a p1 p2 = add a p1 p3
shows p2 = p3
using p1 p2 p1 p2 eq
proof (induct rule: add-casew)
  case InfL
  then show ?case by (simp add: add-0-l)
next
  case (InfR p)
  with p3 have add a p3 p = p by (simp add: add-comm)
  with ab p3 ⟨on-curve a b p⟩
  show ?case by (rule uniq-zero [symmetric])
next
  case (Opp p)
  from ⟨Infinity = add a p p3⟩ [symmetric]
  show ?case by (rule uniq-opp [symmetric])
next
  case (Gen p1 x1 y1 p2 x2 y2 p4 x4 y4 l)
  from ⟨on-curve a b p1⟩ p3 ⟨on-curve a b p1⟩ p3 ⟨p1 = Point x1 y1⟩
    ⟨p4 = add a p1 p2⟩ ⟨p4 = add a p1 p3⟩ ⟨p1 ≠ opp p2⟩
  show ?case
  proof (induct rule: add-casew)
    case InfL
    then show ?case by (simp add: add-0-l)
  next
    case (InfR p)
    with ⟨on-curve a b p2⟩
    have add a p2 p = p by (simp add: add-comm)
    with ab ⟨on-curve a b p2⟩ ⟨on-curve a b p⟩
    show ?case by (rule uniq-zero)
  next
    case (Opp p)
    then have add a p p2 = Infinity by simp
    then show ?case by (rule uniq-opp)
  next
    case (Gen p1 x1' y1' p3 x3 y3 p5 x5 y5 l')
    from ⟨p4 = p5⟩ ⟨p4 = Point x4 y4⟩ ⟨p5 = Point x5 y5⟩
      ⟨p1 = Point x1' y1'⟩ ⟨p1 = Point x1 y1⟩
      ⟨y4 = - y1 - l * (x4 - x1)⟩ ⟨y5 = - y1' - l' * (x5 - x1')⟩
    have 0 = - y1 - l * (x4 - x1) - (- y1 - l' * (x4 - x1))
      by auto
    then have l' = l ∨ x4 = x1 by auto
    then show ?case
  proof
    assume l' = l
    with ⟨p4 = p5⟩ ⟨p4 = Point x4 y4⟩ ⟨p5 = Point x5 y5⟩
      ⟨p1 = Point x1' y1'⟩ ⟨p1 = Point x1 y1⟩
      ⟨x4 = l ^ 2 - x1 - x2⟩ ⟨x5 = l' ^ 2 - x1' - x3⟩
    have 0 = l ^ 2 - x1 - x2 - (l ^ 2 - x1 - x3)

```

```

    by simp
  then have  $x_2 = x_3$  by simp
  with  $\langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle p_3 = \text{Point } x_3 \ y_3 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle \langle \text{on-curve } a \ b \ p_3 \rangle$ 
  have  $p_2 = p_3 \vee p_2 = \text{opp } p_3$  by (rule curve-elt-opp)
  then show ?case
  proof
    assume  $p_2 = \text{opp } p_3$ 
    with  $\langle \text{on-curve } a \ b \ p_3 \rangle$  have  $\text{opp } p_2 = p_3$ 
      by (simp add: opp-opp)
    with  $\langle p_4 = p_5 \rangle \langle p_4 = \text{add } a \ p_1 \ p_2 \rangle \langle p_5 = \text{add } a \ p_1 \ p_3 \rangle$ 
    have  $\text{add } a \ p_1 \ p_2 = \text{add } a \ p_1 \ (\text{opp } p_2)$  by simp
    show ?case
    proof (cases  $p_1 = \text{opp } p_1$ )
      case True
        with  $\langle p_1 \neq \text{opp } p_2 \rangle \langle p_1 \neq \text{opp } p_3 \rangle$ 
        have  $p_1 \neq p_2 \ p_1 \neq p_3$  by auto
        with  $\langle l' = l \rangle \langle x_1 = x_2 \wedge \neg \rightarrow \langle x_1' = x_3 \wedge \neg \rightarrow \rangle$ 
           $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \langle p_1 = \text{Point } x_1' \ y_1' \rangle$ 
           $\langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle p_3 = \text{Point } x_3 \ y_3 \rangle$ 
           $\langle p_2 = \text{opp } p_3 \rangle$ 
        have  $\text{eq: } (y_2 - y_1) / (x_2 - x_1) = (y_3 - y_1) / (x_2 - x_1)$  and  $x_1 \neq x_2$ 
          by (auto simp add: opp-Point)
        from eq have  $y_2 = y_3$ 
          apply (field (prems))
          apply (simp-all add:  $\langle x_1 \neq x_2 \rangle$  [symmetric])
        done
        with  $\langle p_2 = \text{opp } p_3 \rangle \langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle p_3 = \text{Point } x_3 \ y_3 \rangle$ 
        show ?thesis by (simp add: opp-Point)
      case False
        with  $\langle \text{on-curve } a \ b \ p_1 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle$ 
           $\langle \text{add } a \ p_1 \ p_2 = \text{add } a \ p_1 \ (\text{opp } p_2) \rangle$ 
        have  $p_2 = \text{opp } p_2$  by (rule compat-add-opp)
        with  $\langle \text{opp } p_2 = p_3 \rangle$  show ?thesis by simp
    qed
  qed
next
  assume  $x_4 = x_1$ 
  with  $\langle p_4 = \text{Point } x_4 \ y_4 \rangle$  [simplified  $\langle p_4 = \text{add } a \ p_1 \ p_2 \rangle$ ]
     $\langle p_1 = \text{Point } x_1 \ y_1 \rangle$ 
    add-closed [OF  $\langle \text{on-curve } a \ b \ p_1 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle$ ]
     $\langle \text{on-curve } a \ b \ p_1 \rangle$ 
  have  $\text{add } a \ p_1 \ p_2 = p_1 \vee \text{add } a \ p_1 \ p_2 = \text{opp } p_1$  by (rule curve-elt-opp)
  then show ?case
  proof
    assume  $\text{add } a \ p_1 \ p_2 = p_1$ 
    with  $\langle \text{on-curve } a \ b \ p_1 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle$ 
    have  $\text{add } a \ p_2 \ p_1 = p_1$  by (simp add: add-comm)

```

```

with  $ab$   $\langle on\text{-}curve\ a\ b\ p_2 \rangle$   $\langle on\text{-}curve\ a\ b\ p_1 \rangle$ 
have  $p_2 = Infinity$  by (rule uniq-zero)
moreover from  $\langle add\ a\ p_1\ p_2 = p_1 \rangle$ 
   $\langle p_4 = p_5 \rangle$   $\langle p_4 = add\ a\ p_1\ p_2 \rangle$   $\langle p_5 = add\ a\ p_1\ p_3 \rangle$ 
   $\langle on\text{-}curve\ a\ b\ p_1 \rangle$   $\langle on\text{-}curve\ a\ b\ p_3 \rangle$ 
have  $add\ a\ p_3\ p_1 = p_1$  by (simp add: add-comm)
with  $ab$   $\langle on\text{-}curve\ a\ b\ p_3 \rangle$   $\langle on\text{-}curve\ a\ b\ p_1 \rangle$ 
have  $p_3 = Infinity$  by (rule uniq-zero)
ultimately show ?case by simp
next
assume  $add\ a\ p_1\ p_2 = opp\ p_1$ 
with  $ab$   $\langle on\text{-}curve\ a\ b\ p_1 \rangle$   $\langle on\text{-}curve\ a\ b\ p_2 \rangle$ 
have  $p_2 = add\ a\ (opp\ p_1)\ (opp\ p_1)$  by (rule add-opp-double-opp)
moreover from  $\langle add\ a\ p_1\ p_2 = opp\ p_1 \rangle$ 
   $\langle p_4 = p_5 \rangle$   $\langle p_4 = add\ a\ p_1\ p_2 \rangle$   $\langle p_5 = add\ a\ p_1\ p_3 \rangle$ 
have  $add\ a\ p_1\ p_3 = opp\ p_1$  by simp
with  $ab$   $\langle on\text{-}curve\ a\ b\ p_1 \rangle$   $\langle on\text{-}curve\ a\ b\ p_3 \rangle$ 
have  $p_3 = add\ a\ (opp\ p_1)\ (opp\ p_1)$  by (rule add-opp-double-opp)
ultimately show ?case by simp
qed
qed
qed
qed

```

**lemma** *add-minus-id*:

```

assumes  $ab$ : nonsingular\ a\ b
and  $p_1$ : on-curve\ a\ b\ p_1
and  $p_2$ : on-curve\ a\ b\ p_2
shows  $add\ a\ (add\ a\ p_1\ p_2)\ (opp\ p_2) = p_1$ 
proof (cases\ add\ a\ p_1\ p_2 = opp\ p_2)
case True
then have  $add\ a\ (add\ a\ p_1\ p_2)\ (opp\ p_2) = add\ a\ (opp\ p_2)\ (opp\ p_2)$ 
  by simp
also from  $p_1\ p_2\ True$  have  $add\ a\ p_2\ p_1 = opp\ p_2$ 
  by (simp add: add-comm)
with  $ab\ p_2\ p_1$  have  $add\ a\ (opp\ p_2)\ (opp\ p_2) = p_1$ 
  by (rule add-opp-double-opp [symmetric])
finally show ?thesis .
next
case False
from  $p_1\ p_2\ p_1\ p_2\ False$  show ?thesis
proof (induct\ rule:\ add-case)
case InfL
then show ?case by (simp add: add-opp)
next
case InfR
show ?case by (simp add: add-0-r)
next
case Opp

```

```

then show ?case by (simp add: opp-opp add-0-l)
next
case (Tan p1 x1 y1 p2 x2 y2 l)
note ab ⟨on-curve a b p1⟩
moreover from ⟨y1 ≠ 0⟩ ⟨p1 = Point x1 y1⟩
have p1 ≠ opp p1 by (simp add: opp-Point)
moreover from ⟨p2 = add a p1 p1⟩ ⟨p2 ≠ opp p1⟩
have add a p1 p1 ≠ opp p1 by simp
ultimately have add a (add a p1 p1) (opp p1) = p1
  by (rule compat-add-triple)
with ⟨p2 = add a p1 p1⟩ show ?case by simp
next
case (Gen p1 x1 y1 p2 x2 y2 p3 x3 y3 l)
from ⟨p3 = add a p1 p2⟩ ⟨on-curve a b p2⟩
have p3 = add a p1 (opp (opp p2)) by (simp add: opp-opp)
with
  add-closed [OF ⟨on-curve a b p1⟩ ⟨on-curve a b p2⟩,
    folded ⟨p3 = add a p1 p2⟩]
  opp-closed [OF ⟨on-curve a b p2⟩]
  opp-closed [OF ⟨on-curve a b p2⟩]
  opp-opp [of p2]
  Gen
show ?case
proof (induct rule: add-case)
  case InFL
  then show ?case by simp
next
  case InFR
  then show ?case by (simp add: add-0-r)
next
  case (Opp p)
  from ⟨p = add a p1 (opp (opp p))⟩
  have add a p1 p = p by (simp add: opp-opp)
  with ab ⟨on-curve a b p1⟩ ⟨on-curve a b p⟩
  show ?case by (rule uniq-zero [symmetric])
next
  case Tan
  then show ?case by simp
next
case (Gen p4 x4 y4 p5 x5 y5 p6 x6 y6 l')
from ⟨on-curve a b p5⟩ ⟨opp p5 = p2⟩
  ⟨p2 = Point x2 y2⟩ ⟨p5 = Point x5 y5⟩
have y5 = - y2 x5 = x2
  by (auto simp add: opp-Point on-curve-def)
from ⟨p4 = Point x3 y3⟩ ⟨p4 = Point x4 y4⟩
have x4 = x3 y4 = y3 by simp-all
from ⟨x4 ≠ x5⟩ show ?case
  apply (simp add:
    ⟨y5 = - y2⟩ ⟨x5 = x2⟩

```

```

    ⟨x4 = x3⟩ ⟨y4 = y3⟩
    ⟨p6 = Point x6 y6⟩ ⟨p1 = Point x1 y1⟩
    ⟨x6 = l' ^ 2 - x4 - x5⟩ ⟨y6 = - y4 - l' * (x6 - x4)⟩
    ⟨l' = (y5 - y4) / (x5 - x4)⟩
    ⟨x3 = l ^ 2 - x1 - x2⟩ ⟨y3 = - y1 - l * (x3 - x1)⟩
    ⟨l = (y2 - y1) / (x2 - x1)⟩
  apply (rule conjI)
  apply field
  apply (rule conjI)
  apply (rule notI)
  apply (erule notE)
  apply (ring (prems))
  apply (rule sym)
  apply field
  apply (simp-all add: ⟨x1 ≠ x2⟩ [symmetric])
  apply field
  apply (rule conjI)
  apply (rule notI)
  apply (erule notE)
  apply (ring (prems))
  apply (rule sym)
  apply field
  apply (simp-all add: ⟨x1 ≠ x2⟩ [symmetric])
done
qed
qed
qed

lemma add-shift-minus:
  assumes ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and p3: on-curve a b p3
  and eq: add a p1 p2 = p3
  shows p1 = add a p3 (opp p2)
proof -
  note eq
  also from add-minus-id [OF ab p3 opp-closed [OF p2]] p2
  have p3 = add a (add a p3 (opp p2)) p2 by (simp add: opp-opp)
  finally have add a p2 p1 = add a p2 (add a p3 (opp p2))
  using p1 p2 p3
  by (simp add: add-comm [of a b] add-closed opp-closed)
  with ab p2 p1 add-closed [OF p3 opp-closed [OF p2]]
  show ?thesis by (rule cancel)
qed

lemma degen-assoc:
  assumes ab: nonsingular a b
  and p1: on-curve a b p1

```

```

and p2: on-curve a b p2
and p3: on-curve a b p3
and H:
  (p1 = Infinity  $\vee$  p2 = Infinity  $\vee$  p3 = Infinity)  $\vee$ 
  (p1 = opp p2  $\vee$  p2 = opp p3)  $\vee$ 
  (opp p1 = add a p2 p3  $\vee$  opp p3 = add a p1 p2)
shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
using H
proof (elim disjE)
  assume p1 = Infinity
  then show ?thesis by (simp add: add-0-l)
next
  assume p2 = Infinity
  then show ?thesis by (simp add: add-0-l add-0-r)
next
  assume p3 = Infinity
  then show ?thesis by (simp add: add-0-r)
next
  assume p1 = opp p2
  from p2 p3
  have add a (opp p2) (add a p2 p3) = add a (add a p3 p2) (opp p2)
    by (simp add: add-comm [of a b] add-closed opp-closed)
  also from ab p3 p2 have ... = p3 by (rule add-minus-id)
  also have ... = add a Infinity p3 by (simp add: add-0-l)
  also from p2 have ... = add a (add a p2 (opp p2)) p3
    by (simp add: add-opp)
  also from p2 have ... = add a (add a (opp p2) p2) p3
    by (simp add: add-comm [of a b] opp-closed)
  finally show ?thesis using ⟨p1 = opp p2⟩ by simp
next
  assume p2 = opp p3
  from p3
  have add a p1 (add a (opp p3) p3) = add a p1 (add a p3 (opp p3))
    by (simp add: add-comm [of a b] opp-closed)
  also from ab p1 p3
  have ... = add a (add a p1 (opp p3)) (opp (opp p3))
    by (simp add: add-opp add-minus-id add-0-r opp-closed)
  finally show ?thesis using p3 ⟨p2 = opp p3⟩
    by (simp add: opp-opp)
next
  assume eq: opp p1 = add a p2 p3
  from eq [symmetric] p1
  have add a p1 (add a p2 p3) = Infinity by (simp add: add-opp)
  also from p3 have ... = add a p3 (opp p3) by (simp add: add-opp)
  also from p3 have ... = add a (opp p3) p3
    by (simp add: add-comm [of a b] opp-closed)
  also from ab p2 p3
  have ... = add a (add a (add a (opp p3) (opp p2)) (opp (opp p2))) p3
    by (simp add: add-minus-id opp-closed)

```

```

also from  $p_2 p_3$ 
have  $\dots = \text{add } a (\text{add } a (\text{add } a (\text{opp } p_2) (\text{opp } p_3)) p_2) p_3$ 
  by (simp add: add-comm [of a b] opp-opp opp-closed)
finally show ?thesis
  using opp-add [OF p2 p3] eq [symmetric] p1
  by (simp add: opp-opp)
next
assume eq: opp p3 = add a p1 p2
from opp-add [OF p1 p2] eq [symmetric] p3
have  $\text{add } a p_1 (\text{add } a p_2 p_3) = \text{add } a p_1 (\text{add } a p_2 (\text{add } a (\text{opp } p_1) (\text{opp } p_2)))$ 
  by (simp add: opp-opp)
also from  $p_1 p_2$ 
have  $\dots = \text{add } a p_1 (\text{add } a (\text{add } a (\text{opp } p_1) (\text{opp } p_2)) (\text{opp } (\text{opp } p_2)))$ 
  by (simp add: add-comm [of a b] opp-opp add-closed opp-closed)
also from ab p1 p2 have  $\dots = \text{Infinity}$ 
  by (simp add: add-minus-id add-opp opp-closed)
also from  $p_3$  have  $\dots = \text{add } a p_3 (\text{opp } p_3)$  by (simp add: add-opp)
also from  $p_3$  have  $\dots = \text{add } a (\text{opp } p_3) p_3$ 
  by (simp add: add-comm [of a b] opp-closed)
finally show ?thesis using eq [symmetric] by simp
qed

```

**lemma** *spec4-assoc*:

```

assumes ab: nonsingular a b
and  $p_1: \text{on-curve } a b p_1$ 
and  $p_2: \text{on-curve } a b p_2$ 
shows  $\text{add } a p_1 (\text{add } a p_2 p_2) = \text{add } a (\text{add } a p_1 p_2) p_2$ 
proof (cases p1 = Infinity)
  case True
    from ab p1 p2 p2
    show ?thesis by (rule degen-assoc) (simp add: True)
  next
    case False
    show ?thesis
    proof (cases p2 = Infinity)
      case True
        from ab p1 p2 p2
        show ?thesis by (rule degen-assoc) (simp add: True)
      next
        case False
        show ?thesis
        proof (cases p2 = opp p2)
          case True
            from ab p1 p2 p2
            show ?thesis by (rule degen-assoc) (simp add: True [symmetric])
          next
            case False
            show ?thesis
            proof (cases p1 = opp p2)

```



```

case True
from ab p1 p2 p2
show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p1 = add a p2 p2)
  case True
  from ab p1 p2 p2
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p2 = add a p1 p2)
  case True
  from ab p1 p2 p2
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p1 = add a p2 p2)
  case True
  from p1 p2 ⟨p1 ≠ opp p2⟩ ⟨p2 ≠ opp p2⟩
    ⟨opp p1 ≠ add a p2 p2⟩ ⟨opp p2 ≠ add a p1 p2⟩
    ⟨p1 ≠ Infinity⟩ ⟨p2 ≠ Infinity⟩
  show ?thesis
  apply (simp add: True)
  apply (rule spec3-assoc)
  apply (simp-all add: is-generic-def is-tangent-def)
  apply (rule notI)
  apply (drule uniq-zero [OF ab p2 p2])
  apply simp
  apply (intro conjI notI)
  apply (erule notE)
  apply (rule uniq-opp [of a])
  apply (simp add: add-comm [of a b] add-closed)
  apply (erule notE)
  apply (drule uniq-zero [OF ab add-closed [OF p2 p2] p2])
  apply simp
  done
next
case False
show ?thesis
proof (cases p2 = add a p1 p2)
  case True
  from ab p1 p2 True [symmetric]
  have p1 = Infinity by (rule uniq-zero)
  then show ?thesis by (simp add: add-0-l)
next

```

```

case False
show ?thesis
proof (cases p1 = p2)
  case True
  with p2 show ?thesis
  by (simp add: add-comm [of a b] add-closed)
next
case False
with p1 p2 ⟨p1 ≠ Infinity⟩ ⟨p2 ≠ Infinity⟩
  ⟨p1 ≠ opp p2⟩ ⟨p2 ≠ opp p2⟩
  ⟨p1 ≠ add a p2 p2⟩ ⟨p2 ≠ add a p1 p2⟩ ⟨opp p2 ≠ add a p1 p2⟩
show ?thesis
  apply (rule-tac spec2-assoc)
  apply (simp-all add: is-generic-def is-tangent-def)
  apply (rule notI)
  apply (erule notE [of p1 = opp p2])
  apply (rule uniq-opp [of a])
  apply (simp add: add-comm)
  apply (intro conjI notI)
  apply (erule notE [of p2 = opp p2])
  apply (rule uniq-opp)
  apply assumption+
  apply (rule notE [OF ⟨opp p1 ≠ add a p2 p2⟩])
  apply (simp add: opp-opp)
done
qed
qed
qed
qed
qed
qed
qed
qed
qed

```

```

lemma add-assoc:
  assumes ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and p3: on-curve a b p3
  shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
proof (cases p1 = Infinity)
  case True
  from ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p2 = Infinity)

```

```

case True
from ab p1 p2 p3
show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p3 = Infinity)
  case True
  from ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p1 = p2)
  case True
  from p2 p3
  have add a p2 (add a p2 p3) = add a (add a p3 p2) p2
    by (simp add: add-comm [of a b] add-closed)
  also from ab p3 p2 have ... = add a p3 (add a p2 p2)
    by (simp add: spec4-assoc)
  also from p2 p3
  have ... = add a (add a p2 p2) p3
    by (simp add: add-comm [of a b] add-closed)
  finally show ?thesis using True by simp
next
case False
show ?thesis
proof (cases p1 = opp p2)
  case True
  from ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p2 = p3)
  case True
  with ab p1 p3 show ?thesis
    by (simp add: spec4-assoc)
next
case False
show ?thesis
proof (cases p2 = opp p3)
  case True
  from ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p1 = add a p2 p3)

```

```

    case True
    from ab p1 p2 p3
    show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p3 = add a p1 p2)
  case True
  from ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p1 = add a p2 p3)
  case True
  with ab p2 p3 show ?thesis
  apply simp
  apply (rule cancel [OF ab opp-closed [OF p3]])
  apply (simp-all add: add-closed)
  apply (simp add: spec4-assoc add-closed opp-closed)
  apply (simp add: add-comm [of a b opp p3]
    add-closed opp-closed add-minus-id)
  apply (simp add: add-comm [of a b] add-closed)
  done
next
case False
show ?thesis
proof (cases p3 = add a p1 p2)
  case True
  with ab p1 p2 show ?thesis
  apply simp
  apply (rule cancel [OF ab opp-closed [OF p1]])
  apply (simp-all add: add-closed)
  apply (simp add: spec4-assoc add-closed opp-closed)
  apply (simp add: add-comm [of a b opp p1] add-comm [of a b
    p1]
    add-closed opp-closed add-minus-id)
  done
next
case False
with p1 p2 p3
  ⟨p1 ≠ Infinity⟩ ⟨p2 ≠ Infinity⟩ ⟨p3 ≠ Infinity⟩
  ⟨p1 ≠ p2⟩ ⟨p1 ≠ opp p2⟩ ⟨p2 ≠ p3⟩ ⟨p2 ≠ opp p3⟩
  ⟨opp p3 ≠ add a p1 p2⟩ ⟨p1 ≠ add a p2 p3⟩
show ?thesis
  apply (rule-tac spec1-assoc [of a b])
  apply (simp-all add: is-generic-def)
  apply (rule notI)
  apply (erule notE [of p1 = opp p2])

```

```

    apply (rule uniq-opp [of a])
    apply (simp add: add-comm)
    apply (intro conjI notI)
    apply (erule notE [of p2 = opp p3])
    apply (rule uniq-opp [of a])
    apply (simp add: add-comm)
    apply (rule notE [OF ⟨opp p1 ≠ add a p2 p3⟩])
    apply (simp add: opp-opp)
  done
qed
qed
qed
qed
qed
qed
qed
qed
qed
qed
qed

```

**lemma** *add-comm'*:

```

  nonsingular a b  $\implies$ 
  on-curve a b p1  $\implies$  on-curve a b p2  $\implies$  on-curve a b p3  $\implies$ 
  add a p2 (add a p1 p3) = add a p1 (add a p2 p3)
  by (simp add: add-assoc add-comm)

```

**primrec** (in *ell-field*) *point-mult* :: 'a  $\Rightarrow$  nat  $\Rightarrow$  'a point  $\Rightarrow$  'a point  
**where**

```

  point-mult a 0 p = Infinity
  | point-mult a (Suc n) p = add a p (point-mult a n p)

```

**lemma** *point-mult-closed*: on-curve a b p  $\implies$  on-curve a b (point-mult a n p)  
**by** (induct n) (simp-all add: add-closed)

**lemma** *point-mult-add*:

```

  on-curve a b p  $\implies$  nonsingular a b  $\implies$ 
  point-mult a (m + n) p = add a (point-mult a m p) (point-mult a n p)
  by (induct m) (simp-all add: add-assoc point-mult-closed add-0-l)

```

**lemma** *point-mult-mult*:

```

  on-curve a b p  $\implies$  nonsingular a b  $\implies$ 
  point-mult a (m * n) p = point-mult a n (point-mult a m p)
  by (induct n) (simp-all add: point-mult-add)

```

**lemma** *point-mult2-eq-double*:

```

  point-mult a 2 p = add a p p
  by (simp add: numeral-2-eq-2 add-0-r)

```

## 2.2 Projective Coordinates

**type-synonym** 'a ppoint = 'a × 'a × 'a

**context** ell-field begin

**definition** pdouble :: 'a ⇒ 'a ppoint ⇒ 'a ppoint **where**

pdouble a p =  
 (let (x, y, z) = p  
 in  
 if z = 0 then p  
 else  
 let  
 l = 2 \* y \* z;  
 m = 3 \* x ^ 2 + a \* z ^ 2  
 in  
 (l \* (m ^ 2 - 4 \* x \* y \* l),  
 m \* (6 \* x \* y \* l - m ^ 2) -  
 2 \* y ^ 2 \* l ^ 2,  
 l ^ 3))

**definition** padd :: 'a ⇒ 'a ppoint ⇒ 'a ppoint ⇒ 'a ppoint **where**

padd a p1 p2 =  
 (let  
 (x1, y1, z1) = p1;  
 (x2, y2, z2) = p2  
 in  
 if z1 = 0 then p2  
 else if z2 = 0 then p1  
 else  
 let  
 d1 = x2 \* z1;  
 d2 = x1 \* z2;  
 l = d1 - d2;  
 m = y2 \* z1 - y1 \* z2  
 in  
 if l = 0 then  
 if m = 0 then pdouble a p1  
 else (0, 0, 0)  
 else  
 let h = m ^ 2 \* z1 \* z2 - (d1 + d2) \* l ^ 2  
 in  
 (l \* h,  
 (d2 \* l ^ 2 - h) \* m - l ^ 3 \* y1 \* z2,  
 l ^ 3 \* z1 \* z2))

**definition** make-affine :: 'a ppoint ⇒ 'a point **where**

make-affine p =  
 (let (x, y, z) = p  
 in if z = 0 then Infinity else Point (x / z) (y / z))

```

definition on-curvep :: 'a ⇒ 'a ⇒ 'a ppoint ⇒ bool where
  on-curvep a b = (λ(x, y, z). z ≠ 0 ⟶
    y ^ 2 * z = x ^ 3 + a * x * z ^ 2 + b * z ^ 3)

end

lemma on-curvep-infinity [simp]: on-curvep a b (x, y, 0)
  by (simp add: on-curvep-def)

lemma make-affine-infinity [simp]: make-affine (x, y, 0) = Infinity
  by (simp add: make-affine-def)

lemma on-curvep-iff-on-curve:
  on-curvep a b p = on-curve a b (make-affine p)
proof (induct p rule: prod-induct3)
  case (fields x y z)
  show on-curvep a b (x, y, z) = on-curve a b (make-affine (x, y, z))
  proof
    assume H: on-curvep a b (x, y, z)
    then have yz: z ≠ 0 ⟹ y ^ 2 * z = x ^ 3 + a * x * z ^ 2 + b * z ^ 3
      by (simp-all add: on-curvep-def)
    show on-curve a b (make-affine (x, y, z))
    proof (cases z = 0)
      case True
      then show ?thesis by (simp add: on-curve-def make-affine-def)
    next
      case False
      then show ?thesis
        apply (simp add: on-curve-def make-affine-def)
        apply (field yz [OF False])
        apply assumption
        done
    qed
  next
    assume H: on-curve a b (make-affine (x, y, z))
    show on-curvep a b (x, y, z)
    proof (cases z = 0)
      case True
      then show ?thesis
        by (simp add: on-curvep-def)
    next
      case False
      from H show ?thesis
        apply (simp add: on-curve-def on-curvep-def make-affine-def False)
        apply (field (prems))
        apply field
        apply (simp-all add: False)
        done
  qed

```

qed  
 qed  
 qed

**lemma** *pdouble-infinity* [*simp*]:  $pdouble\ a\ (x, y, 0) = (x, y, 0)$   
 by (*simp add: pdouble-def*)

**lemma** *padd-infinity-l* [*simp*]:  $padd\ a\ (x, y, 0)\ p = p$   
 by (*simp add: padd-def*)

**lemma** *pdouble-correct*:  
 $make\ affine\ (pdouble\ a\ p) = add\ a\ (make\ affine\ p)\ (make\ affine\ p)$   
**proof** (*induct p rule: prod-induct3*)  
 case (*fields x y z*)  
 then show ?*case*  
 apply (*auto simp add: add-def pdouble-def make-affine-def eq-opp-is-zero Let-def*)  
 apply *field*  
 apply *simp*  
 apply *field*  
 apply *simp*  
 done  
 qed

**lemma** *padd-correct*:  
 assumes  $p_1: on\ curvep\ a\ b\ p_1$  and  $p_2: on\ curvep\ a\ b\ p_2$   
 shows  $make\ affine\ (padd\ a\ p_1\ p_2) = add\ a\ (make\ affine\ p_1)\ (make\ affine\ p_2)$   
 using  $p_1$   
**proof** (*induct p<sub>1</sub> rule: prod-induct3*)  
 case (*fields x<sub>1</sub> y<sub>1</sub> z<sub>1</sub>*)  
 note  $p_1' = fields$   
 from  $p_2$  show ?*case*  
**proof** (*induct p<sub>2</sub> rule: prod-induct3*)  
 case (*fields x<sub>2</sub> y<sub>2</sub> z<sub>2</sub>*)  
 then have  
 $yz_2: z_2 \neq 0 \implies y_2^2 * z_2 * z_1^3 =$   
 $(x_2^3 + a * x_2 * z_2^2 + b * z_2^3) * z_1^3$   
 by (*simp-all add: on-curvep-def*)  
 from  $p_1'$  have  
 $yz_1: z_1 \neq 0 \implies y_1^2 * z_1 * z_2^3 =$   
 $(x_1^3 + a * x_1 * z_1^2 + b * z_1^3) * z_2^3$   
 by (*simp-all add: on-curvep-def*)  
 show ?*case*  
**proof** (*cases z<sub>1</sub> = 0*)  
 case *True*  
 then show ?*thesis*  
 by (*simp add: add-def padd-def make-affine-def*)  
 next  
 case *False*  
 show ?*thesis*



```

proof (cases z2 = 0)
  case True
  then show ?thesis
    by (simp add: add-def padd-def make-affine-def)
next
  case False
  show ?thesis
  proof (cases x2 * z1 - x1 * z2 = 0)
    case True
    note x = this
    then have x': x2 * z1 = x1 * z2 by simp
    show ?thesis
  proof (cases y2 * z1 - y1 * z2 = 0)
    case True
    then have y: y2 * z1 = y1 * z2 by simp
    from ⟨z1 ≠ 0⟩ ⟨z2 ≠ 0⟩ x
    have make-affine (x2, y2, z2) = make-affine (x1, y1, z1)
    apply (simp add: make-affine-def)
    apply (rule conjI)
    apply (field x')
    apply simp
    apply (field y)
    apply simp
    done
  with True x ⟨z1 ≠ 0⟩ ⟨z2 ≠ 0⟩ p1' fields show ?thesis
    by (simp add: padd-def pdouble-correct)
  next
  case False
  have y2 ^ 2 * z1 ^ 3 * z2 = y1 ^ 2 * z1 * z2 ^ 3
    by (ring yz1 [OF ⟨z1 ≠ 0⟩] yz2 [OF ⟨z2 ≠ 0⟩] x')
  then have y2 ^ 2 * z1 ^ 3 * z2 / z1 / z2 =
    y1 ^ 2 * z1 * z2 ^ 3 / z1 / z2
    by simp
  then have (y2 * z1) * (y2 * z1) = (y1 * z2) * (y1 * z2)
    apply (field (prems))
    apply (field)
    apply (rule TrueI)
    apply (simp add: ⟨z1 ≠ 0⟩ ⟨z2 ≠ 0⟩)
    done
  with False
  have y2z1: y2 * z1 = - (y1 * z2)
    by (simp add: square-eq-iff)
  from x False ⟨z1 ≠ 0⟩ ⟨z2 ≠ 0⟩ show ?thesis
    apply (simp add: padd-def add-def make-affine-def Let-def)
    apply (rule conjI)
    apply (rule impI)
    apply (field x')
    apply simp
    apply (field y2z1)

```

```

      apply simp
    done
  qed
next
case False
then have  $x_1 / z_1 \neq x_2 / z_2$ 
  apply (rule-tac notI)
  apply (erule notE)
  apply (drule sym)
  apply (field (prems))
  apply ring
  apply (simp add:  $\langle z_1 \neq 0 \rangle \langle z_2 \neq 0 \rangle$ )
  done
with False  $\langle z_1 \neq 0 \rangle \langle z_2 \neq 0 \rangle$ 
show ?thesis
  apply (auto simp add: padd-def add-def make-affine-def Let-def)
  apply field
  apply simp
  apply field
  apply simp
  done
qed
qed
qed
qed
qed

```

**lemma** *pdouble-closed*:

*on-curvep a b p*  $\implies$  *on-curvep a b (pdouble a p)*  
**by** (*simp add: on-curvep-iff-on-curve pdouble-correct add-closed*)

**lemma** *padd-closed*:

*on-curvep a b p<sub>1</sub>*  $\implies$  *on-curvep a b p<sub>2</sub>*  $\implies$  *on-curvep a b (padd a p<sub>1</sub> p<sub>2</sub>)*  
**by** (*simp add: on-curvep-iff-on-curve padd-correct add-closed*)

**primrec** (*in ell-field*) *ppoint-mult* :: 'a  $\Rightarrow$  nat  $\Rightarrow$  'a *ppoint*  $\Rightarrow$  'a *ppoint*  
**where**

*ppoint-mult a 0 p* = (0, 0, 0)  
| *ppoint-mult a (Suc n) p* = *padd a p (ppoint-mult a n p)*

**lemma** *ppoint-mult-closed* [*simp*]:

*on-curvep a b p*  $\implies$  *on-curvep a b (ppoint-mult a n p)*  
**by** (*induct n (simp-all add: padd-closed)*)

**lemma** *ppoint-mult-correct*: *on-curvep a b p*  $\implies$

*make-affine (ppoint-mult a n p) = point-mult a n (make-affine p)*  
**by** (*induct n (simp-all add: padd-correct)*)

**context** *ell-field* **begin**

**definition** *proj-eq* :: 'a ppoint  $\Rightarrow$  'a ppoint  $\Rightarrow$  bool **where**  
*proj-eq* = ( $\lambda(x_1, y_1, z_1) (x_2, y_2, z_2).$   
 $(z_1 = 0) = (z_2 = 0) \wedge x_1 * z_2 = x_2 * z_1 \wedge y_1 * z_2 = y_2 * z_1$ )

**end**

**lemma** *proj-eq-refl*: *proj-eq* p p  
**by** (*auto simp add: proj-eq-def*)

**lemma** *proj-eq-sym*: *proj-eq* p p'  $\Longrightarrow$  *proj-eq* p' p  
**by** (*auto simp add: proj-eq-def*)

**lemma** *proj-eq-trans*:  
*in-carrierp* p  $\Longrightarrow$  *in-carrierp* p'  $\Longrightarrow$  *in-carrierp* p''  $\Longrightarrow$   
*proj-eq* p p'  $\Longrightarrow$  *proj-eq* p' p''  $\Longrightarrow$  *proj-eq* p p''

**proof** (*induct p rule: prod-induct3*)

**case** (*fields* x y z)

**then show** ?*case*

**proof** (*induct p' rule: prod-induct3*)

**case** (*fields* x' y' z')

**then show** ?*case*

**proof** (*induct p'' rule: prod-induct3*)

**case** (*fields* x'' y'' z'')

**then have**

z:  $(z = 0) = (z' = 0) (z' = 0) = (z'' = 0)$  **and**

$x * z' * z'' = x' * z * z''$

$y * z' * z'' = y' * z * z''$

**and** xy:

$x' * z'' = x'' * z'$

$y' * z'' = y'' * z'$

**by** (*simp-all add: proj-eq-def*)

**from**  $\langle x * z' * z'' = x' * z * z'' \rangle$

**have**  $(x * z'') * z' = (x'' * z) * z'$

**by** (*ring (prems) xy*) (*ring xy*)

**moreover from**  $\langle y * z' * z'' = y' * z * z'' \rangle$

**have**  $(y * z'') * z' = (y'' * z) * z'$

**by** (*ring (prems) xy*) (*ring xy*)

**ultimately show** ?*case* **using** z

**by** (*auto simp add: proj-eq-def*)

**qed**

**qed**

**qed**

**lemma** *make-affine-proj-eq-iff*:

*proj-eq* p p' = (*make-affine* p = *make-affine* p')

**proof** (*induct p rule: prod-induct3*)

**case** (*fields* x y z)

**then show** ?*case*

```

proof (induct p' rule: prod-induct3)
  case (fields x' y' z')
  show ?case
proof
  assume proj-eq (x, y, z) (x', y', z')
  then have (z = 0) = (z' = 0)
    and xy: x * z' = x' * z y * z' = y' * z
    by (simp-all add: proj-eq-def)
  then show make-affine (x, y, z) = make-affine (x', y', z')
    apply (auto simp add: make-affine-def)
    apply (field xy)
    apply simp
    apply (field xy)
    apply simp
    done
next
  assume H: make-affine (x, y, z) = make-affine (x', y', z')
  show proj-eq (x, y, z) (x', y', z')
  proof (cases z = 0)
    case True
    with H have z' = 0 by (simp add: make-affine-def split: if-split-asm)
    with True show ?thesis by (simp add: proj-eq-def)
  next
    case False
    with H have z' ≠ 0 x / z = x' / z' y / z = y' / z'
      by (simp-all add: make-affine-def split: if-split-asm)
    from ⟨x / z = x' / z'⟩
    have x * z' = x' * z
      apply (field (prems))
      apply field
      apply (simp-all add: ⟨z ≠ 0⟩ ⟨z' ≠ 0⟩)
      done
    moreover from ⟨y / z = y' / z'⟩
    have y * z' = y' * z
      apply (field (prems))
      apply field
      apply (simp-all add: ⟨z ≠ 0⟩ ⟨z' ≠ 0⟩)
      done
    ultimately show ?thesis
      by (simp add: proj-eq-def ⟨z ≠ 0⟩ ⟨z' ≠ 0⟩)
  qed
qed
qed
qed

```

```

lemma pdouble-proj-eq-cong:
  proj-eq p p'  $\implies$  proj-eq (pdouble a p) (pdouble a p')
  by (simp add: make-affine-proj-eq-iff pdouble-correct)

```

```

lemma padd-proj-eq-cong:
  on-curvep a b p1  $\implies$  on-curvep a b p1'  $\implies$  on-curvep a b p2  $\implies$  on-curvep a b
  p2'  $\implies$ 
  proj-eq p1 p1'  $\implies$  proj-eq p2 p2'  $\implies$  proj-eq (padd a p1 p2) (padd a p1' p2')
  by (simp add: make-affine-proj-eq-iff padd-correct)

end

```

### 3 Formalization using Locales

```

theory Elliptic-Locale
imports HOL-Decision-Procs.Reflective-Field
begin

```

#### 3.1 Affine Coordinates

```

datatype 'a point = Infinity | Point 'a 'a

```

```

locale ell-field = field +
  assumes two-not-zero: «2»  $\neq$  0
begin

```

```

declare two-not-zero [simplified, simp add]

```

```

lemma neg-equal-zero:
  assumes x: x  $\in$  carrier R
  shows ( $\ominus$  x = x) = (x = 0)
proof
  assume  $\ominus$  x = x
  with x have «2»  $\otimes$  x = x  $\oplus$   $\ominus$  x
  by (simp add: of-int-2 l-distr)
  with x show x = 0 by (simp add: r-neg integral-iff)
qed simp

```

```

lemmas equal-neg-zero = trans [OF eq-commute neg-equal-zero]

```

```

definition nonsingular :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool where
  nonsingular a b = («4»  $\otimes$  a [ $\ulcorner$ ] (3::nat)  $\oplus$  «27»  $\otimes$  b [ $\ulcorner$ ] (2::nat)  $\neq$  0)

```

```

definition on-curve :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a point  $\Rightarrow$  bool where
  on-curve a b p = (case p of
    Infinity  $\Rightarrow$  True
  | Point x y  $\Rightarrow$  x  $\in$  carrier R  $\wedge$  y  $\in$  carrier R  $\wedge$ 
    y [ $\ulcorner$ ] (2::nat) = x [ $\ulcorner$ ] (3::nat)  $\oplus$  a  $\otimes$  x  $\oplus$  b)

```

```

definition add :: 'a  $\Rightarrow$  'a point  $\Rightarrow$  'a point  $\Rightarrow$  'a point where
  add a p1 p2 = (case p1 of
    Infinity  $\Rightarrow$  p2
  | Point x1 y1  $\Rightarrow$  (case p2 of

```

```

Infinity ⇒ p1
| Point x2 y2 ⇒
  if x1 = x2 then
    if y1 = ⊖ y2 then Infinity
    else
      let
        l = («3» ⊗ x1 [↑] (2::nat) ⊕ a) ⊗ («2» ⊗ y1);
        x3 = l [↑] (2::nat) ⊖ «2» ⊗ x1
      in
        Point x3 (⊖ y1 ⊖ l ⊗ (x3 ⊖ x1))
  else
    let
      l = (y2 ⊖ y1) ⊗ (x2 ⊖ x1);
      x3 = l [↑] (2::nat) ⊖ x1 ⊖ x2
    in
      Point x3 (⊖ y1 ⊖ l ⊗ (x3 ⊖ x1)))

```

**definition** *opp* :: 'a point ⇒ 'a point **where**

```

opp p = (case p of
  Infinity ⇒ Infinity
  | Point x y ⇒ Point x (⊖ y))

```

**lemma** *on-curve-infinity* [*simp*]: *on-curve* *a* *b* *Infinity*  
**by** (*simp* *add*: *on-curve-def*)

**lemma** *opp-Infinity* [*simp*]: *opp* *Infinity* = *Infinity*  
**by** (*simp* *add*: *opp-def*)

**lemma** *opp-Point*: *opp* (*Point* *x* *y*) = *Point* *x* (⊖ *y*)  
**by** (*simp* *add*: *opp-def*)

**lemma** *opp-opp*: *on-curve* *a* *b* *p* ⇒ *opp* (*opp* *p*) = *p*  
**by** (*auto* *simp* *add*: *opp-def* *on-curve-def* *split*: *point.split*)

**lemma** *opp-closed*:  
*on-curve* *a* *b* *p* ⇒ *on-curve* *a* *b* (*opp* *p*)  
**by** (*auto* *simp* *add*: *on-curve-def* *opp-def* *power2-eq-square*  
*l-minus* *r-minus* *split*: *point.split*)

**lemma** *curve-elt-opp*:  
**assumes** *p*<sub>1</sub> = *Point* *x*<sub>1</sub> *y*<sub>1</sub>  
**and** *p*<sub>2</sub> = *Point* *x*<sub>2</sub> *y*<sub>2</sub>  
**and** *on-curve* *a* *b* *p*<sub>1</sub>  
**and** *on-curve* *a* *b* *p*<sub>2</sub>  
**and** *x*<sub>1</sub> = *x*<sub>2</sub>  
**shows** *p*<sub>1</sub> = *p*<sub>2</sub> ∨ *p*<sub>1</sub> = *opp* *p*<sub>2</sub>

**proof** –

```

from ⟨p1 = Point x1 y1⟩ ⟨on-curve a b p1⟩
have y1 ∈ carrier R y1 [↑] (2::nat) = x1 [↑] (3::nat) ⊕ a ⊗ x1 ⊕ b

```

by (*simp-all add: on-curve-def*)  
**moreover from**  $\langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle \text{on-curve } a \ b \ p_2 \rangle \langle x_1 = x_2 \rangle$   
**have**  $y_2 \in \text{carrier } R \ x_1 \ [\wedge] \ (3::\text{nat}) \oplus a \otimes x_1 \oplus b = y_2 \ [\wedge] \ (2::\text{nat})$   
 by (*simp-all add: on-curve-def*)  
**ultimately have**  $y_1 = y_2 \vee y_1 = \ominus y_2$   
 by (*simp add: square-eq-iff power2-eq-square*)  
**with**  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \langle p_2 = \text{Point } x_2 \ y_2 \rangle \langle x_1 = x_2 \rangle$  **show** *?thesis*  
 by (*auto simp add: opp-def*)  
**qed**

**lemma** *add-closed*:

**assumes**  $a \in \text{carrier } R$  **and**  $b \in \text{carrier } R$   
**and** *on-curve*  $a \ b \ p_1$  **and** *on-curve*  $a \ b \ p_2$   
**shows** *on-curve*  $a \ b \ (\text{add } a \ p_1 \ p_2)$   
**proof** (*cases p1*)  
**case** (*Point*  $x_1 \ y_1$ )  
**note** *Point'* = *this*  
**show** *?thesis*  
**proof** (*cases p2*)  
**case** (*Point*  $x_2 \ y_2$ )  
**show** *?thesis*  
**proof** (*cases x1 = x2*)  
**case** *True*  
**note** *True'* = *this*  
**show** *?thesis*  
**proof** (*cases y1 =  $\ominus$  y2*)  
**case** *True*  
**with** *True' Point Point'*  
**show** *?thesis*  
 by (*simp add: on-curve-def add-def*)  
**next**  
**case** *False*  
**from**  $\langle \text{on-curve } a \ b \ p_1 \rangle \ \text{Point}' \ \text{True}'$   
**have**  $x_2 \in \text{carrier } R \ y_1 \in \text{carrier } R$  **and**  
 $\text{on-curve1: } y_1 \ [\wedge] \ (2::\text{nat}) = x_2 \ [\wedge] \ (3::\text{nat}) \oplus a \otimes x_2 \oplus b$   
 by (*simp-all add: on-curve-def*)  
**from** *False True' Point Point' assms*  
**have**  $y_1 \neq \mathbf{0}$   
**apply** (*auto simp add: on-curve-def nat-pow-zero*)  
**apply** (*drule sym [of 0]*)  
**apply** *simp*  
**done**  
**with** *False True' Point Point' assms*  
**show** *?thesis*  
**apply** (*simp add: on-curve-def add-def Let-def integral-iff*)  
**apply** (*field on-curve1*)  
**apply** (*simp add: integral-iff*)  
**done**  
**qed**

```

next
  case False
  from ⟨on-curve a b p1⟩ Point'
  have  $x_1 \in \text{carrier } R$   $y_1 \in \text{carrier } R$ 
    and on-curve1:  $y_1 [\ulcorner] (2::\text{nat}) = x_1 [\ulcorner] (3::\text{nat}) \oplus a \otimes x_1 \oplus b$ 
    by (simp-all add: on-curve-def)
  from ⟨on-curve a b p2⟩ Point
  have  $x_2 \in \text{carrier } R$   $y_2 \in \text{carrier } R$ 
    and on-curve2:  $y_2 [\ulcorner] (2::\text{nat}) = x_2 [\ulcorner] (3::\text{nat}) \oplus a \otimes x_2 \oplus b$ 
    by (simp-all add: on-curve-def)
  from assms not-sym [OF False] show ?thesis
    apply (simp add: on-curve-def add-def Let-def False Point Point' eq-diff0)
    apply (field on-curve1 on-curve2)
    apply (simp add: eq-diff0)
  done
qed
next
  case Infinity
  with Point ⟨on-curve a b p1⟩ show ?thesis
    by (simp add: add-def)
qed
next
  case Infinity
  with ⟨on-curve a b p2⟩ show ?thesis
    by (simp add: add-def)
qed

```

**lemma** *add-case* [*consumes 4, case-names InfL InfR Opp Tan Gen*]:

```

assumes  $a \in \text{carrier } R$ 
and  $b \in \text{carrier } R$ 
and p: on-curve a b p
and q: on-curve a b q
and R1:  $\bigwedge p. P \text{ Infinity } p p$ 
and R2:  $\bigwedge p. P p \text{ Infinity } p$ 
and R3:  $\bigwedge p. \text{on-curve } a b p \implies P p (\text{opp } p) \text{ Infinity}$ 
and R4:  $\bigwedge p_1 x_1 y_1 p_2 x_2 y_2 l.$ 
   $p_1 = \text{Point } x_1 y_1 \implies p_2 = \text{Point } x_2 y_2 \implies$ 
   $p_2 = \text{add } a p_1 p_1 \implies y_1 \neq \mathbf{0} \implies$ 
   $l = (\llbracket 3 \rrbracket \otimes x_1 [\ulcorner] (2::\text{nat}) \oplus a) \otimes (\llbracket 2 \rrbracket \otimes y_1) \implies$ 
   $x_2 = l [\ulcorner] (2::\text{nat}) \ominus \llbracket 2 \rrbracket \otimes x_1 \implies$ 
   $y_2 = \ominus y_1 \ominus l \otimes (x_2 \ominus x_1) \implies$ 
   $P p_1 p_1 p_2$ 
and R5:  $\bigwedge p_1 x_1 y_1 p_2 x_2 y_2 p_3 x_3 y_3 l.$ 
   $p_1 = \text{Point } x_1 y_1 \implies p_2 = \text{Point } x_2 y_2 \implies p_3 = \text{Point } x_3 y_3 \implies$ 
   $p_3 = \text{add } a p_1 p_2 \implies x_1 \neq x_2 \implies$ 
   $l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \implies$ 
   $x_3 = l [\ulcorner] (2::\text{nat}) \ominus x_1 \ominus x_2 \implies$ 
   $y_3 = \ominus y_1 \ominus l \otimes (x_3 \ominus x_1) \implies$ 
   $P p_1 p_2 p_3$ 

```



```

shows  $P p q$  (add a p q)
proof (cases p)
  case Infinity
  then show ?thesis
    by (simp add: add-def R1)
next
case (Point  $x_1 y_1$ )
note  $Point' = this$ 
with p have  $x_1 \in carrier R$   $y_1 \in carrier R$ 
  and  $p': y_1 [\hat{\phantom{x}}] (2::nat) = x_1 [\hat{\phantom{x}}] (3::nat) \oplus a \otimes x_1 \oplus b$ 
  by (simp-all add: on-curve-def)
show ?thesis
proof (cases q)
  case Infinity
  with Point show ?thesis
    by (simp add: add-def R2)
next
case (Point  $x_2 y_2$ )
with q have  $x_2 \in carrier R$   $y_2 \in carrier R$ 
  and  $q': y_2 [\hat{\phantom{x}}] (2::nat) = x_2 [\hat{\phantom{x}}] (3::nat) \oplus a \otimes x_2 \oplus b$ 
  by (simp-all add: on-curve-def)
show ?thesis
proof (cases  $x_1 = x_2$ )
  case True
  note  $True' = this$ 
  show ?thesis
  proof (cases  $y_1 = \ominus y_2$ )
    case True
    with p Point Point' True' R3 [of p]  $\langle y_2 \in carrier R \rangle$  show ?thesis
      by (simp add: add-def opp-def)
  next
  case False
  have  $(y_1 \ominus y_2) \otimes (y_1 \oplus y_2) = \mathbf{0}$ 
    by (ring True' p' q')
  with False  $\langle y_1 \in carrier R \rangle \langle y_2 \in carrier R \rangle$  have  $y_1 = y_2$ 
    by (simp add: eq-neg-iff-add-eq-0 integral-iff eq-diff0)
  with False True' Point Point' show ?thesis
    apply simp
    apply (rule R4)
    apply (auto simp add: add-def Let-def)
  done
qed
next
case False
with Point Point' show ?thesis
  apply -
  apply (rule R5)
  apply (auto simp add: add-def Let-def)
  done

```

qed  
 qed  
 qed

**lemma** *add-casew* [*consumes 4, case-names InfL InfR Opp Gen*]:

**assumes** *a*:  $a \in \text{carrier } R$   
**and** *b*:  $b \in \text{carrier } R$   
**and** *p*: *on-curve* *a b p*  
**and** *q*: *on-curve* *a b q*  
**and** *R1*:  $\bigwedge p. P \text{ Infinity } p p$   
**and** *R2*:  $\bigwedge p. P p \text{ Infinity } p$   
**and** *R3*:  $\bigwedge p. \text{on-curve } a b p \implies P p (\text{opp } p) \text{ Infinity}$   
**and** *R4*:  $\bigwedge p_1 x_1 y_1 p_2 x_2 y_2 p_3 x_3 y_3 l.$   
 $p_1 = \text{Point } x_1 y_1 \implies p_2 = \text{Point } x_2 y_2 \implies p_3 = \text{Point } x_3 y_3 \implies$   
 $p_3 = \text{add } a p_1 p_2 \implies p_1 \neq \text{opp } p_2 \implies$   
 $x_1 = x_2 \wedge y_1 = y_2 \wedge l = (\llcorner 3 \gg \otimes x_1 [\uparrow] (2::\text{nat}) \oplus a) \otimes (\llcorner 2 \gg \otimes y_1) \vee$   
 $x_1 \neq x_2 \wedge l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \implies$   
 $x_3 = l [\uparrow] (2::\text{nat}) \ominus x_1 \ominus x_2 \implies$   
 $y_3 = \ominus y_1 \ominus l \otimes (x_3 \ominus x_1) \implies$   
 $P p_1 p_2 p_3$   
**shows**  $P p q (\text{add } a p q)$   
**using** *a b p q p q*  
**proof** (*induct rule: add-case*)  
**case** *InfL*  
**show** *?case by (rule R1)*  
**next**  
**case** *InfR*  
**show** *?case by (rule R2)*  
**next**  
**case** (*Opp p*)  
**from**  $\langle \text{on-curve } a b p \rangle$  **show** *?case by (rule R3)*  
**next**  
**case** (*Tan p<sub>1</sub> x<sub>1</sub> y<sub>1</sub> p<sub>2</sub> x<sub>2</sub> y<sub>2</sub> l*)  
**with** *a b* **show** *?case*  
**apply** (*rule-tac R4*)  
**apply** *assumption+*  
**apply** (*simp add: opp-Point equal-neg-zero on-curve-def*)  
**apply** *simp*  
**apply** (*simp add: minus-eq mult2 integral-iff a-assoc r-minus on-curve-def*)  
**apply** *simp*  
**done**  
**next**  
**case** (*Gen p<sub>1</sub> x<sub>1</sub> y<sub>1</sub> p<sub>2</sub> x<sub>2</sub> y<sub>2</sub> p<sub>3</sub> x<sub>3</sub> y<sub>3</sub> l*)  
**then show** *?case*  
**apply** (*rule-tac R4*)  
**apply** *assumption+*  
**apply** (*simp add: opp-Point*)  
**apply** *simp-all*  
**done**

qed

**definition**

*is-tangent*  $p\ q = (p \neq \text{Infinity} \wedge p = q \wedge p \neq \text{opp } q)$

**definition**

*is-generic*  $p\ q =$   
 $(p \neq \text{Infinity} \wedge q \neq \text{Infinity} \wedge$   
 $p \neq q \wedge p \neq \text{opp } q)$

**lemma** *spec1-assoc*:

**assumes**  $a: a \in \text{carrier } R$   
**and**  $b: b \in \text{carrier } R$   
**and**  $p_1: \text{on-curve } a\ b\ p_1$   
**and**  $p_2: \text{on-curve } a\ b\ p_2$   
**and**  $p_3: \text{on-curve } a\ b\ p_3$   
**and** *is-generic*  $p_1\ p_2$   
**and** *is-generic*  $p_2\ p_3$   
**and** *is-generic*  $(\text{add } a\ p_1\ p_2)\ p_3$   
**and** *is-generic*  $p_1\ (\text{add } a\ p_2\ p_3)$   
**shows**  $\text{add } a\ p_1\ (\text{add } a\ p_2\ p_3) = \text{add } a\ (\text{add } a\ p_1\ p_2)\ p_3$   
**using**  $a\ b\ p_1\ p_2\ \text{assms}$   
**proof** (*induct rule: add-case*)  
  **case** *InfL*  
    **show** *?case* **by** (*simp add: add-def*)  
**next**  
  **case** *InfR*  
    **show** *?case* **by** (*simp add: add-def*)  
**next**  
  **case** *Opp*  
    **then show** *?case* **by** (*simp add: is-generic-def*)  
**next**  
  **case** *Tan*  
    **then show** *?case* **by** (*simp add: is-generic-def*)  
**next**  
  **case** (*Gen*  $p_1\ x_1\ y_1\ p_2\ x_2\ y_2\ p_4\ x_4\ y_4\ l$ )  
    **with**  $a\ b\ \langle \text{on-curve } a\ b\ p_2 \rangle\ \langle \text{on-curve } a\ b\ p_3 \rangle$   
    **show** *?case*  
    **proof** (*induct rule: add-case*)  
      **case** *InfL*  
        **then show** *?case* **by** (*simp add: is-generic-def*)  
    **next**  
      **case** *InfR*  
        **then show** *?case* **by** (*simp add: is-generic-def*)  
    **next**  
      **case** *Opp*  
        **then show** *?case* **by** (*simp add: is-generic-def*)  
    **next**  
      **case** *Tan*

```

    then show ?case by (simp add: is-generic-def)
  next
    case (Gen p2 x2' y2' p3 x3 y3 p5 x5 y5 l1)
    from a b ⟨on-curve a b p2⟩ ⟨on-curve a b p3⟩ ⟨p5 = add a p2 p3⟩
    have on-curve a b p5 by (simp add: add-closed)
    with a b ⟨on-curve a b p1⟩ show ?case using Gen [simplified ⟨p2 = Point x2'
y2'⟩]
  proof (induct rule: add-case)
    case InfL
    then show ?case by (simp add: is-generic-def)
  next
    case InfR
    then show ?case by (simp add: is-generic-def)
  next
    case (Opp p)
    from ⟨on-curve a b p⟩ ⟨is-generic p (opp p)⟩
    show ?case by (simp add: is-generic-def opp-opp)
  next
    case Tan
    then show ?case by (simp add: is-generic-def)
  next
    case (Gen p1 x1' y1' p5' x5' y5' p6 x6 y6 l2)
    from a b ⟨on-curve a b p1⟩ ⟨on-curve a b (Point x2' y2)⟩
      ⟨p4 = add a p1 (Point x2' y2)⟩
    have on-curve a b p4 by (simp add: add-closed)
    with a b show ?case using ⟨on-curve a b p3⟩ Gen
  proof (induct rule: add-case)
    case InfL
    then show ?case by (simp add: is-generic-def)
  next
    case InfR
    then show ?case by (simp add: is-generic-def)
  next
    case (Opp p)
    from ⟨on-curve a b p⟩ ⟨is-generic p (opp p)⟩
    show ?case by (simp add: is-generic-def opp-opp)
  next
    case Tan
    then show ?case by (simp add: is-generic-def)
  next
    case (Gen p4' x4' y4' p3' x3' y3' p7 x7 y7 l3)
    from ⟨p4' = Point x4' y4'⟩ ⟨p4' = Point x4 y4⟩
    have p4: x4' = x4 y4' = y4 by simp-all
    from ⟨p3' = Point x3' y3'⟩ ⟨p3' = Point x3 y3⟩
    have p3: x3' = x3 y3' = y3 by simp-all
    from ⟨p1 = Point x1' y1'⟩ ⟨p1 = Point x1 y1⟩
    have p1: x1' = x1 y1' = y1 by simp-all
    from ⟨p5' = Point x5' y5'⟩ ⟨p5' = Point x5 y5⟩
    have p5: x5' = x5 y5' = y5 by simp-all

```

```

from ⟨Point  $x_2' y_2' = Point x_2 y_2$ ⟩
have  $p_2: x_2' = x_2 y_2' = y_2$  by simp-all
note  $ps = p_1 p_2 p_3 p_4 p_5$ 
from
  ⟨on-curve  $a b p_1$ ⟩ ⟨ $p_1 = Point x_1 y_1$ ⟩
  ⟨on-curve  $a b p_2$ ⟩ ⟨ $p_2 = Point x_2 y_2$ ⟩
  ⟨on-curve  $a b p_3$ ⟩ ⟨ $p_3 = Point x_3 y_3$ ⟩
have
   $x_1 \in carrier R y_1 \in carrier R$  and  $y_1: y_1 [\wedge] (2::nat) = x_1 [\wedge] (3::nat) \oplus$ 
 $a \otimes x_1 \oplus b$  and
   $x_2 \in carrier R y_2 \in carrier R$  and  $y_2: y_2 [\wedge] (2::nat) = x_2 [\wedge] (3::nat) \oplus$ 
 $a \otimes x_2 \oplus b$  and
   $x_3 \in carrier R y_3 \in carrier R$  and  $y_3: y_3 [\wedge] (2::nat) = x_3 [\wedge] (3::nat) \oplus$ 
 $a \otimes x_3 \oplus b$ 
  by (simp-all add: on-curve-def)
show ?case
apply (simp add: ⟨ $p_6 = Point x_6 y_6$ ⟩ ⟨ $p_7 = Point x_7 y_7$ ⟩)
apply (simp only: ps)
  ⟨ $x_6 = l_2 [\wedge] 2 \ominus x_1' \ominus x_5'$ ⟩ ⟨ $x_7 = l_3 [\wedge] 2 \ominus x_4' \ominus x_3'$ ⟩
  ⟨ $y_6 = \ominus y_1' \ominus l_2 \otimes (x_6 \ominus x_1')$ ⟩ ⟨ $y_7 = \ominus y_4' \ominus l_3 \otimes (x_7 \ominus x_4')$ ⟩
  ⟨ $l_2 = (y_5' \ominus y_1') \otimes (x_5' \ominus x_1')$ ⟩ ⟨ $l_3 = (y_3' \ominus y_4') \otimes (x_3' \ominus x_4')$ ⟩
  ⟨ $l_1 = (y_3 \ominus y_2') \otimes (x_3 \ominus x_2')$ ⟩ ⟨ $l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1)$ ⟩
  ⟨ $x_5 = l_1 [\wedge] 2 \ominus x_2' \ominus x_3$ ⟩ ⟨ $y_5 = \ominus y_2' \ominus l_1 \otimes (x_5 \ominus x_2')$ ⟩
  ⟨ $x_4 = l [\wedge] 2 \ominus x_1 \ominus x_2$ ⟩ ⟨ $y_4 = \ominus y_1 \ominus l \otimes (x_4 \ominus x_1)$ ⟩)
apply (rule conjI)
apply (field y1 y2 y3)
apply (rule conjI)
apply (simp add: eq-diff0 ⟨ $x_3 \in carrier R$ ⟩ ⟨ $x_2 \in carrier R$ ⟩
  not-sym [OF ⟨ $x_2' \neq x_3$ ⟩ [simplified ⟨ $x_2' = x_2$ ⟩]])
apply (rule conjI)
apply (rule notI)
apply (ring (prems) y1 y2)
apply (cut-tac ⟨ $x_1' \neq x_5'$ ⟩ [simplified ⟨ $x_5' = x_5$ ⟩ ⟨ $x_1' = x_1$ ⟩ ⟨ $x_5 = l_1 [\wedge] 2$ 
 $\ominus x_2' \ominus x_3$ ⟩
  ⟨ $l_1 = (y_3 \ominus y_2') \otimes (x_3 \ominus x_2')$ ⟩ ⟨ $y_2' = y_2$ ⟩ ⟨ $x_2' = x_2$ ⟩])
apply (erule notE)
apply (rule sym)
apply (field y1 y2)
apply (simp add: eq-diff0 ⟨ $x_3 \in carrier R$ ⟩ ⟨ $x_2 \in carrier R$ ⟩
  not-sym [OF ⟨ $x_2' \neq x_3$ ⟩ [simplified ⟨ $x_2' = x_2$ ⟩]])
apply (rule conjI)
apply (simp add: eq-diff0 ⟨ $x_2 \in carrier R$ ⟩ ⟨ $x_1 \in carrier R$ ⟩ not-sym [OF
  ⟨ $x_1 \neq x_2$ ⟩])
apply (rule notI)
apply (ring (prems) y1 y2)
apply (cut-tac ⟨ $x_4' \neq x_3'$ ⟩ [simplified ⟨ $x_4' = x_4$ ⟩ ⟨ $x_3' = x_3$ ⟩ ⟨ $x_4 = l [\wedge] 2$ 
 $\ominus x_1 \ominus x_2$ ⟩
  ⟨ $l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1)$ ⟩])
apply (erule notE)

```

```

    apply (rule sym)
    apply (field y1 y2)
    apply (simp add: eq-diff0 ⟨x2 ∈ carrier R⟩ ⟨x1 ∈ carrier R⟩ not-sym [OF
⟨x1 ≠ x2⟩])
    apply (field y1 y2 y3)
    apply (rule conjI)
    apply (rule notI)
    apply (ring (prems) y1 y2)
    apply (cut-tac ⟨x1' ≠ x5'⟩ [simplified ⟨x5' = x5⟩ ⟨x1' = x1⟩ ⟨x5 = l1 [^] 2
⊖ x2' ⊖ x3⟩
    ⟨l1 = (y3 ⊖ y2') ⊙ (x3 ⊖ x2')⟩ ⟨y2' = y2⟩ ⟨x2' = x2⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field y1 y2)
    apply (simp add: eq-diff0 ⟨x3 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩
not-sym [OF ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩]])
    apply (rule conjI)
    apply (simp add: eq-diff0 ⟨x3 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩
not-sym [OF ⟨x2' ≠ x3⟩ [simplified ⟨x2' = x2⟩]])
    apply (rule conjI)
    apply (rule notI)
    apply (ring (prems) y1 y2)
    apply (cut-tac ⟨x4' ≠ x3'⟩ [simplified ⟨x4' = x4⟩ ⟨x3' = x3⟩ ⟨x4 = l [^] 2
⊖ x1 ⊖ x2⟩
    ⟨l = (y2 ⊖ y1) ⊙ (x2 ⊖ x1)⟩])
    apply (erule notE)
    apply (rule sym)
    apply (field y1 y2)
    apply (simp-all add: eq-diff0 ⟨x2 ∈ carrier R⟩ ⟨x1 ∈ carrier R⟩ not-sym
[OF ⟨x1 ≠ x2⟩])
done
qed
qed
qed
qed

```

**lemma** *spec2-assoc*:

```

assumes a: a ∈ carrier R
and b: b ∈ carrier R
and p1: on-curve a b p1
and p2: on-curve a b p2
and p3: on-curve a b p3
and is-generic p1 p2
and is-tangent p2 p3
and is-generic (add a p1 p2) p3
and is-generic p1 (add a p2 p3)
shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
using a b p1 p2 assms
proof (induct rule: add-case)

```

```

    case InfL
    show ?case by (simp add: add-def)
next
    case InfR
    show ?case by (simp add: add-def)
next
    case Opp
    then show ?case by (simp add: is-generic-def)
next
    case Tan
    then show ?case by (simp add: is-generic-def)
next
    case (Gen  $p_1 x_1 y_1 p_2 x_2 y_2 p_4 x_4 y_4 l$ )
    with  $a b \langle \text{on-curve } a b p_2 \rangle \langle \text{on-curve } a b p_3 \rangle$ 
    show ?case
    proof (induct rule: add-case)
      case InfL
      then show ?case by (simp add: is-generic-def)
    next
      case InfR
      then show ?case by (simp add: is-generic-def)
    next
      case Opp
      then show ?case by (simp add: is-generic-def)
    next
      case (Tan  $p_2 x_2' y_2' p_5 x_5 y_5 l_1$ )
      from  $a b \langle \text{on-curve } a b p_2 \rangle \langle p_5 = \text{add } a p_2 p_2 \rangle$ 
      have  $\text{on-curve } a b p_5$  by (simp add: add-closed)
      with  $a b \langle \text{on-curve } a b p_1 \rangle$  show ?case using Tan
    proof (induct rule: add-case)
      case InfL
      then show ?case by (simp add: is-generic-def)
    next
      case InfR
      then show ?case by (simp add: is-generic-def)
    next
      case (Opp  $p$ )
      from  $\langle \text{is-generic } p (\text{opp } p) \rangle \langle \text{on-curve } a b p \rangle$ 
      show ?case by (simp add: is-generic-def opp-opp)
    next
      case Tan
      then show ?case by (simp add: is-generic-def)
    next
      case (Gen  $p_1 x_1' y_1' p_5' x_5' y_5' p_6 x_6 y_6 l_2$ )
      from  $a b \langle \text{on-curve } a b p_1 \rangle \langle \text{on-curve } a b p_2 \rangle \langle p_4 = \text{add } a p_1 p_2 \rangle$ 
      have  $\text{on-curve } a b p_4$  by (simp add: add-closed)
      with  $a b$  show ?case using  $\langle \text{on-curve } a b p_2 \rangle$  Gen
    proof (induct rule: add-case)
      case InfL

```

```

    then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
  case (Opp p)
  from ⟨is-generic p (opp p)⟩ ⟨on-curve a b p⟩
  show ?case by (simp add: is-generic-def opp-opp)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p4' x4' y4' p3' x3' y3' p7 x7 y7 l3)
  from
    ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
    ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
  have
    x1 ∈ carrier R y1 ∈ carrier R and y1: y1 [∧] (2::nat) = x1 [∧] (3::nat) ⊕
a ⊗ x1 ⊕ b and
    x2 ∈ carrier R y2 ∈ carrier R and y2: y2 [∧] (2::nat) = x2 [∧] (3::nat) ⊕
a ⊗ x2 ⊕ b
  by (simp-all add: on-curve-def)
  from
    ⟨p5' = Point x5' y5'⟩
    ⟨p5' = Point x5 y5⟩
    ⟨p4' = Point x4' y4'⟩
    ⟨p4' = Point x4 y4⟩
    ⟨p3' = Point x2' y2'⟩
    ⟨p3' = Point x2 y2⟩
    ⟨p3' = Point x3' y3'⟩
    ⟨p1 = Point x1' y1'⟩
    ⟨p1 = Point x1 y1⟩
  have ps:
    x5' = x5 y5' = y5
    x4' = x4 y4' = y4 x3' = x2 y3' = y2 x2' = x2 y2' = y2
    x1' = x1 y1' = y1
  by simp-all
  show ?case
  apply (simp add: ⟨p6 = Point x6 y6⟩ ⟨p7 = Point x7 y7⟩)
  apply (simp only: ps
    ⟨x7 = l3 [∧] 2 ⊖ x4' ⊖ x3'⟩
    ⟨y7 = ⊖ y4' ⊖ l3 ⊗ (x7 ⊖ x4')⟩
    ⟨l3 = (y3' ⊖ y4') ⊗ (x3' ⊖ x4')⟩
    ⟨x6 = l2 [∧] 2 ⊖ x1' ⊖ x5'⟩
    ⟨y6 = ⊖ y1' ⊖ l2 ⊗ (x6 ⊖ x1')⟩
    ⟨l2 = (y5' ⊖ y1') ⊗ (x5' ⊖ x1')⟩
    ⟨x5 = l1 [∧] 2 ⊖ «2» ⊗ x2'⟩
    ⟨y5 = ⊖ y2' ⊖ l1 ⊗ (x5 ⊖ x2')⟩
    ⟨l1 = («3» ⊗ x2' [∧] 2 ⊕ a) ⊗ («2» ⊗ y2')⟩

```



$\langle x_4 = l [\ulcorner] 2 \ominus x_1 \ominus x_2 \rangle$   
 $\langle y_4 = \ominus y_1 \ominus l \otimes (x_4 \ominus x_1) \rangle$   
 $\langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle$   
**apply** (*rule conjI*)  
**apply** (*field y1 y2*)  
**apply** (*intro conjI*)  
**apply** (*simp add: integral-iff* [*OF* -  $\langle y_2 \in \text{carrier } R \rangle$ ]  $\langle y_2' \neq \mathbf{0} \rangle$  [*simplified*  
 $\langle y_2' = y_2 \rangle$ ])  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE* [*OF*  $\langle x_1' \neq x_5' \rangle$ ] [*simplified*  
 $\langle x_5 = l_1 [\ulcorner] 2 \ominus \langle 2 \rangle \otimes x_2' \rangle$   
 $\langle l_1 = (\langle 3 \rangle \otimes x_2' [\ulcorner] 2 \oplus a) \otimes (\langle 2 \rangle \otimes y_2' \rangle$   
 $\langle x_1' = x_1 \rangle \langle x_2' = x_2 \rangle \langle y_2' = y_2 \rangle \langle x_5' = x_5 \rangle$ ])])  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add: integral-iff* [*OF* -  $\langle y_2 \in \text{carrier } R \rangle$ ]  $\langle y_2' \neq \mathbf{0} \rangle$  [*simplified*  
 $\langle y_2' = y_2 \rangle$ ])  
**apply** (*simp add: eq-diff0*  $\langle x_2 \in \text{carrier } R \rangle \langle x_1 \in \text{carrier } R \rangle$  *not-sym* [*OF*  
 $\langle x_1 \neq x_2 \rangle$ ])  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE* [*OF*  $\langle x_4' \neq x_3' \rangle$ ] [*simplified*  
 $\langle x_4 = l [\ulcorner] 2 \ominus x_1 \ominus x_2 \rangle$   
 $\langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle$   
 $\langle x_4' = x_4 \rangle \langle x_3' = x_2 \rangle$ ])])  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add: eq-diff0*  $\langle x_2 \in \text{carrier } R \rangle \langle x_1 \in \text{carrier } R \rangle$  *not-sym* [*OF*  
 $\langle x_1 \neq x_2 \rangle$ ])  
**apply** (*field y1 y2*)  
**apply** (*intro conjI*)  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE* [*OF*  $\langle x_1' \neq x_5' \rangle$ ] [*simplified*  
 $\langle x_5 = l_1 [\ulcorner] 2 \ominus \langle 2 \rangle \otimes x_2' \rangle$   
 $\langle l_1 = (\langle 3 \rangle \otimes x_2' [\ulcorner] 2 \oplus a) \otimes (\langle 2 \rangle \otimes y_2' \rangle$   
 $\langle x_1' = x_1 \rangle \langle x_2' = x_2 \rangle \langle y_2' = y_2 \rangle \langle x_5' = x_5 \rangle$ ])])  
**apply** (*rule sym*)  
**apply** (*field y1 y2*)  
**apply** (*simp add: integral-iff* [*OF* -  $\langle y_2 \in \text{carrier } R \rangle$ ]  $\langle y_2' \neq \mathbf{0} \rangle$  [*simplified*  
 $\langle y_2' = y_2 \rangle$ ])  
**apply** (*simp add: integral-iff* [*OF* -  $\langle y_2 \in \text{carrier } R \rangle$ ]  $\langle y_2' \neq \mathbf{0} \rangle$  [*simplified*  
 $\langle y_2' = y_2 \rangle$ ])  
**apply** (*rule notI*)  
**apply** (*ring (prems) y1 y2*)  
**apply** (*rule notE* [*OF*  $\langle x_4' \neq x_3' \rangle$ ] [*simplified*  
 $\langle x_4 = l [\ulcorner] 2 \ominus x_1 \ominus x_2 \rangle$   
 $\langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle$

```

      ⟨ $x_4' = x_4$ ⟩ ⟨ $x_3' = x_2$ ⟩]])
    apply (rule sym)
    apply (field y1 y2)
    apply (simp-all add: eq-diff0 ⟨ $x_2 \in \text{carrier } R$ ⟩ ⟨ $x_1 \in \text{carrier } R$ ⟩ not-sym
[OF ⟨ $x_1 \neq x_2$ ⟩])
  done
  qed
  qed
  next
  case (Gen p3 x3 y3 p5 x5 y5 p6 x6 y6 l1)
  then show ?case by (simp add: is-tangent-def)
  qed
  qed

```

lemma spec3-assoc:

```

  assumes a:  $a \in \text{carrier } R$ 
  and b:  $b \in \text{carrier } R$ 
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and p3: on-curve a b p3
  and is-generic p1 p2
  and is-tangent p2 p3
  and is-generic (add a p1 p2) p3
  and is-tangent p1 (add a p2 p3)
  shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
  using a b p1 p2 assms
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
  case Opp
  then show ?case by (simp add: is-generic-def)
next
  case Tan
  then show ?case by (simp add: is-generic-def)
next
  case (Gen p1 x1 y1 p2 x2 y2 p4 x4 y4 l)
  with a b ⟨on-curve a b p2⟩ ⟨on-curve a b p3⟩
  show ?case
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next

```

```

    case Opp
  then show ?case by (simp add: is-tangent-def opp-opp)
next
case (Tan p2 x2' y2' p5 x5 y5 l1)
from a b ⟨on-curve a b p2⟩ ⟨p5 = add a p2 p2⟩
have on-curve a b p5 by (simp add: add-closed)
with a b ⟨on-curve a b p1⟩ show ?case using Tan
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
  case Opp
  then show ?case by (simp add: is-tangent-def opp-opp)
next
case (Tan p1 x1' y1' p6 x6 y6 l2)
from a b ⟨on-curve a b p1⟩ ⟨on-curve a b p2⟩ ⟨p4 = add a p1 p2⟩
have on-curve a b p4 by (simp add: add-closed)
with a b show ?case using ⟨on-curve a b p2⟩ Tan
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: is-generic-def)
next
  case InfR
  then show ?case by (simp add: is-generic-def)
next
case (Opp p)
from ⟨is-generic p (opp p)⟩ ⟨on-curve a b p⟩
show ?case by (simp add: is-generic-def opp-opp)
next
case Tan
then show ?case by (simp add: is-generic-def)
next
case (Gen p4' x4' y4' p2' x2'' y2'' p7 x7 y7 l3)
from
  ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
  ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
have
  x1 ∈ carrier R y1 ∈ carrier R and y1: y1 [∧] (2::nat) = x1 [∧] (3::nat) ⊕
a ⊗ x1 ⊕ b and
  x2 ∈ carrier R y2 ∈ carrier R and y2: y2 [∧] (2::nat) = x2 [∧] (3::nat) ⊕
a ⊗ x2 ⊕ b
  by (simp-all add: on-curve-def)
from
  ⟨p4' = Point x4' y4'⟩
  ⟨p4' = Point x4 y4⟩
  ⟨p2' = Point x2' y2'⟩

```

```

⟨p2' = Point x2 y2⟩
⟨p2' = Point x2'' y2''⟩
⟨p1 = Point x1' y1'⟩
⟨p1 = Point x1 y1⟩
⟨p1 = Point x5 y5⟩
have ps:
  x4' = x4 y4' = y4 x2' = x2 y2' = y2 x2'' = x2 y2'' = y2
  x1' = x5 y1' = y5 x1 = x5 y1 = y5
  by simp-all
note qs =
  ⟨x7 = l3 [∧] 2 ⊖ x4' ⊖ x2''⟩
  ⟨y7 = ⊖ y4' ⊖ l3 ⊗ (x7 ⊖ x4')⟩
  ⟨l3 = (y2'' ⊖ y4') ⊙ (x2'' ⊖ x4')⟩
  ⟨x6 = l2 [∧] 2 ⊖ «2» ⊗ x1'⟩
  ⟨y6 = ⊖ y1' ⊖ l2 ⊗ (x6 ⊖ x1')⟩
  ⟨x5 = l1 [∧] 2 ⊖ «2» ⊗ x2'⟩
  ⟨y5 = ⊖ y2' ⊖ l1 ⊗ (x5 ⊖ x2')⟩
  ⟨l1 = («3» ⊗ x2' [∧] 2 ⊕ a) ⊙ («2» ⊗ y2')⟩
  ⟨l2 = («3» ⊗ x1' [∧] 2 ⊕ a) ⊙ («2» ⊗ y1')⟩
  ⟨x4 = l [∧] 2 ⊖ x1 ⊖ x2⟩
  ⟨y4 = ⊖ y1 ⊖ l ⊗ (x4 ⊖ x1)⟩
  ⟨l = (y2 ⊖ y1) ⊙ (x2 ⊖ x1)⟩
from ⟨y2 ∈ carrier R⟩ ⟨y2' ≠ 0⟩ ⟨y2' = y2⟩
have «2» ⊗ y2 ≠ 0 by (simp add: integral-iff)
show ?case
  apply (simp add: ⟨p6 = Point x6 y6⟩ ⟨p7 = Point x7 y7⟩)
  apply (simp only: ps qs)
  apply (rule conjI)
  apply (field y2)
  apply (intro conjI)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨y1' ≠ 0⟩])
  apply (simp only: ps qs)
  apply field
  apply (rule «2» ⊗ y2 ≠ 0)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨x1 ≠ x2⟩])
  apply (rule sym)
  apply (simp only: ps qs)
  apply field
  apply (rule «2» ⊗ y2 ≠ 0)
  apply (rule «2» ⊗ y2 ≠ 0)
  apply (rule notI)
  apply (ring (prems))
  apply (rule notE [OF ⟨x4' ≠ x2''⟩])
  apply (rule sym)
  apply (simp only: ps qs)

```

```

apply field
apply (intro conjI)
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (erule thin-rl)
apply (rule notI)
apply (ring (prems))
apply (rule notE [OF ‹x₁ ≠ x₂›])
apply (rule sym)
apply (simp only: ps qs)
apply field
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (field y₂)
apply (intro conjI)
apply (rule notI)
apply (ring (prems))
apply (rule notE [OF ‹y₁' ≠ 0›])
apply (simp only: ps qs)
apply field
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (rule notI)
apply (ring (prems))
apply (rule notE [OF ‹x₄' ≠ x₂''›])
apply (rule sym)
apply (simp only: ps qs)
apply field
apply (erule thin-rl)
apply (rule conjI)
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (rule notI)
apply (ring (prems))
apply (rule notE [OF ‹x₁ ≠ x₂›])
apply (rule sym)
apply (simp only: ps qs)
apply field
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
apply (rule notI)
apply (ring (prems))
apply (rule notE [OF ‹x₁ ≠ x₂›])
apply (rule sym)
apply (simp only: ps qs)
apply field
apply (rule ‹«2» ⊗ y₂ ≠ 0›)
done
qed
next
case Gen
then show ?case by (simp add: is-tangent-def)
qed

```

```

next
  case Gen
  then show ?case by (simp add: is-tangent-def)
qed
qed

lemma add-0-l: add a Infinity p = p
  by (simp add: add-def)

lemma add-0-r: add a p Infinity = p
  by (simp add: add-def split: point.split)

lemma add-opp: on-curve a b p  $\implies$  add a p (opp p) = Infinity
  by (simp add: add-def opp-def on-curve-def split: point.split-asm)

lemma add-comm:
  assumes a  $\in$  carrier R b  $\in$  carrier R on-curve a b p1 on-curve a b p2
  shows add a p1 p2 = add a p2 p1
proof (cases p1)
  case Infinity
  then show ?thesis by (simp add: add-0-l add-0-r)
next
  case (Point x1 y1)
  note Point' = this
  with  $\langle$ on-curve a b p1 $\rangle$ 
  have x1  $\in$  carrier R y1  $\in$  carrier R
    and y1: y1 [∧] (2::nat) = x1 [∧] (3::nat)  $\oplus$  a  $\otimes$  x1  $\oplus$  b
    by (simp-all add: on-curve-def)
  show ?thesis
proof (cases p2)
  case Infinity
  then show ?thesis by (simp add: add-0-l add-0-r)
next
  case (Point x2 y2)
  with  $\langle$ on-curve a b p2 $\rangle$  have x2  $\in$  carrier R y2  $\in$  carrier R
    and y2: y2 [∧] (2::nat) = x2 [∧] (3::nat)  $\oplus$  a  $\otimes$  x2  $\oplus$  b
    by (simp-all add: on-curve-def)
  show ?thesis
proof (cases x1 = x2)
  case True
  show ?thesis
proof (cases y1 =  $\ominus$  y2)
  case True
  with Point Point'  $\langle$ x1 = x2 $\rangle$   $\langle$ y2  $\in$  carrier R $\rangle$  show ?thesis
  by (simp add: add-def)
next
  case False
  with y1 y2 [symmetric]  $\langle$ y1  $\in$  carrier R $\rangle$   $\langle$ y2  $\in$  carrier R $\rangle$   $\langle$ x1 = x2 $\rangle$  Point
Point'

```

```

    show ?thesis
      by (simp add: power2-eq-square square-eq-iff)
  qed
next
case False
with Point Point' show ?thesis
  apply (simp add: add-def Let-def)
  apply (rule conjI)
  apply field
  apply (cut-tac ⟨x1 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩)
  apply (simp add: eq-diff0)
  apply field
  apply (cut-tac ⟨x1 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩)
  apply (simp add: eq-diff0)
done
  qed
qed
qed

lemma uniq-opp:
  assumes on-curve a b p2
  and add a p1 p2 = Infinity
  shows p2 = opp p1
  using assms
  by (auto simp add: on-curve-def add-def opp-def Let-def
    split: point.split-asm if-split-asm)

lemma uniq-zero:
  assumes a: a ∈ carrier R
  and b: b ∈ carrier R
  and ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and add: add a p1 p2 = p2
  shows p1 = Infinity
  using a b p1 p2 assms
proof (induct rule: add-case)
  case InfL
  show ?case ..
next
  case InfR
  then show ?case by simp
next
  case Opp
  then show ?case by (simp add: opp-def split: point.split-asm)
next
  case (Tan p1 x1 y1 p2 x2 y2 l)
  from ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
  have x1 ∈ carrier R y1 ∈ carrier R by (simp-all add: on-curve-def)

```

**with**  $a \langle l = (\langle \langle 3 \rangle \rangle \otimes x_1 [\uparrow] 2 \oplus a) \otimes (\langle \langle 2 \rangle \rangle \otimes y_1) \rangle \langle y_1 \neq \mathbf{0} \rangle$   
**have**  $l \in \text{carrier } R$  **by** (*simp add: integral-iff*)  
**from**  $\langle p_1 = \text{Point } x_1 y_1 \rangle \langle p_2 = \text{Point } x_2 y_2 \rangle \langle p_2 = p_1 \rangle$   
**have**  $x_2 = x_1 y_2 = y_1$  **by** *simp-all*  
**with**  $\langle x_1 \in \text{carrier } R \rangle \langle y_1 \in \text{carrier } R \rangle \langle l \in \text{carrier } R \rangle \langle y_2 = \ominus y_1 \ominus l \otimes (x_2 \ominus x_1) \rangle \langle y_1 \neq \mathbf{0} \rangle$   
**have**  $\ominus y_1 = y_1$  **by** (*simp add: r-neg minus-eq*)  
**with**  $\langle y_1 \in \text{carrier } R \rangle \langle y_1 \neq \mathbf{0} \rangle$   
**show** *?case* **by** (*simp add: neg-equal-zero*)  
**next**  
**case** (*Gen*  $p_1 x_1 y_1 p_2 x_2 y_2 p_3 x_3 y_3 l$ )  
**then have**  $x_1 \in \text{carrier } R y_1 \in \text{carrier } R x_2 \in \text{carrier } R y_2 \in \text{carrier } R$   
**and**  $y_1: y_1 [\uparrow] (2::\text{nat}) = x_1 [\uparrow] (3::\text{nat}) \oplus a \otimes x_1 \oplus b$   
**and**  $y_2: y_2 [\uparrow] (2::\text{nat}) = x_2 [\uparrow] (3::\text{nat}) \oplus a \otimes x_2 \oplus b$   
**by** (*simp-all add: on-curve-def*)  
**with**  $\langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle \langle x_1 \neq x_2 \rangle$   
**have**  $l \in \text{carrier } R$  **by** (*simp add: eq-diff0*)  
**from**  $\langle p_3 = p_2 \rangle \langle p_2 = \text{Point } x_2 y_2 \rangle \langle p_3 = \text{Point } x_3 y_3 \rangle$   
**have**  $ps: x_3 = x_2 y_3 = y_2$  **by** *simp-all*  
**with**  $\langle y_3 = \ominus y_1 \ominus l \otimes (x_3 \ominus x_1) \rangle$   
**have**  $y_2 = \ominus y_1 \ominus l \otimes (x_2 \ominus x_1)$  **by** *simp*  
**also from**  $\langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle \langle x_1 \neq x_2 \rangle$   
 $\langle x_1 \in \text{carrier } R \rangle \langle x_2 \in \text{carrier } R \rangle \langle y_1 \in \text{carrier } R \rangle \langle y_2 \in \text{carrier } R \rangle$   
**have**  $l \otimes (x_2 \ominus x_1) = y_2 \ominus y_1$   
**by** (*simp add: m-div-def m-assoc eq-diff0*)  
**also from**  $\langle y_1 \in \text{carrier } R \rangle \langle y_2 \in \text{carrier } R \rangle$   
**have**  $\ominus y_1 \ominus (y_2 \ominus y_1) = (\ominus y_1 \oplus y_1) \oplus \ominus y_2$   
**by** (*simp add: minus-eq minus-add a-ac*)  
**finally have**  $y_2 = \mathbf{0}$  **using**  $\langle y_1 \in \text{carrier } R \rangle \langle y_2 \in \text{carrier } R \rangle$   
**by** (*simp add: l-neg equal-neg-zero*)  
**with**  $\langle p_2 = \text{Point } x_2 y_2 \rangle \langle \text{on-curve } a b p_2 \rangle$   
 $\langle a \in \text{carrier } R \rangle \langle b \in \text{carrier } R \rangle \langle x_2 \in \text{carrier } R \rangle$   
**have**  $x_2: x_2 [\uparrow] (3::\text{nat}) = \ominus (a \otimes x_2 \oplus b)$   
**by** (*simp add: on-curve-def nat-pow-zero eq-neg-iff-add-eq-0 a-assoc*)  
**from**  $\langle x_3 = l [\uparrow] 2 \ominus x_1 \ominus x_2 \rangle \langle x_3 = x_2 \rangle$   
**have**  $l [\uparrow] (2::\text{nat}) \ominus x_1 \ominus x_2 \ominus x_2 = x_2 \ominus x_2$  **by** *simp*  
**with**  $\langle x_1 \in \text{carrier } R \rangle \langle x_2 \in \text{carrier } R \rangle \langle l \in \text{carrier } R \rangle$   
**have**  $l [\uparrow] (2::\text{nat}) \ominus x_1 \ominus \langle \langle 2 \rangle \rangle \otimes x_2 = \mathbf{0}$   
**by** (*simp add: of-int-2 l-distr minus-eq a-ac minus-add r-neg*)  
**then have**  $x_2 \otimes (l [\uparrow] (2::\text{nat}) \ominus x_1 \ominus \langle \langle 2 \rangle \rangle \otimes x_2) = x_2 \otimes \mathbf{0}$  **by** *simp*  
**then have**  $(x_2 \ominus x_1) \otimes (\langle \langle 2 \rangle \rangle \otimes a \otimes x_2 \oplus \langle \langle 3 \rangle \rangle \otimes b) = \mathbf{0}$   
**apply** (*simp add: \langle l = (y\_2 \ominus y\_1) \otimes (x\_2 \ominus x\_1) \rangle \langle y\_2 = \mathbf{0} \rangle*)  
**apply** (*field (prems) y1 x2*)  
**apply** (*ring y1 x2*)  
**apply** (*simp add: eq-diff0 \langle x\_2 \in \text{carrier } R \rangle \langle x\_1 \in \text{carrier } R \rangle not-sym [OF \langle x\_1 \neq x\_2 \rangle]*)  
**done**  
**with** *not-sym [OF \langle x\_1 \neq x\_2 \rangle]*  
 $\langle x_2 \in \text{carrier } R \rangle \langle x_1 \in \text{carrier } R \rangle \langle a \in \text{carrier } R \rangle \langle b \in \text{carrier } R \rangle$



```

have «2»  $\otimes a \otimes x_2 \oplus \llbracket 3 \rrbracket \otimes b = \mathbf{0}$ 
  by (simp add: integral-iff eq-diff0)
with  $\langle a \in \text{carrier } R \rangle \langle b \in \text{carrier } R \rangle \langle x_2 \in \text{carrier } R \rangle$ 
have «2»  $\otimes a \otimes x_2 = \ominus (\llbracket 3 \rrbracket \otimes b)$ 
  by (simp add: eq-neg-iff-add-eq-0)
from  $y_2$  [symmetric]  $\langle y_2 = \mathbf{0} \rangle \langle a \in \text{carrier } R \rangle$ 
have  $\ominus (\llbracket 2 \rrbracket \otimes a) [\wedge] (3::\text{nat}) \otimes (x_2 [\wedge] (3::\text{nat}) \oplus a \otimes x_2 \oplus b) = \mathbf{0}$ 
  by (simp add: nat-pow-zero)
then have  $b \otimes (\llbracket 4 \rrbracket \otimes a [\wedge] (3::\text{nat}) \oplus \llbracket 27 \rrbracket \otimes b [\wedge] (2::\text{nat})) = \mathbf{0}$ 
  apply (ring (prems)  $\langle \llbracket 2 \rrbracket \otimes a \otimes x_2 = \ominus (\llbracket 3 \rrbracket \otimes b) \rangle$ )
  apply (ring  $\langle \llbracket 2 \rrbracket \otimes a \otimes x_2 = \ominus (\llbracket 3 \rrbracket \otimes b) \rangle$ )
done
with  $ab$   $a$   $b$  have  $b = \mathbf{0}$  by (simp add: nonsingular-def integral-iff)
with  $\langle \llbracket 2 \rrbracket \otimes a \otimes x_2 \oplus \llbracket 3 \rrbracket \otimes b = \mathbf{0} \rangle \langle ab \ a \ b \ \langle x_2 \in \text{carrier } R \rangle$ 
have  $x_2 = \mathbf{0}$  by (simp add: nonsingular-def nat-pow-zero integral-iff)
from  $\langle l [\wedge] (2::\text{nat}) \ominus x_1 \ominus \llbracket 2 \rrbracket \otimes x_2 = \mathbf{0} \rangle$ 
show ?case
  apply (simp add:  $\langle x_2 = \mathbf{0} \rangle \langle y_2 = \mathbf{0} \rangle \langle l = (y_2 \ominus y_1) \otimes (x_2 \ominus x_1) \rangle$ )
  apply (field (prems)  $y_1 \ \langle b = \mathbf{0} \rangle$ )
  apply (insert  $a \ b \ ab \ \langle x_1 \in \text{carrier } R \rangle \langle b = \mathbf{0} \rangle \langle x_1 \neq x_2 \rangle \langle x_2 = \mathbf{0} \rangle$ )
  apply (simp add: nonsingular-def nat-pow-zero integral-iff)
  apply (simp add: trans [OF eq-commute eq-neg-iff-add-eq-0])
done
qed

```

lemma opp-add:

```

assumes  $a: a \in \text{carrier } R$ 
and  $b: b \in \text{carrier } R$ 
and  $p_1: \text{on-curve } a \ b \ p_1$ 
and  $p_2: \text{on-curve } a \ b \ p_2$ 
shows  $\text{opp } (\text{add } a \ p_1 \ p_2) = \text{add } a \ (\text{opp } p_1) \ (\text{opp } p_2)$ 
proof (cases  $p_1$ )
  case Infinity
  then show ?thesis by (simp add: add-def opp-def)
next
  case (Point  $x_1 \ y_1$ )
  show ?thesis
  proof (cases  $p_2$ )
    case Infinity
    with  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle$  show ?thesis
    by (simp add: add-def opp-def)
  next
    case (Point  $x_2 \ y_2$ )
    with  $\langle p_1 = \text{Point } x_1 \ y_1 \rangle \ p_1 \ p_2$ 
    have  $x_1 \in \text{carrier } R \ y_1 \in \text{carrier } R \ x_1 [\wedge] (3::\text{nat}) \oplus a \otimes x_1 \oplus b = y_1 [\wedge] (2::\text{nat})$ 
       $x_2 \in \text{carrier } R \ y_2 \in \text{carrier } R \ x_2 [\wedge] (3::\text{nat}) \oplus a \otimes x_2 \oplus b = y_2 [\wedge] (2::\text{nat})$ 
    by (simp-all add: on-curve-def)
    with  $\text{Point } \langle p_1 = \text{Point } x_1 \ y_1 \rangle$  show ?thesis

```

```

apply (cases  $x_1 = x_2$ )
apply (cases  $y_1 = \ominus y_2$ )
apply (simp add: add-def opp-def Let-def)
apply (simp add: add-def opp-def Let-def neg-equal-swap)
apply (rule conjI)
apply field
apply (auto simp add: integral-iff nat-pow-zero
  trans [OF eq-commute eq-neg-iff-add-eq-0])[1]
apply field
apply (auto simp add: integral-iff nat-pow-zero
  trans [OF eq-commute eq-neg-iff-add-eq-0])[1]
apply (simp add: add-def opp-def Let-def)
apply (rule conjI)
apply field
apply (simp add: eq-diff0)
apply field
apply (simp add: eq-diff0)
done
qed
qed

lemma compat-add-opp:
  assumes  $a: a \in \text{carrier } R$ 
  and  $b: b \in \text{carrier } R$ 
  and  $p_1: \text{on-curve } a \ b \ p_1$ 
  and  $p_2: \text{on-curve } a \ b \ p_2$ 
  and  $\text{add } a \ p_1 \ p_2 = \text{add } a \ p_1 \ (\text{opp } p_2)$ 
  and  $p_1 \neq \text{opp } p_1$ 
  shows  $p_2 = \text{opp } p_2$ 
  using  $a \ b \ p_1 \ p_2 \ \text{assms}$ 
proof (induct rule: add-case)
  case InfL
  then show ?case by (simp add: add-0-l)
next
  case InfR
  then show ?case by (simp add: opp-def add-0-r)
next
  case (Opp  $p$ )
  then have  $\text{add } a \ p \ p = \text{Infinity}$  by (simp add: opp-opp)
  with  $\langle \text{on-curve } a \ b \ p \rangle$  have  $p = \text{opp } p$  by (rule uniq-opp)
  with  $\langle p \neq \text{opp } p \rangle$  show ?case ..
next
  case (Tan  $p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ l$ )
  then have  $\text{add } a \ p_1 \ p_1 = \text{Infinity}$ 
  by (simp add: add-opp)
  with  $\langle \text{on-curve } a \ b \ p_1 \rangle$  have  $p_1 = \text{opp } p_1$  by (rule uniq-opp)
  with  $\langle p_1 \neq \text{opp } p_1 \rangle$  show ?case ..
next
  case (Gen  $p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ p_3 \ x_3 \ y_3 \ l$ )

```

```

then have  $x_1 \in \text{carrier } R$   $y_1 \in \text{carrier } R$   $x_2 \in \text{carrier } R$   $y_2 \in \text{carrier } R$ 
  by (simp-all add: on-curve-def)
have  $\langle 2 \rangle \otimes \langle 2 \rangle \neq \mathbf{0}$ 
  by (simp add: integral-iff)
then have  $\langle 4 \rangle \neq \mathbf{0}$  by (simp add: of-int-mult [symmetric])
from Gen have  $((\ominus y_2 \ominus y_1) \otimes (x_2 \ominus x_1)) [\uparrow] (2::\text{nat}) \ominus x_1 \ominus x_2 =$ 
   $((y_2 \ominus y_1) \otimes (x_2 \ominus x_1)) [\uparrow] (2::\text{nat}) \ominus x_1 \ominus x_2$ 
  by (simp add: add-def opp-def Let-def)
then show ?case
  apply (field (prems))
  apply (insert  $\langle y_1 \in \text{carrier } R \rangle$   $\langle y_2 \in \text{carrier } R \rangle$   $\langle p_1 \neq \text{opp } p_1 \rangle$ 
     $\langle p_1 = \text{Point } x_1 y_1 \rangle$   $\langle p_2 = \text{Point } x_2 y_2 \rangle$   $\langle \langle 4 \rangle \neq \mathbf{0} \rangle$  [1])
  apply (simp add: integral-iff opp-def eq-neg-iff-add-eq-0 mult2)
  apply (insert  $\langle x_1 \in \text{carrier } R \rangle$   $\langle x_2 \in \text{carrier } R \rangle$   $\langle x_1 \neq x_2 \rangle$ )
  apply (simp add: eq-diff0)
  done
qed

lemma compat-add-triple:
  assumes  $a: a \in \text{carrier } R$ 
  and  $b: b \in \text{carrier } R$ 
  and  $ab: \text{nonsingular } a b$ 
  and  $p: \text{on-curve } a b p$ 
  and  $p \neq \text{opp } p$ 
  and  $\text{add } a p p \neq \text{opp } p$ 
  shows  $\text{add } a (\text{add } a p p) (\text{opp } p) = p$ 
  using  $a b$  add-closed [OF a b p p] opp-closed [OF p] assms
proof (induct add a p p opp p rule: add-case)
  case InfL
  from  $\langle p \neq \text{opp } p \rangle$  uniq-opp [OF p  $\langle \text{Infinity} = \text{add } a p p \rangle$  [symmetric]]
  show ?case ..
next
  case InfR
  then show ?case by (simp add: opp-def split: point.split-asm)
next
  case Opp
  then have  $\text{opp } (\text{opp } (\text{add } a p p)) = \text{opp } (\text{opp } p)$  by simp
  with  $\langle \text{on-curve } a b (\text{add } a p p) \rangle$   $\langle \text{on-curve } a b p \rangle$ 
  have  $\text{add } a p p = p$  by (simp add: opp-opp)
  with uniq-zero [OF a b ab p p]  $\langle p \neq \text{opp } p \rangle$ 
  show ?case by (simp add: opp-def)
next
  case Tan
  then show ?case by simp
next
  case (Gen  $x_1 y_1 x_2 y_2 p_3 x_3 y_3 l$ )
  with opp-closed [OF p]
  have  $x_2 \in \text{carrier } R$   $y_2 \in \text{carrier } R$ 
  by (simp-all add: on-curve-def)

```

```

from ⟨opp p = Point x2 y2⟩ p
have p = Point x2 (⊖ y2)
  by (auto simp add: opp-def on-curve-def neg-equal-swap split: point.split-asm)
with ⟨add a p p = Point x1 y1⟩ [symmetric]
obtain l' where l':
  l' = (⟨3⟩ ⊗ x2 [⌈] (2::nat) ⊕ a) ⊗ (⟨2⟩ ⊗ ⊖ y2)
  and xy: x1 = l' [⌈] (2::nat) ⊖ ⟨2⟩ ⊗ x2
  y1 = ⊖ (⊖ y2) ⊖ l' ⊗ (x1 ⊖ x2)
  and y2: ⊖ y2 ≠ ⊖ (⊖ y2)
  by (simp add: add-def Let-def split: if-split-asm)
from l' ⟨x2 ∈ carrier R⟩ ⟨y2 ∈ carrier R⟩ a y2
have l' ∈ carrier R by (simp add: neg-equal-zero neg-equal-swap integral-iff)
have x3 = x2
  apply (simp add: xy)
  ⟨l = (y2 ⊖ y1) ⊗ (x2 ⊖ x1)⟩ ⟨x3 = l [⌈] 2 ⊖ x1 ⊖ x2⟩
  apply field
  apply (insert ⟨x1 ≠ x2⟩ ⟨x2 ∈ carrier R⟩ ⟨l' ∈ carrier R⟩)
  apply (simp add: xy eq-diff0)
  done
then have p3 = p ∨ p3 = opp p
  by (rule curve-elt-opp [OF ⟨p3 = Point x3 y3⟩ ⟨p = Point x2 (⊖ y2)⟩
    add-closed [OF a b add-closed [OF a b p p] opp-closed [OF p],
    folded ⟨p3 = add a (add a p p) (opp p)⟩
    ⟨on-curve a b p⟩])
then show ?case
proof
  assume p3 = p
  with ⟨p3 = add a (add a p p) (opp p)⟩
  show ?thesis by simp
next
  assume p3 = opp p
  with ⟨p3 = add a (add a p p) (opp p)⟩
  have add a (add a p p) (opp p) = opp p by simp
  with a b ab add-closed [OF a b p p] opp-closed [OF p]
  have add a p p = Infinity by (rule uniq-zero)
  with ⟨add a p p = Point x1 y1⟩ show ?thesis by simp
qed
qed

lemma add-opp-double-opp:
  assumes a: a ∈ carrier R
  and b: b ∈ carrier R
  and ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and add a p1 p2 = opp p1
  shows p2 = add a (opp p1) (opp p1)
proof (cases p1 = opp p1)
  case True

```

```

with assms have  $add\ a\ p_2\ p_1 = p_1$  by (simp add: add-comm)
with  $a\ b\ ab\ p_2\ p_1$  have  $p_2 = Infinity$  by (rule uniq-zero)
also from  $\langle on-curve\ a\ b\ p_1 \rangle$  have  $\dots = add\ a\ p_1\ (opp\ p_1)$ 
  by (simp add: add-opp)
also from True have  $\dots = add\ a\ (opp\ p_1)\ (opp\ p_1)$  by simp
finally show ?thesis .
next
case False
from  $a\ b\ p_1\ p_2\ False\ assms$  show ?thesis
proof (induct rule: add-case)
  case InfL
  then show ?case by simp
next
  case InfR
  then show ?case by simp
next
  case Opp
  then show ?case by (simp add: add-0-l)
next
  case (Tan  $p_1\ x_1\ y_1\ p_2\ x_2\ y_2\ l$ )
  from  $\langle p_2 = opp\ p_1 \rangle\ \langle on-curve\ a\ b\ p_1 \rangle$ 
  have  $p_1 = opp\ p_2$  by (simp add: opp-opp)
  also note  $\langle p_2 = add\ a\ p_1\ p_1 \rangle$ 
  finally show ?case using  $a\ b\ \langle on-curve\ a\ b\ p_1 \rangle$ 
    by (simp add: opp-add)
next
  case (Gen  $p_1\ x_1\ y_1\ p_2\ x_2\ y_2\ p_3\ x_3\ y_3\ l$ )
  from  $\langle on-curve\ a\ b\ p_1 \rangle\ \langle p_1 = Point\ x_1\ y_1 \rangle$ 
  have  $x_1 \in carrier\ R\ y_1 \in carrier\ R$ 
    and  $y_1: y_1 [\wedge] (2::nat) = x_1 [\wedge] (3::nat) \oplus a \otimes x_1 \oplus b$ 
    by (simp-all add: on-curve-def)
  from  $\langle on-curve\ a\ b\ p_2 \rangle\ \langle p_2 = Point\ x_2\ y_2 \rangle$ 
  have  $x_2 \in carrier\ R\ y_2 \in carrier\ R$ 
    and  $y_2: y_2 [\wedge] (2::nat) = x_2 [\wedge] (3::nat) \oplus a \otimes x_2 \oplus b$ 
    by (simp-all add: on-curve-def)
  from  $\langle p_1 = Point\ x_1\ y_1 \rangle\ \langle p_1 \neq opp\ p_1 \rangle\ \langle y_1 \in carrier\ R \rangle$ 
  have  $y_1 \neq 0$ 
    by (simp add: opp-Point integral-iff equal-neg-zero)
  from Gen have  $x_1 = ((y_2 \ominus y_1) \otimes (x_2 \ominus x_1)) [\wedge] (2::nat) \ominus x_1 \ominus x_2$ 
    by (simp add: opp-Point)
  then have  $\langle 2 \rangle \otimes y_2 \otimes y_1 = a \otimes x_2 \oplus \langle 3 \rangle \otimes x_2 \otimes x_1 [\wedge] (2::nat) \oplus a \otimes x_1$ 
     $\ominus$ 
     $x_1 [\wedge] (3::nat) \oplus \langle 2 \rangle \otimes b$ 
    apply (field (prems) y_1 y_2)
    apply (field y_1 y_2)
    apply simp
    apply (insert  $\langle x_1 \neq x_2 \rangle\ \langle x_1 \in carrier\ R \rangle\ \langle x_2 \in carrier\ R \rangle$ )
    apply (simp add: eq-diff0)
  done

```

```

then have (x2 ⊖ (((«3» ⊗ x1 [⌈] (2::nat) ⊕ a) ⊙ («2» ⊗ (⊖ y1))) [⌈] (2::nat)
⊖
  («2» ⊗ x1)) ⊗ (x2 ⊖ x1) [⌈] (2::nat) = 0
apply (drule-tac f=λx. x [⌈] (2::nat) in arg-cong)
apply (field (prems) y1 y2)
apply (field y1 y2)
apply (insert ⟨y1 ≠ 0⟩ ⟨y1 ∈ carrier R⟩)
apply (simp-all add: integral-iff neg-equal-swap)
done
with a ⟨x1 ∈ carrier R⟩ ⟨y1 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩
  ⟨y1 ≠ 0⟩ ⟨x1 ≠ x2⟩
have x2 = (((«3» ⊗ x1 [⌈] (2::nat) ⊕ a) ⊙ («2» ⊗ (⊖ y1))) [⌈] (2::nat) ⊖
  «2» ⊗ x1
  by (simp add: integral-iff eq-diff0 neg-equal-swap)
with ⟨p2 = Point x2 y2⟩ - ⟨on-curve a b p2⟩
  add-closed [OF a b
    opp-closed [OF ⟨on-curve a b p1⟩] opp-closed [OF ⟨on-curve a b p1⟩]]
have p2 = add a (opp p1) (opp p1) ∨ p2 = opp (add a (opp p1) (opp p1))
apply (rule curve-elt-opp)
apply (insert ⟨y1 ∈ carrier R⟩ ⟨y1 ≠ 0⟩)
apply (simp add: add-def opp-Point neg-equal-zero Let-def ⟨p1 = Point x1
y1⟩)
done
then show ?case
proof
  assume p2 = opp (add a (opp p1) (opp p1))
  with a b ⟨on-curve a b p1⟩
  have p2 = add a p1 p1
    by (simp add: opp-add opp-opp opp-closed)
  show ?case
  proof (cases add a p1 p1 = opp p1)
    case True
    from a b ⟨on-curve a b p1⟩
    show ?thesis
    apply (simp add: opp-add [symmetric] ⟨p2 = add a p1 p1⟩ True)
    apply (simp add: ⟨p3 = add a p1 p2⟩ [simplified ⟨p3 = opp p1⟩])
    apply (simp add: ⟨p2 = add a p1 p1⟩ True add-opp)
    done
  next
  case False
  from a b ⟨on-curve a b p1⟩
  have add a p1 (opp p2) = opp (add a (add a p1 p1) (opp p1))
    by (simp add: ⟨p2 = add a p1 p1⟩
      opp-add add-closed opp-closed opp-opp add-comm)
  with a b ab ⟨on-curve a b p1⟩ ⟨p1 ≠ opp p1⟩ False
  have add a p1 (opp p2) = opp p1
    by (simp add: compat-add-triple)
  with ⟨p3 = add a p1 p2⟩ ⟨p3 = opp p1⟩
  have add a p1 p2 = add a p1 (opp p2) by simp

```

```

with a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩
have p₂ = opp p₂ using ⟨p₁ ≠ opp p₁⟩
  by (rule compat-add-opp)
with a b ⟨on-curve a b p₁⟩ ⟨p₂ = add a p₁ p₁⟩
show ?thesis by (simp add: opp-add)
qed
qed
qed
qed

```

lemma cancel:

```

assumes a: a ∈ carrier R
and b: b ∈ carrier R
and ab: nonsingular a b
and p₁: on-curve a b p₁
and p₂: on-curve a b p₂
and p₃: on-curve a b p₃
and eq: add a p₁ p₂ = add a p₁ p₃
shows p₂ = p₃
using a b p₁ p₂ p₁ p₂ eq
proof (induct rule: add-casew)
  case InfL
  then show ?case by (simp add: add-0-l)
next
  case (InfR p)
  with a b p₃ have add a p₃ p = p by (simp add: add-comm)
  with a b ab p₃ ⟨on-curve a b p⟩
  show ?case by (rule uniq-zero [symmetric])
next
  case (Opp p)
  from p₃ ⟨Infinity = add a p p₃⟩ [symmetric]
  show ?case by (rule uniq-opp [symmetric])
next
  case (Gen p₁ x₁ y₁ p₂ x₂ y₂ p₄ x₄ y₄ l)
  from ⟨on-curve a b p₁⟩ ⟨p₁ = Point x₁ y₁⟩
  have x₁ ∈ carrier R y₁ ∈ carrier R
    by (simp-all add: on-curve-def)
  from ⟨on-curve a b p₂⟩ ⟨p₂ = Point x₂ y₂⟩
  have x₂ ∈ carrier R y₂ ∈ carrier R
    by (simp-all add: on-curve-def)
  from add-closed [OF a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩]
  ⟨p₄ = add a p₁ p₂⟩ [symmetric] ⟨p₄ = Point x₄ y₄⟩
  have x₄ ∈ carrier R y₄ ∈ carrier R
    by (simp-all add: on-curve-def)
  from ⟨- ∨ -⟩ a ⟨p₁ ≠ opp p₂⟩ ⟨p₁ = Point x₁ y₁⟩ ⟨p₂ = Point x₂ y₂⟩
  ⟨x₁ ∈ carrier R⟩ ⟨y₁ ∈ carrier R⟩
  ⟨x₂ ∈ carrier R⟩ ⟨y₂ ∈ carrier R⟩
  have l ∈ carrier R
    by (auto simp add: opp-Point equal-neg-zero integral-iff eq-diff0)

```

```

from  $a\ b\ \langle \text{on-curve } a\ b\ p_1 \rangle\ p_3\ \langle \text{on-curve } a\ b\ p_1 \rangle\ p_3\ \langle p_1 = \text{Point } x_1\ y_1 \rangle$ 
   $\langle p_4 = \text{add } a\ p_1\ p_2 \rangle\ \langle p_4 = \text{add } a\ p_1\ p_3 \rangle\ \langle p_1 \neq \text{opp } p_2 \rangle$ 
show ?case
proof (induct rule: add-casew)
  case InfL
  then show ?case by (simp add: add-0-l)
next
  case (InfR  $p$ )
  with  $a\ b\ \langle \text{on-curve } a\ b\ p_2 \rangle$ 
  have  $\text{add } a\ p_2\ p = p$  by (simp add: add-comm)
  with  $a\ b\ ab\ \langle \text{on-curve } a\ b\ p_2 \rangle\ \langle \text{on-curve } a\ b\ p \rangle$ 
  show ?case by (rule uniq-zero)
next
  case (Opp  $p$ )
  then have  $\text{add } a\ p\ p_2 = \text{Infinity}$  by simp
  with  $\langle \text{on-curve } a\ b\ p_2 \rangle$  show ?case by (rule uniq-opp)
next
  case (Gen  $p_1\ x_1'\ y_1'\ p_3\ x_3\ y_3\ p_5\ x_5\ y_5\ l'$ )
  from  $\langle \text{on-curve } a\ b\ p_3 \rangle\ \langle p_3 = \text{Point } x_3\ y_3 \rangle$ 
  have  $x_3 \in \text{carrier } R\ y_3 \in \text{carrier } R$ 
  by (simp-all add: on-curve-def)
  from  $\langle x_1' = x_3 \wedge - \vee \rightarrow a \rangle\ \langle p_1 \neq \text{opp } p_3 \rangle$ 
   $\langle p_1 = \text{Point } x_1\ y_1 \rangle\ \langle p_1 = \text{Point } x_1'\ y_1' \rangle\ \langle p_3 = \text{Point } x_3\ y_3 \rangle$ 
   $\langle x_1 \in \text{carrier } R \rangle\ \langle y_1 \in \text{carrier } R \rangle$ 
   $\langle x_3 \in \text{carrier } R \rangle\ \langle y_3 \in \text{carrier } R \rangle$ 
  have  $l' \in \text{carrier } R$ 
  by (auto simp add: opp-Point equal-neg-zero integral-iff eq-diff0)
  from  $\langle p_4 = p_5 \rangle\ \langle p_4 = \text{Point } x_4\ y_4 \rangle\ \langle p_5 = \text{Point } x_5\ y_5 \rangle$ 
   $\langle p_1 = \text{Point } x_1'\ y_1' \rangle\ \langle p_1 = \text{Point } x_1\ y_1 \rangle$ 
   $\langle y_4 = \ominus y_1 \ominus l \otimes (x_4 \ominus x_1) \rangle\ \langle y_5 = \ominus y_1' \ominus l' \otimes (x_5 \ominus x_1') \rangle$ 
   $\langle x_1 \in \text{carrier } R \rangle\ \langle y_1 \in \text{carrier } R \rangle\ \langle x_4 \in \text{carrier } R \rangle\ \langle l' \in \text{carrier } R \rangle$ 
  have  $\mathbf{0} = \ominus y_1 \ominus l \otimes (x_4 \ominus x_1) \ominus (\ominus y_1 \ominus l' \otimes (x_4 \ominus x_1))$ 
  by (auto simp add: trans [OF eq-commute eq-diff0])
  with  $\langle x_1 \in \text{carrier } R \rangle\ \langle y_1 \in \text{carrier } R \rangle\ \langle x_4 \in \text{carrier } R \rangle$ 
   $\langle l \in \text{carrier } R \rangle\ \langle l' \in \text{carrier } R \rangle$ 
  have  $(l' \ominus l) \otimes (x_4 \ominus x_1) = \mathbf{0}$ 
  apply simp
  apply (rule eq-diff0 [THEN iffD1])
  apply simp
  apply simp
  apply ring
  done
  with  $\langle x_1 \in \text{carrier } R \rangle\ \langle x_4 \in \text{carrier } R \rangle\ \langle l \in \text{carrier } R \rangle\ \langle l' \in \text{carrier } R \rangle$ 
  have  $l' = l \vee x_4 = x_1$ 
  by (simp add: integral-iff eq-diff0)
  then show ?case
proof
  assume  $l' = l$ 
  with  $\langle p_4 = p_5 \rangle\ \langle p_4 = \text{Point } x_4\ y_4 \rangle\ \langle p_5 = \text{Point } x_5\ y_5 \rangle$ 

```



```

    ⟨p1 = Point x1' y1'⟩ ⟨p1 = Point x1 y1⟩
    ⟨x4 = l [∧] 2 ⊖ x1 ⊖ x2⟩ ⟨x5 = l' [∧] 2 ⊖ x1' ⊖ x3⟩
    ⟨x1 ∈ carrier R⟩ ⟨x3 ∈ carrier R⟩ ⟨l ∈ carrier R⟩
  have 0 = l [∧] (2::nat) ⊖ x1 ⊖ x2 ⊖ (l [∧] (2::nat) ⊖ x1 ⊖ x3)
    by (simp add: trans [OF eq-commute eq-diff0])
  with ⟨x1 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩ ⟨x3 ∈ carrier R⟩ ⟨l ∈ carrier R⟩
  have x2 = x3
    apply (rule-tac eq-diff0 [THEN iffD1, THEN sym])
    apply simp-all
    apply (rule eq-diff0 [THEN iffD1])
    apply simp-all[2]
    apply ring
    done
  with ⟨p2 = Point x2 y2⟩ ⟨p3 = Point x3 y3⟩ ⟨on-curve a b p2⟩ ⟨on-curve a b
p3⟩
  have p2 = p3 ∨ p2 = opp p3 by (rule curve-elt-opp)
  then show ?case
  proof
    assume p2 = opp p3
    with ⟨on-curve a b p3⟩ have opp p2 = p3
      by (simp add: opp-opp)
    with ⟨p4 = p5⟩ ⟨p4 = add a p1 p2⟩ ⟨p5 = add a p1 p3⟩
    have add a p1 p2 = add a p1 (opp p2) by simp
    show ?case
    proof (cases p1 = opp p1)
      case True
        with ⟨p1 ≠ opp p2⟩ ⟨p1 ≠ opp p3⟩
        have p1 ≠ p2 p1 ≠ p3 by auto
        with ⟨l' = l⟩ ⟨x1 = x2 ∧ -∨ -⟩ ⟨x1' = x3 ∧ -∨ -⟩
        ⟨p1 = Point x1 y1⟩ ⟨p1 = Point x1' y1'⟩
        ⟨p2 = Point x2 y2⟩ ⟨p3 = Point x3 y3⟩
        ⟨p2 = opp p3⟩
        have eq: (y2 ⊖ y1) ⊙ (x2 ⊖ x1) = (y3 ⊖ y1) ⊙ (x2 ⊖ x1) and x1 ≠ x2
          by (auto simp add: opp-Point)
        from eq have y2 = y3
          apply (field (prems))
          apply (rule eq-diff0 [THEN iffD1])
          apply (insert ⟨x1 ≠ x2⟩ ⟨x1 ∈ carrier R⟩ ⟨y1 ∈ carrier R⟩
            ⟨x2 ∈ carrier R⟩ ⟨y2 ∈ carrier R⟩ ⟨y3 ∈ carrier R⟩)
          apply simp-all
          apply (erule subst)
          apply (rule eq-diff0 [THEN iffD1])
          apply simp-all
          apply ring
          apply (simp add: eq-diff0)
          done
        with ⟨p2 = opp p3⟩ ⟨p2 = Point x2 y2⟩ ⟨p3 = Point x3 y3⟩
        show ?thesis by (simp add: opp-Point)
      next

```

```

    case False
    with a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩
      ⟨add a p₁ p₂ = add a p₁ (opp p₂)⟩
    have p₂ = opp p₂ by (rule compat-add-opp)
    with ⟨opp p₂ = p₃⟩ show ?thesis by simp
  qed
qed
next
assume x₄ = x₁
with ⟨p₄ = Point x₄ y₄⟩ [simplified ⟨p₄ = add a p₁ p₂⟩]
  ⟨p₁ = Point x₁ y₁⟩
  add-closed [OF a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩]
  ⟨on-curve a b p₁⟩
have add a p₁ p₂ = p₁ ∨ add a p₁ p₂ = opp p₁ by (rule curve-elt-opp)
then show ?case
proof
  assume add a p₁ p₂ = p₁
  with a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩
  have add a p₂ p₁ = p₁ by (simp add: add-comm)
  with a b ab ⟨on-curve a b p₂⟩ ⟨on-curve a b p₁⟩
  have p₂ = Infinity by (rule uniq-zero)
  moreover from ⟨add a p₁ p₂ = p₁⟩
    ⟨p₄ = p₅⟩ ⟨p₄ = add a p₁ p₂⟩ ⟨p₅ = add a p₁ p₃⟩
    a b ⟨on-curve a b p₁⟩ ⟨on-curve a b p₃⟩
  have add a p₃ p₁ = p₁ by (simp add: add-comm)
  with a b ab ⟨on-curve a b p₃⟩ ⟨on-curve a b p₁⟩
  have p₃ = Infinity by (rule uniq-zero)
  ultimately show ?case by simp
next
assume add a p₁ p₂ = opp p₁
with a b ab ⟨on-curve a b p₁⟩ ⟨on-curve a b p₂⟩
have p₂ = add a (opp p₁) (opp p₁) by (rule add-opp-double-opp)
moreover from ⟨add a p₁ p₂ = opp p₁⟩
  ⟨p₄ = p₅⟩ ⟨p₄ = add a p₁ p₂⟩ ⟨p₅ = add a p₁ p₃⟩
have add a p₁ p₃ = opp p₁ by simp
with a b ab ⟨on-curve a b p₁⟩ ⟨on-curve a b p₃⟩
have p₃ = add a (opp p₁) (opp p₁) by (rule add-opp-double-opp)
ultimately show ?case by simp
qed
qed
qed
qed

```

lemma *add-minus-id*:  
 assumes *a*:  $a \in \text{carrier } R$   
 and *b*:  $b \in \text{carrier } R$   
 and *ab*: *nonsingular a b*  
 and *p₁*: *on-curve a b p₁*  
 and *p₂*: *on-curve a b p₂*

**shows**  $\text{add } a \ (\text{add } a \ p_1 \ p_2) \ (\text{opp } p_2) = p_1$   
**proof** (*cases*  $\text{add } a \ p_1 \ p_2 = \text{opp } p_2$ )  
    **case** *True*  
    **then have**  $\text{add } a \ (\text{add } a \ p_1 \ p_2) \ (\text{opp } p_2) = \text{add } a \ (\text{opp } p_2) \ (\text{opp } p_2)$   
        **by** *simp*  
    **also from**  $a \ b \ p_1 \ p_2 \ \text{True}$  **have**  $\text{add } a \ p_2 \ p_1 = \text{opp } p_2$   
        **by** (*simp* *add: add-comm*)  
    **with**  $a \ b \ ab \ p_2 \ p_1$  **have**  $\text{add } a \ (\text{opp } p_2) \ (\text{opp } p_2) = p_1$   
        **by** (*rule* *add-opp-double-opp* [*symmetric*])  
    **finally show** *?thesis* .  
**next**  
    **case** *False*  
    **from**  $a \ b \ p_1 \ p_2 \ p_1 \ p_2 \ \text{False}$  **show** *?thesis*  
    **proof** (*induct* *rule: add-case*)  
        **case** *InfL*  
        **then show** *?case* **by** (*simp* *add: add-opp*)  
    **next**  
        **case** *InfR*  
        **show** *?case* **by** (*simp* *add: add-0-r*)  
    **next**  
        **case** *Opp*  
        **then show** *?case* **by** (*simp* *add: opp-opp add-0-l*)  
    **next**  
        **case** (*Tan*  $p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ l$ )  
        **note**  $a \ b \ ab \ \langle \text{on-curve } a \ b \ p_1 \rangle$   
        **moreover from**  $\langle \text{on-curve } a \ b \ p_1 \rangle \ \langle p_1 = \text{Point } x_1 \ y_1 \rangle$   
        **have**  $y_1 \in \text{carrier } R$  **by** (*simp* *add: on-curve-def*)  
        **with**  $\langle y_1 \neq 0 \rangle \ \langle p_1 = \text{Point } x_1 \ y_1 \rangle$   
        **have**  $p_1 \neq \text{opp } p_1$  **by** (*simp* *add: opp-Point equal-neg-zero*)  
        **moreover from**  $\langle p_2 = \text{add } a \ p_1 \ p_1 \rangle \ \langle p_2 \neq \text{opp } p_1 \rangle$   
        **have**  $\text{add } a \ p_1 \ p_1 \neq \text{opp } p_1$  **by** *simp*  
        **ultimately have**  $\text{add } a \ (\text{add } a \ p_1 \ p_1) \ (\text{opp } p_1) = p_1$   
            **by** (*rule* *compat-add-triple*)  
        **with**  $\langle p_2 = \text{add } a \ p_1 \ p_1 \rangle$  **show** *?case* **by** *simp*  
    **next**  
        **case** (*Gen*  $p_1 \ x_1 \ y_1 \ p_2 \ x_2 \ y_2 \ p_3 \ x_3 \ y_3 \ l$ )  
        **from**  $\langle p_3 = \text{add } a \ p_1 \ p_2 \rangle \ \langle \text{on-curve } a \ b \ p_2 \rangle$   
        **have**  $p_3 = \text{add } a \ p_1 \ (\text{opp } (\text{opp } p_2))$  **by** (*simp* *add: opp-opp*)  
        **with**  $a \ b$   
            *add-closed* [*OF*  $a \ b \ \langle \text{on-curve } a \ b \ p_1 \rangle \ \langle \text{on-curve } a \ b \ p_2 \rangle$ ,  
                *folded*  $\langle p_3 = \text{add } a \ p_1 \ p_2 \rangle$ ]  
            *opp-closed* [*OF*  $\langle \text{on-curve } a \ b \ p_2 \rangle$ ]  
            *opp-closed* [*OF*  $\langle \text{on-curve } a \ b \ p_2 \rangle$ ]  
            *opp-opp* [*OF*  $\langle \text{on-curve } a \ b \ p_2 \rangle$ ]  
            *Gen*  
        **show** *?case*  
        **proof** (*induct* *rule: add-case*)  
            **case** *InfL*  
            **then show** *?case* **by** *simp*

```

next
  case InfR
  then show ?case by (simp add: add-0-r)
next
  case (Opp p)
  from ⟨on-curve a b p⟩ ⟨p = add a p1 (opp (opp p))⟩
  have add a p1 p = p by (simp add: opp-opp)
  with a b ab ⟨on-curve a b p1⟩ ⟨on-curve a b p⟩
  show ?case by (rule uniq-zero [symmetric])
next
  case Tan
  then show ?case by simp
next
  case (Gen p4 x4 y4 p5 x5 y5 p6 x6 y6 l')
  from ⟨on-curve a b p1⟩ ⟨p1 = Point x1 y1⟩
  have x1 ∈ carrier R y1 ∈ carrier R
    by (simp-all add: on-curve-def)
  from ⟨on-curve a b p2⟩ ⟨p2 = Point x2 y2⟩
  have x2 ∈ carrier R y2 ∈ carrier R
    by (simp-all add: on-curve-def)
  from ⟨on-curve a b p5⟩ ⟨opp p5 = p2⟩
    ⟨p2 = Point x2 y2⟩ ⟨p5 = Point x5 y5⟩
  have y5 = ⊖ y2 x5 = x2
    by (auto simp add: opp-Point on-curve-def)
  from ⟨p4 = Point x3 y3⟩ ⟨p4 = Point x4 y4⟩
  have x4 = x3 y4 = y3 by simp-all
  from ⟨x4 ≠ x5⟩ show ?case
    apply (simp add:
      ⟨y5 = ⊖ y2⟩ ⟨x5 = x2⟩
      ⟨x4 = x3⟩ ⟨y4 = y3⟩
      ⟨p6 = Point x6 y6⟩ ⟨p1 = Point x1 y1⟩
      ⟨x6 = l' [∧] 2 ⊖ x4 ⊖ x5⟩ ⟨y6 = ⊖ y4 ⊖ l' ⊗ (x6 ⊖ x4)⟩
      ⟨l' = (y5 ⊖ y4) ⊗ (x5 ⊖ x4)⟩
      ⟨x3 = l [∧] 2 ⊖ x1 ⊖ x2⟩ ⟨y3 = ⊖ y1 ⊖ l ⊗ (x3 ⊖ x1)⟩
      ⟨l = (y2 ⊖ y1) ⊗ (x2 ⊖ x1)⟩)
    apply (rule conjI)
    apply field
    apply (rule conjI)
    apply (rule notI)
    apply (erule notE)
    apply (ring (prems))
    apply (rule sym)
    apply field
    apply (simp-all add: eq-diff0 [OF ⟨x2 ∈ carrier R⟩ ⟨x1 ∈ carrier R⟩]
      ⟨x1 ≠ x2⟩ [THEN not-sym])
    apply field
    apply (rule conjI)
    apply (simp add: eq-diff0 [OF ⟨x2 ∈ carrier R⟩ ⟨x1 ∈ carrier R⟩]
      ⟨x1 ≠ x2⟩ [THEN not-sym])

```

```

    apply (rule notI)
    apply (erule notE)
    apply (ring (prems))
    apply (rule sym)
    apply field
    apply (simp add: eq-diff0 [OF ⟨x2 ∈ carrier R⟩ ⟨x1 ∈ carrier R⟩]
      ⟨x1 ≠ x2⟩ [THEN not-sym])
  done
qed
qed
qed

```

**lemma** *add-shift-minus*:  
**assumes** *a*:  $a \in \text{carrier } R$   
**and** *b*:  $b \in \text{carrier } R$   
**and** *ab*: *nonsingular* *a b*  
**and** *p*<sub>1</sub>: *on-curve* *a b p*<sub>1</sub>  
**and** *p*<sub>2</sub>: *on-curve* *a b p*<sub>2</sub>  
**and** *p*<sub>3</sub>: *on-curve* *a b p*<sub>3</sub>  
**and** *eq*:  $\text{add } a \ p_1 \ p_2 = p_3$   
**shows**  $p_1 = \text{add } a \ p_3 \ (\text{opp } p_2)$

**proof** –  
**note** *eq*  
**also from** *add-minus-id* [OF *a b ab p*<sub>3</sub> *opp-closed* [OF *p*<sub>2</sub>]] *p*<sub>2</sub>  
**have**  $p_3 = \text{add } a \ (\text{add } a \ p_3 \ (\text{opp } p_2)) \ p_2$  **by** (*simp add: opp-opp*)  
**finally have**  $\text{add } a \ p_2 \ p_1 = \text{add } a \ p_2 \ (\text{add } a \ p_3 \ (\text{opp } p_2))$   
**using** *a b p*<sub>1</sub> *p*<sub>2</sub> *p*<sub>3</sub>  
**by** (*simp add: add-comm add-closed opp-closed*)  
**with** *a b ab p*<sub>2</sub> *p*<sub>1</sub> *add-closed* [OF *a b p*<sub>3</sub> *opp-closed* [OF *p*<sub>2</sub>]]  
**show** *?thesis* **by** (*rule cancel*)  
**qed**

**lemma** *degen-assoc*:  
**assumes** *a*:  $a \in \text{carrier } R$   
**and** *b*:  $b \in \text{carrier } R$   
**and** *ab*: *nonsingular* *a b*  
**and** *p*<sub>1</sub>: *on-curve* *a b p*<sub>1</sub>  
**and** *p*<sub>2</sub>: *on-curve* *a b p*<sub>2</sub>  
**and** *p*<sub>3</sub>: *on-curve* *a b p*<sub>3</sub>  
**and** *H*:  
 $(p_1 = \text{Infinity} \vee p_2 = \text{Infinity} \vee p_3 = \text{Infinity}) \vee$   
 $(p_1 = \text{opp } p_2 \vee p_2 = \text{opp } p_3) \vee$   
 $(\text{opp } p_1 = \text{add } a \ p_2 \ p_3 \vee \text{opp } p_3 = \text{add } a \ p_1 \ p_2)$   
**shows**  $\text{add } a \ p_1 \ (\text{add } a \ p_2 \ p_3) = \text{add } a \ (\text{add } a \ p_1 \ p_2) \ p_3$   
**using** *H*  
**proof** (*elim disjE*)  
**assume**  $p_1 = \text{Infinity}$   
**then show** *?thesis* **by** (*simp add: add-0-l*)  
**next**

```

    assume  $p_2 = \text{Infinity}$ 
    then show ?thesis by (simp add: add-0-l add-0-r)
next
    assume  $p_3 = \text{Infinity}$ 
    then show ?thesis by (simp add: add-0-r)
next
    assume  $p_1 = \text{opp } p_2$ 
    from  $a \ b \ p_2 \ p_3$ 
    have  $\text{add } a \ (\text{opp } p_2) \ (\text{add } a \ p_2 \ p_3) = \text{add } a \ (\text{add } a \ p_3 \ p_2) \ (\text{opp } p_2)$ 
      by (simp add: add-comm add-closed opp-closed)
    also from  $a \ b \ ab \ p_3 \ p_2$  have  $\dots = p_3$  by (rule add-minus-id)
    also have  $\dots = \text{add } a \ \text{Infinity} \ p_3$  by (simp add: add-0-l)
    also from  $p_2$  have  $\dots = \text{add } a \ (\text{add } a \ p_2 \ (\text{opp } p_2)) \ p_3$ 
      by (simp add: add-opp)
    also from  $a \ b \ p_2$  have  $\dots = \text{add } a \ (\text{add } a \ (\text{opp } p_2) \ p_2) \ p_3$ 
      by (simp add: add-comm opp-closed)
    finally show ?thesis using  $\langle p_1 = \text{opp } p_2 \rangle$  by simp
next
    assume  $p_2 = \text{opp } p_3$ 
    from  $a \ b \ p_3$ 
    have  $\text{add } a \ p_1 \ (\text{add } a \ (\text{opp } p_3) \ p_3) = \text{add } a \ p_1 \ (\text{add } a \ p_3 \ (\text{opp } p_3))$ 
      by (simp add: add-comm opp-closed)
    also from  $a \ b \ ab \ p_1 \ p_3$ 
    have  $\dots = \text{add } a \ (\text{add } a \ p_1 \ (\text{opp } p_3)) \ (\text{opp } (\text{opp } p_3))$ 
      by (simp add: add-opp add-minus-id add-0-r opp-closed)
    finally show ?thesis using  $p_3 \ \langle p_2 = \text{opp } p_3 \rangle$ 
      by (simp add: opp-opp)
next
    assume  $eq: \text{opp } p_1 = \text{add } a \ p_2 \ p_3$ 
    from  $eq$  [symmetric]  $p_1$ 
    have  $\text{add } a \ p_1 \ (\text{add } a \ p_2 \ p_3) = \text{Infinity}$  by (simp add: add-opp)
    also from  $p_3$  have  $\dots = \text{add } a \ p_3 \ (\text{opp } p_3)$  by (simp add: add-opp)
    also from  $a \ b \ p_3$  have  $\dots = \text{add } a \ (\text{opp } p_3) \ p_3$ 
      by (simp add: add-comm opp-closed)
    also from  $a \ b \ ab \ p_2 \ p_3$ 
    have  $\dots = \text{add } a \ (\text{add } a \ (\text{add } a \ (\text{opp } p_3) \ (\text{opp } p_2)) \ (\text{opp } (\text{opp } p_2))) \ p_3$ 
      by (simp add: add-minus-id opp-closed)
    also from  $a \ b \ p_2 \ p_3$ 
    have  $\dots = \text{add } a \ (\text{add } a \ (\text{add } a \ (\text{opp } p_2) \ (\text{opp } p_3)) \ p_2) \ p_3$ 
      by (simp add: add-comm opp-opp opp-closed)
    finally show ?thesis
      using opp-add [OF  $a \ b \ p_2 \ p_3$ ]  $eq$  [symmetric]  $p_1$ 
      by (simp add: opp-opp)
next
    assume  $eq: \text{opp } p_3 = \text{add } a \ p_1 \ p_2$ 
    from opp-add [OF  $a \ b \ p_1 \ p_2$ ]  $eq$  [symmetric]  $p_3$ 
    have  $\text{add } a \ p_1 \ (\text{add } a \ p_2 \ p_3) = \text{add } a \ p_1 \ (\text{add } a \ p_2 \ (\text{add } a \ (\text{opp } p_1) \ (\text{opp } p_2)))$ 
      by (simp add: opp-opp)
    also from  $a \ b \ p_1 \ p_2$ 

```

```

have ... = add a p1 (add a (add a (opp p1) (opp p2)) (opp (opp p2)))
  by (simp add: add-comm opp-opp add-closed opp-closed)
also from a b ab p1 p2 have ... = Infinity
  by (simp add: add-minus-id add-opp opp-closed)
also from p3 have ... = add a p3 (opp p3) by (simp add: add-opp)
also from a b p3 have ... = add a (opp p3) p3
  by (simp add: add-comm opp-closed)
finally show ?thesis using eq [symmetric] by simp
qed

```

**lemma** *spec<sub>4</sub>-assoc*:

```

assumes a: a ∈ carrier R
and b: b ∈ carrier R
and ab: nonsingular a b
and p1: on-curve a b p1
and p2: on-curve a b p2
shows add a p1 (add a p2 p2) = add a (add a p1 p2) p2
proof (cases p1 = Infinity)
  case True
    from a b ab p1 p2 p2
    show ?thesis by (rule degen-assoc) (simp add: True)
  next
    case False
    show ?thesis
    proof (cases p2 = Infinity)
      case True
        from a b ab p1 p2 p2
        show ?thesis by (rule degen-assoc) (simp add: True)
      next
        case False
        show ?thesis
        proof (cases p2 = opp p2)
          case True
            from a b ab p1 p2 p2
            show ?thesis by (rule degen-assoc) (simp add: True [symmetric])
          next
            case False
            show ?thesis
            proof (cases p1 = opp p2)
              case True
                from a b ab p1 p2 p2
                show ?thesis by (rule degen-assoc) (simp add: True)
              next
                case False
                show ?thesis
                proof (cases opp p1 = add a p2 p2)
                  case True
                    from a b ab p1 p2 p2
                    show ?thesis by (rule degen-assoc) (simp add: True)
                  next
                    case False
                    show ?thesis
                    proof (cases opp p1 = add a p2 p2)
                      case True
                        from a b ab p1 p2 p2
                        show ?thesis by (rule degen-assoc) (simp add: True)
                      next
                        case False
                        show ?thesis
                        proof (cases opp p1 = add a p2 p2)
                          case True
                            from a b ab p1 p2 p2
                            show ?thesis by (rule degen-assoc) (simp add: True)
                          next
                            case False
                            show ?thesis
                            proof (cases opp p1 = add a p2 p2)
                              case True
                                from a b ab p1 p2 p2
                                show ?thesis by (rule degen-assoc) (simp add: True)
                              next
                                case False
                                show ?thesis
                                proof (cases opp p1 = add a p2 p2)
                                  case True
                                    from a b ab p1 p2 p2
                                    show ?thesis by (rule degen-assoc) (simp add: True)
                                  next
                                    case False
                                    show ?thesis
                                    proof (cases opp p1 = add a p2 p2)
                                      case True
                                        from a b ab p1 p2 p2
                                        show ?thesis by (rule degen-assoc) (simp add: True)
                                      next
                                        case False
                                        show ?thesis
                                        proof (cases opp p1 = add a p2 p2)
                                          case True
                                            from a b ab p1 p2 p2
                                            show ?thesis by (rule degen-assoc) (simp add: True)
                                          next
                                            case False
                                            show ?thesis
                                            proof (cases opp p1 = add a p2 p2)
                                              case True
                                                from a b ab p1 p2 p2
                                                show ?thesis by (rule degen-assoc) (simp add: True)
                                              next
                                                case False
                                                show ?thesis
                                                proof (cases opp p1 = add a p2 p2)
                                                  case True
                                                    from a b ab p1 p2 p2
                                                    show ?thesis by (rule degen-assoc) (simp add: True)
                                                  next
                                                    case False
                                                    show ?thesis
                                                    proof (cases opp p1 = add a p2 p2)
                                                      case True
                                                        from a b ab p1 p2 p2
                                                        show ?thesis by (rule degen-assoc) (simp add: True)
                                                      next
                                                        case False
                                                        show ?thesis

```

```

next
  case False
  show ?thesis
  proof (cases opp p2 = add a p1 p2)
    case True
    from a b ab p1 p2 p2
    show ?thesis by (rule degen-assoc) (simp add: True)
  next
  case False
  show ?thesis
  proof (cases p1 = add a p2 p2)
    case True
    from a b p1 p2 ⟨p1 ≠ opp p2⟩ ⟨p2 ≠ opp p2⟩
      ⟨opp p1 ≠ add a p2 p2⟩ ⟨opp p2 ≠ add a p1 p2⟩
      ⟨p1 ≠ Infinity⟩ ⟨p2 ≠ Infinity⟩
    show ?thesis
    apply (simp add: True)
    apply (rule spec3-assoc [OF a b])
    apply (simp-all add: is-generic-def is-tangent-def)
    apply (rule notI)
    apply (drule uniq-zero [OF a b ab p2 p2])
    apply simp
    apply (intro conjI notI)
    apply (erule notE)
    apply (rule uniq-opp [of a b])
    apply (simp-all add: add-comm add-closed)[2]
    apply (erule notE)
    apply (drule uniq-zero [OF a b ab add-closed [OF a b p2 p2] p2])
    apply simp
    done
  next
  case False
  show ?thesis
  proof (cases p2 = add a p1 p2)
    case True
    from a b ab p1 p2 True [symmetric]
    have p1 = Infinity by (rule uniq-zero)
    then show ?thesis by (simp add: add-0-l)
  next
  case False
  show ?thesis
  proof (cases p1 = p2)
    case True
    with a b p2 show ?thesis
    by (simp add: add-comm add-closed)
  next
  case False
  with a b p1 p2 ⟨p1 ≠ Infinity⟩ ⟨p2 ≠ Infinity⟩
    ⟨p1 ≠ opp p2⟩ ⟨p2 ≠ opp p2⟩

```



```

    ⟨p1 ≠ add a p2 p2⟩ ⟨p2 ≠ add a p1 p2⟩ ⟨opp p2 ≠ add a p1 p2⟩
  show ?thesis
    apply (rule-tac spec2-assoc [OF a b])
    apply (simp-all add: is-generic-def is-tangent-def)
    apply (rule notI)
    apply (erule notE [of p1 = opp p2])
    apply (rule uniq-opp)
    apply assumption
    apply (simp add: add-comm)
    apply (intro conjI notI)
    apply (erule notE [of p2 = opp p2])
    apply (rule uniq-opp)
    apply assumption+
    apply (rule notE [OF ⟨opp p1 ≠ add a p2 p2⟩])
    apply (simp add: opp-opp [OF add-closed [OF a b p2 p2]])
  done
qed
qed
qed
qed
qed
qed
qed
qed
qed

```

```

lemma add-assoc:
  assumes a: a ∈ carrier R
  and b: b ∈ carrier R
  and ab: nonsingular a b
  and p1: on-curve a b p1
  and p2: on-curve a b p2
  and p3: on-curve a b p3
  shows add a p1 (add a p2 p3) = add a (add a p1 p2) p3
proof (cases p1 = Infinity)
  case True
  from a b ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
  case False
  show ?thesis
  proof (cases p2 = Infinity)
    case True
    from a b ab p1 p2 p3
    show ?thesis by (rule degen-assoc) (simp add: True)
  next
    case False
    show ?thesis
    proof (cases p3 = Infinity)

```

```

case True
from a b ab p1 p2 p3
show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p1 = p2)
  case True
  from a b p2 p3
  have add a p2 (add a p2 p3) = add a (add a p3 p2) p2
    by (simp add: add-comm add-closed)
  also from a b ab p3 p2 have ... = add a p3 (add a p2 p2)
    by (simp add: spec4-assoc)
  also from a b p2 p3
  have ... = add a (add a p2 p2) p3
    by (simp add: add-comm add-closed)
  finally show ?thesis using True by simp
next
case False
show ?thesis
proof (cases p1 = opp p2)
  case True
  from a b ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p2 = p3)
  case True
  with a b ab p1 p3 show ?thesis
    by (simp add: spec4-assoc)
next
case False
show ?thesis
proof (cases p2 = opp p3)
  case True
  from a b ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p1 = add a p2 p3)
  case True
  from a b ab p1 p2 p3
  show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases opp p3 = add a p1 p2)

```

```

case True
from a b ab p1 p2 p3
show ?thesis by (rule degen-assoc) (simp add: True)
next
case False
show ?thesis
proof (cases p1 = add a p2 p3)
  case True
  with a b ab p2 p3 show ?thesis
  apply simp
  apply (rule cancel [OF a b ab opp-closed [OF p3]])
  apply (simp-all add: add-closed)
  apply (simp add: spec4-assoc add-closed opp-closed)
  apply (simp add: add-comm [of a b opp p3]
    add-closed opp-closed add-minus-id)
  apply (simp add: add-comm add-closed)
  done
next
case False
show ?thesis
proof (cases p3 = add a p1 p2)
  case True
  with a b ab p1 p2 show ?thesis
  apply simp
  apply (rule cancel [OF a b ab opp-closed [OF p1]])
  apply (simp-all add: add-closed)
  apply (simp add: spec4-assoc add-closed opp-closed)
  apply (simp add: add-comm [of a b opp p1] add-comm [of a b
    add-closed opp-closed add-minus-id])
  done
next
case False
with a b p1 p2 p3
  <p1 ≠ Infinity> <p2 ≠ Infinity> <p3 ≠ Infinity>
  <p1 ≠ p2> <p1 ≠ opp p2> <p2 ≠ p3> <p2 ≠ opp p3>
  <opp p3 ≠ add a p1 p2> <p1 ≠ add a p2 p3>
show ?thesis
  apply (rule-tac spec1-assoc [of a b])
  apply (simp-all add: is-generic-def)
  apply (rule notI)
  apply (erule notE [of p1 = opp p2])
  apply (rule uniq-opp)
  apply assumption
  apply (simp add: add-comm)
  apply (intro conjI notI)
  apply (erule notE [of p2 = opp p3])
  apply (rule uniq-opp)
  apply assumption

```

```

    apply (simp add: add-comm)
    apply (rule notE [OF ‹opp p1 ≠ add a p2 p3›])
    apply (simp add: opp-opp [OF add-closed [OF a b p2 p3]])
  done
qed
qed
qed
qed
qed
qed
qed
qed
qed
qed
qed

```

**lemma** *add-comm'*:

```

a ∈ carrier R ⇒ b ∈ carrier R ⇒ nonsingular a b ⇒
on-curve a b p1 ⇒ on-curve a b p2 ⇒ on-curve a b p3 ⇒
add a p2 (add a p1 p3) = add a p1 (add a p2 p3)
by (simp add: add-assoc add-comm)

```

**primrec** *point-mult* :: 'a ⇒ nat ⇒ 'a point ⇒ 'a point  
**where**

```

point-mult a 0 p = Infinity
| point-mult a (Suc n) p = add a p (point-mult a n p)

```

**lemma** *point-mult-closed*: a ∈ carrier R ⇒ b ∈ carrier R ⇒  
on-curve a b p ⇒ on-curve a b (point-mult a n p)  
**by** (induct n) (simp-all add: add-closed)

**lemma** *point-mult-add*:

```

a ∈ carrier R ⇒ b ∈ carrier R ⇒ on-curve a b p ⇒ nonsingular a b ⇒
point-mult a (m + n) p = add a (point-mult a m p) (point-mult a n p)
by (induct m) (simp-all add: add-assoc point-mult-closed add-0-l)

```

**lemma** *point-mult-mult*:

```

a ∈ carrier R ⇒ b ∈ carrier R ⇒ on-curve a b p ⇒ nonsingular a b ⇒
point-mult a (m * n) p = point-mult a n (point-mult a m p)
by (induct n) (simp-all add: point-mult-add)

```

**lemma** *point-mult2-eq-double*:

```

point-mult a 2 p = add a p p
by (simp add: numeral-2-eq-2 add-0-r)

```

**end**

### 3.2 Projective Coordinates

**type-synonym**  $'a$  *ppoint* =  $'a \times 'a \times 'a$

**definition** (in *cring*) *pdouble* ::  $'a \Rightarrow 'a$  *ppoint*  $\Rightarrow 'a$  *ppoint* **where**

*pdouble*  $a$   $p$  =  
 (let  $(x, y, z) = p$   
 in  
 if  $z = \mathbf{0}$  then  $p$   
 else  
 let  
 $l = \langle 2 \rangle \otimes y \otimes z$ ;  
 $m = \langle 3 \rangle \otimes x \ [\uparrow] (2::nat) \oplus a \otimes z \ [\uparrow] (2::nat)$   
 in  
 $(l \otimes (m \ [\uparrow] (2::nat) \ominus \langle 4 \rangle \otimes x \otimes y \otimes l),$   
 $m \otimes (\langle 6 \rangle \otimes x \otimes y \otimes l \ominus m \ [\uparrow] (2::nat)) \ominus$   
 $\langle 2 \rangle \otimes y \ [\uparrow] (2::nat) \otimes l \ [\uparrow] (2::nat),$   
 $l \ [\uparrow] (3::nat)))$

**definition** (in *cring*) *padd* ::  $'a \Rightarrow 'a$  *ppoint*  $\Rightarrow 'a$  *ppoint*  $\Rightarrow 'a$  *ppoint* **where**

*padd*  $a$   $p_1$   $p_2$  =  
 (let  
 $(x_1, y_1, z_1) = p_1$ ;  
 $(x_2, y_2, z_2) = p_2$   
 in  
 if  $z_1 = \mathbf{0}$  then  $p_2$   
 else if  $z_2 = \mathbf{0}$  then  $p_1$   
 else  
 let  
 $d_1 = x_2 \otimes z_1$ ;  
 $d_2 = x_1 \otimes z_2$ ;  
 $l = d_1 \ominus d_2$ ;  
 $m = y_2 \otimes z_1 \ominus y_1 \otimes z_2$   
 in  
 if  $l = \mathbf{0}$  then  
 if  $m = \mathbf{0}$  then *pdouble*  $a$   $p_1$   
 else  $(\mathbf{0}, \mathbf{0}, \mathbf{0})$   
 else  
 let  $h = m \ [\uparrow] (2::nat) \otimes z_1 \otimes z_2 \ominus (d_1 \oplus d_2) \otimes l \ [\uparrow] (2::nat)$   
 in  
 $(l \otimes h,$   
 $(d_2 \otimes l \ [\uparrow] (2::nat) \ominus h) \otimes m \ominus l \ [\uparrow] (3::nat) \otimes y_1 \otimes z_2,$   
 $l \ [\uparrow] (3::nat) \otimes z_1 \otimes z_2))$

**definition** (in *field*) *make-affine* ::  $'a$  *ppoint*  $\Rightarrow 'a$  *point* **where**

*make-affine*  $p$  =  
 (let  $(x, y, z) = p$   
 in if  $z = \mathbf{0}$  then *Infinity* else *Point*  $(x \otimes z) (y \otimes z)$ )

**definition** (in *cring*) *in-carrierp* ::  $'a$  *ppoint*  $\Rightarrow bool$  **where**

$in\text{-carrier}p = (\lambda(x, y, z). x \in carrier\ R \wedge y \in carrier\ R \wedge z \in carrier\ R)$

**definition** (in *cring*)  $on\text{-curve}p :: 'a \Rightarrow 'a \Rightarrow 'a\ ppoint \Rightarrow bool$  **where**

$on\text{-curve}p\ a\ b = (\lambda(x, y, z).$   
 $x \in carrier\ R \wedge y \in carrier\ R \wedge z \in carrier\ R \wedge$   
 $(z \neq \mathbf{0} \longrightarrow$   
 $y [\wedge] (2::nat) \otimes z = x [\wedge] (3::nat) \oplus a \otimes x \otimes z [\wedge] (2::nat) \oplus b \otimes z [\wedge]$   
 $(3::nat)))$

**lemma** (in *cring*)  $on\text{-curve}p\text{-infinity}$  [simp]:  $on\text{-curve}p\ a\ b\ (x, y, \mathbf{0}) = (x \in carrier\ R \wedge y \in carrier\ R)$

**by** (simp add: *on-curvep-def*)

**lemma** (in *field*)  $make\text{-affine}\text{-infinity}$  [simp]:  $make\text{-affine}\ (x, y, \mathbf{0}) = Infinity$

**by** (simp add: *make-affine-def*)

**lemma** (in *cring*)  $on\text{-curve}p\text{-imp-in-carrier}p$  [simp]:  $on\text{-curve}p\ a\ b\ p \Longrightarrow in\text{-carrier}p\ p$

**by** (auto simp add: *on-curvep-def in-carrierp-def*)

**lemma** (in *ell-field*)  $on\text{-curve}p\text{-iff-on-curve}$ :

**assumes**  $a \in carrier\ R\ b \in carrier\ R\ in\text{-carrier}p\ p$

**shows**  $on\text{-curve}p\ a\ b\ p = on\text{-curve}\ a\ b\ (make\text{-affine}\ p)$

**using** *assms*

**proof** (induct *p* rule: *prod-induct3*)

**case** (*fields*  $x\ y\ z$ )

**show**  $on\text{-curve}p\ a\ b\ (x, y, z) = on\text{-curve}\ a\ b\ (make\text{-affine}\ (x, y, z))$

**proof**

**assume**  $H: on\text{-curve}p\ a\ b\ (x, y, z)$

**then have**  $carrier: x \in carrier\ R\ y \in carrier\ R\ z \in carrier\ R$

**and**  $yz: z \neq \mathbf{0} \Longrightarrow$

$y [\wedge] (2::nat) \otimes z = x [\wedge] (3::nat) \oplus a \otimes x \otimes z [\wedge] (2::nat) \oplus b \otimes z [\wedge]$   
 $(3::nat)$

**by** (simp-all add: *on-curvep-def*)

**show**  $on\text{-curve}\ a\ b\ (make\text{-affine}\ (x, y, z))$

**proof** (*cases*  $z = \mathbf{0}$ )

**case** *True*

**then show** *?thesis* **by** (simp add: *on-curve-def make-affine-def*)

**next**

**case** *False*

**then show** *?thesis*

**apply** (simp add: *on-curve-def make-affine-def carrier*)

**apply** (*field*  $yz$  [*OF False*])

**apply** *assumption*

**done**

**qed**

**next**

**assume**  $H: on\text{-curve}\ a\ b\ (make\text{-affine}\ (x, y, z))$

**show**  $on\text{-curve}p\ a\ b\ (x, y, z)$

```

proof (cases z = 0)
  case True
  with ⟨in-carrierp (x, y, z)⟩ show ?thesis
  by (simp add: on-curvep-def in-carrierp-def)
next
  case False
  from ⟨in-carrierp (x, y, z)⟩
  have carrier: x ∈ carrier R y ∈ carrier R z ∈ carrier R
  by (simp-all add: in-carrierp-def)
  from H show ?thesis
  apply (simp add: on-curve-def on-curvep-def make-affine-def carrier False)
  apply (field (prems))
  apply field
  apply (simp-all add: False)
  done
qed
qed
qed

```

```

lemma (in cring) pdouble-in-carrierp:
  a ∈ carrier R ⇒ in-carrierp p ⇒ in-carrierp (pdouble a p)
  by (auto simp add: in-carrierp-def pdouble-def Let-def split: prod.split)

```

```

lemma (in cring) padd-in-carrierp:
  a ∈ carrier R ⇒ in-carrierp p1 ⇒ in-carrierp p2 ⇒ in-carrierp (padd a p1 p2)
  by (auto simp add: padd-def Let-def pdouble-in-carrierp split: prod.split)
  (auto simp add: in-carrierp-def)

```

```

lemma (in cring) pdouble-infinity [simp]: pdouble a (x, y, 0) = (x, y, 0)
  by (simp add: pdouble-def)

```

```

lemma (in cring) padd-infinity-l [simp]: padd a (x, y, 0) p = p
  by (simp add: padd-def)

```

```

lemma (in ell-field) pdouble-correct:
  a ∈ carrier R ⇒ in-carrierp p ⇒
  make-affine (pdouble a p) = add a (make-affine p) (make-affine p)
proof (induct p rule: prod-induct3)
  case (fields x y z)
  then have x ∈ carrier R y ∈ carrier R z ∈ carrier R
  by (simp-all add: in-carrierp-def)
  then show ?case
  apply (auto simp add: add-def pdouble-def make-affine-def equal-neg-zero divide-eq-0-iff
  integral-iff Let-def simp del: minus-divide-left)
  apply field
  apply (simp add: integral-iff)
  apply field

```

```

  apply (simp add: integral-iff)
done
qed

```

**lemma** (in *ell-field*) *padd-correct*:

```

  assumes a: a ∈ carrier R and b: b ∈ carrier R
  and p1: on-curvep a b p1 and p2: on-curvep a b p2
  shows make-affine (padd a p1 p2) = add a (make-affine p1) (make-affine p2)
  using p1
proof (induct p1 rule: prod-induct3)
  case (fields x1 y1 z1)
  note p1' = fields
  from p2 show ?case
proof (induct p2 rule: prod-induct3)
  case (fields x2 y2 z2)
  then have x2 ∈ carrier R y2 ∈ carrier R z2 ∈ carrier R and
    yz2: z2 ≠ 0 ⇒ y2 [↑] (2::nat) ⊗ z2 ⊗ z1 [↑] (3::nat) =
      (x2 [↑] (3::nat) ⊕ a ⊗ x2 ⊗ z2 [↑] (2::nat) ⊕ b ⊗ z2 [↑] (3::nat)) ⊗ z1 [↑]
(3::nat)
    by (simp-all add: on-curvep-def)
  from p1' have x1 ∈ carrier R y1 ∈ carrier R z1 ∈ carrier R and
    yz1: z1 ≠ 0 ⇒ y1 [↑] (2::nat) ⊗ z1 ⊗ z2 [↑] (3::nat) =
      (x1 [↑] (3::nat) ⊕ a ⊗ x1 ⊗ z1 [↑] (2::nat) ⊕ b ⊗ z1 [↑] (3::nat)) ⊗ z2 [↑]
(3::nat)
    by (simp-all add: on-curvep-def)
  show ?case
proof (cases z1 = 0)
  case True
  then show ?thesis
    by (simp add: add-def padd-def make-affine-def)
  next
  case False
  show ?thesis
proof (cases z2 = 0)
  case True
  then show ?thesis
    by (simp add: add-def padd-def make-affine-def)
  next
  case False
  show ?thesis
proof (cases x2 ⊗ z1 ⊖ x1 ⊗ z2 = 0)
  case True
  note x = this
  with ⟨x1 ∈ carrier R⟩ ⟨x2 ∈ carrier R⟩ ⟨z1 ∈ carrier R⟩ ⟨z2 ∈ carrier R⟩
  have x': x2 ⊗ z1 = x1 ⊗ z2 by (simp add: eq-diff0)
  show ?thesis
proof (cases y2 ⊗ z1 ⊖ y1 ⊗ z2 = 0)
  case True
  with ⟨y1 ∈ carrier R⟩ ⟨y2 ∈ carrier R⟩ ⟨z1 ∈ carrier R⟩ ⟨z2 ∈ carrier

```



$R$

```

have  $y: y_2 \otimes z_1 = y_1 \otimes z_2$  by (simp add: eq-diff0)
from  $\langle z_1 \neq 0 \rangle \langle z_2 \neq 0 \rangle x$ 
have  $\text{make-affine}(x_2, y_2, z_2) = \text{make-affine}(x_1, y_1, z_1)$ 
  apply (simp add: make-affine-def)
  apply (rule conjI)
  apply (field x')
  apply simp
  apply (field y)
  apply simp
  done
with  $\text{True } x \langle z_1 \neq 0 \rangle \langle z_2 \neq 0 \rangle p_1'$  fields a show ?thesis
  by (simp add: padd-def pdouble-correct)
next
case False
have  $y_2 [\wedge] (2::\text{nat}) \otimes z_1 [\wedge] (3::\text{nat}) \otimes z_2 =$ 
   $y_1 [\wedge] (2::\text{nat}) \otimes z_1 \otimes z_2 [\wedge] (3::\text{nat})$ 
  by (ring yz1 [OF <z1 ≠ 0>] yz2 [OF <z2 ≠ 0>] x')
then have  $y_2 [\wedge] (2::\text{nat}) \otimes z_1 [\wedge] (3::\text{nat}) \otimes z_2 \otimes z_1 \otimes z_2 =$ 
   $y_1 [\wedge] (2::\text{nat}) \otimes z_1 \otimes z_2 [\wedge] (3::\text{nat}) \otimes z_1 \otimes z_2$ 
  by simp
then have  $(y_2 \otimes z_1) \otimes (y_2 \otimes z_1) = (y_1 \otimes z_2) \otimes (y_1 \otimes z_2)$ 
  apply (field (prems))
  apply (field)
  apply (rule TrueI)
  apply (simp add: <z1 ≠ 0> <z2 ≠ 0>)
  done
with False
have  $y_2 z_1: y_2 \otimes z_1 = \ominus (y_1 \otimes z_2)$ 
  by (simp add: square-eq-iff eq-diff0)
   $\langle y_1 \in \text{carrier } R \rangle \langle y_2 \in \text{carrier } R \rangle \langle z_1 \in \text{carrier } R \rangle \langle z_2 \in \text{carrier } R \rangle$ 
from  $x \text{ False } \langle z_1 \neq 0 \rangle \langle z_2 \neq 0 \rangle$  show ?thesis
  apply (simp add: padd-def add-def make-affine-def Let-def)
  apply (rule conjI)
  apply (rule impI)
  apply (field x')
  apply simp
  apply (field y2z1)
  apply simp
  done
qed
next
case False
then have  $x_1 \otimes z_1 \neq x_2 \otimes z_2$ 
  apply (rule-tac notI)
  apply (erule notE)
  apply (drule sym)
  apply (field (prems))
  apply ring

```

```

    apply (simp add: ⟨z₁ ≠ 0⟩ ⟨z₂ ≠ 0⟩)
  done
with False ⟨z₁ ≠ 0⟩ ⟨z₂ ≠ 0⟩
  ⟨x₁ ∈ carrier R⟩ ⟨x₂ ∈ carrier R⟩ ⟨z₁ ∈ carrier R⟩ ⟨z₂ ∈ carrier R⟩
show ?thesis
apply (auto simp add: padd-def add-def make-affine-def Let-def integral-iff)
  apply field
  apply (simp add: integral-iff)
  apply field
  apply (simp add: integral-iff)
done
qed
qed
qed
qed
qed

```

**lemma** (in *ell-field*) *pdouble-closed*:  
 assumes  $a \in \text{carrier } R$   $b \in \text{carrier } R$  *on-curvep*  $a$   $b$   $p$   
 shows *on-curvep*  $a$   $b$  (*pdouble*  $a$   $p$ )  
**proof** –  
 from ⟨*on-curvep*  $a$   $b$   $p$ ⟩ **have** *in-carrierp*  $p$  **by** *simp*  
 from *assms* **show** ?*thesis*  
 by (simp add: *on-curvep-iff-on-curve pdouble-in-carrierp pdouble-correct*  
*add-closed* ⟨*in-carrierp*  $p$ ⟩)  
**qed**

**lemma** (in *ell-field*) *padd-closed*:  
 assumes  $a \in \text{carrier } R$   $b \in \text{carrier } R$  *on-curvep*  $a$   $b$   $p_1$  *on-curvep*  $a$   $b$   $p_2$   
 shows *on-curvep*  $a$   $b$  (*padd*  $a$   $p_1$   $p_2$ )  
**proof** –  
 from ⟨*on-curvep*  $a$   $b$   $p_1$ ⟩ **have** *in-carrierp*  $p_1$  **by** *simp*  
 from ⟨*on-curvep*  $a$   $b$   $p_2$ ⟩ **have** *in-carrierp*  $p_2$  **by** *simp*  
 from *assms* **show** ?*thesis*  
 by (simp add: *on-curvep-iff-on-curve padd-in-carrierp padd-correct*  
*add-closed* ⟨*in-carrierp*  $p_1$ ⟩ ⟨*in-carrierp*  $p_2$ ⟩)  
**qed**

**primrec** (in *cring*) *ppoint-mult* :: ' $a \Rightarrow \text{nat} \Rightarrow 'a$  *ppoint*  $\Rightarrow 'a$  *ppoint*  
**where**  
 $\text{ppoint-mult } a \ 0 \ p = (\mathbf{0}, \mathbf{0}, \mathbf{0})$   
 |  $\text{ppoint-mult } a \ (\text{Suc } n) \ p = \text{padd } a \ p \ (\text{ppoint-mult } a \ n \ p)$

**lemma** (in *ell-field*) *ppoint-mult-closed* [*simp*]:  
 $a \in \text{carrier } R \Longrightarrow b \in \text{carrier } R \Longrightarrow \text{on-curvep } a \ b \ p \Longrightarrow \text{on-curvep } a \ b$   
 $(\text{ppoint-mult } a \ n \ p)$   
 by (*induct*  $n$ ) (*simp-all* add: *padd-closed*)

**lemma** (in *ell-field*) *ppoint-mult-correct*:  $a \in \text{carrier } R \Longrightarrow b \in \text{carrier } R \Longrightarrow$

*on-curvep a b p*  $\implies$   
*make-affine (ppoint-mult a n p) = point-mult a n (make-affine p)*  
**by** (*induct n*) (*simp-all add: padd-correct*)

**definition** (*in cring*) *proj-eq* :: 'a ppoint  $\Rightarrow$  'a ppoint  $\Rightarrow$  bool **where**  
*proj-eq* = ( $\lambda(x_1, y_1, z_1) (x_2, y_2, z_2)$ ).  
 $(z_1 = \mathbf{0}) = (z_2 = \mathbf{0}) \wedge x_1 \otimes z_2 = x_2 \otimes z_1 \wedge y_1 \otimes z_2 = y_2 \otimes z_1$ )

**lemma** (*in cring*) *proj-eq-refl*: *proj-eq p p*  
**by** (*auto simp add: proj-eq-def*)

**lemma** (*in cring*) *proj-eq-sym*: *proj-eq p p'  $\implies$  proj-eq p' p*  
**by** (*auto simp add: proj-eq-def*)

**lemma** (*in domain*) *proj-eq-trans*:  
*in-carrierp p  $\implies$  in-carrierp p'  $\implies$  in-carrierp p''  $\implies$*   
*proj-eq p p'  $\implies$  proj-eq p' p''  $\implies$  proj-eq p p''*

**proof** (*induct p rule: prod-induct3*)

**case** (*fields x y z*)

**then show** ?*case*

**proof** (*induct p' rule: prod-induct3*)

**case** (*fields x' y' z'*)

**then show** ?*case*

**proof** (*induct p'' rule: prod-induct3*)

**case** (*fields x'' y'' z''*)

**then have** *carrier*:

$x \in \text{carrier } R \ y \in \text{carrier } R \ z \in \text{carrier } R$

$x' \in \text{carrier } R \ y' \in \text{carrier } R \ z' \in \text{carrier } R$

$x'' \in \text{carrier } R \ y'' \in \text{carrier } R \ z'' \in \text{carrier } R$

**and**  $z: (z = \mathbf{0}) = (z' = \mathbf{0}) \ (z' = \mathbf{0}) = (z'' = \mathbf{0})$  **and**

$x \otimes z' \otimes z'' = x' \otimes z \otimes z''$

$y \otimes z' \otimes z'' = y' \otimes z \otimes z''$

**and** *xy*:

$x' \otimes z'' = x'' \otimes z'$

$y' \otimes z'' = y'' \otimes z'$

**by** (*simp-all add: in-carrierp-def proj-eq-def*)

**from**  $\langle x \otimes z' \otimes z'' = x' \otimes z \otimes z'' \rangle$

**have**  $(x \otimes z'') \otimes z' = (x'' \otimes z) \otimes z'$

**by** (*ring (prems) xy*) (*ring xy*)

**moreover from**  $\langle y \otimes z' \otimes z'' = y' \otimes z \otimes z'' \rangle$

**have**  $(y \otimes z'') \otimes z' = (y'' \otimes z) \otimes z'$

**by** (*ring (prems) xy*) (*ring xy*)

**ultimately show** ?*case using z*

**by** (*auto simp add: proj-eq-def carrier conc*)

**qed**

**qed**

**qed**

**lemma** (*in field*) *make-affine-proj-eq-iff*:

$in\text{-carrierp } p \implies in\text{-carrierp } p' \implies proj\text{-eq } p \ p' = (make\text{-affine } p = make\text{-affine } p')$

**proof** (*induct p rule: prod-induct3*)  
**case** (*fields x y z*)  
**then show** *?case*  
**proof** (*induct p' rule: prod-induct3*)  
**case** (*fields x' y' z'*)  
**then have** *carrier:*  
 $x \in carrier \ R \ y \in carrier \ R \ z \in carrier \ R$   
 $x' \in carrier \ R \ y' \in carrier \ R \ z' \in carrier \ R$   
**by** (*simp-all add: in-carrierp-def*)  
**show** *?case*  
**proof**  
**assume**  $proj\text{-eq } (x, y, z) \ (x', y', z')$   
**then have**  $(z = \mathbf{0}) = (z' = \mathbf{0})$   
**and** *xy:*  $x \otimes z' = x' \otimes z \ y \otimes z' = y' \otimes z$   
**by** (*simp-all add: proj-eq-def*)  
**then show**  $make\text{-affine } (x, y, z) = make\text{-affine } (x', y', z')$   
**apply** (*auto simp add: make-affine-def*)  
**apply** (*field xy*)  
**apply** *simp*  
**apply** (*field xy*)  
**apply** *simp*  
**done**

**next**  
**assume**  $H: make\text{-affine } (x, y, z) = make\text{-affine } (x', y', z')$   
**show**  $proj\text{-eq } (x, y, z) \ (x', y', z')$   
**proof** (*cases z = 0*)  
**case** *True*  
**with**  $H$  **have**  $z' = \mathbf{0}$  **by** (*simp add: make-affine-def split: if-split-asm*)  
**with** *True carrier* **show** *?thesis* **by** (*simp add: proj-eq-def*)

**next**  
**case** *False*  
**with**  $H$  **have**  $z' \neq \mathbf{0} \ x \otimes z = x' \otimes z' \ y \otimes z = y' \otimes z'$   
**by** (*simp-all add: make-affine-def split: if-split-asm*)  
**from**  $\langle x \otimes z = x' \otimes z' \rangle$   
**have**  $x \otimes z' = x' \otimes z$   
**apply** (*field (prems)*)  
**apply** *field*  
**apply** (*simp-all add: <z ≠ 0> <z' ≠ 0>*)  
**done**

**moreover from**  $\langle y \otimes z = y' \otimes z' \rangle$   
**have**  $y \otimes z' = y' \otimes z$   
**apply** (*field (prems)*)  
**apply** *field*  
**apply** (*simp-all add: <z ≠ 0> <z' ≠ 0>*)  
**done**

**ultimately show** *?thesis*  
**by** (*simp add: proj-eq-def <z ≠ 0> <z' ≠ 0>*)

qed  
 qed  
 qed  
 qed

**lemma** (in *ell-field*) *pdouble-proj-eq-cong*:  
 $a \in \text{carrier } R \implies \text{in-carrierp } p \implies \text{in-carrierp } p' \implies \text{proj-eq } p \ p' \implies$   
 $\text{proj-eq } (\text{pdouble } a \ p) \ (\text{pdouble } a \ p')$   
 by (*simp add: make-affine-proj-eq-iff pdouble-in-carrierp pdouble-correct*)

**lemma** (in *ell-field*) *padd-proj-eq-cong*:  
 $a \in \text{carrier } R \implies b \in \text{carrier } R \implies \text{on-curvep } a \ b \ p_1 \implies \text{on-curvep } a \ b \ p_1' \implies$   
 $\text{on-curvep } a \ b \ p_2 \implies \text{on-curvep } a \ b \ p_2' \implies \text{proj-eq } p_1 \ p_1' \implies \text{proj-eq } p_2 \ p_2' \implies$   
 $\text{proj-eq } (\text{padd } a \ p_1 \ p_2) \ (\text{padd } a \ p_1' \ p_2')$   
 by (*simp add: make-affine-proj-eq-iff padd-in-carrierp padd-correct*)

end

## 4 Validating the Specification

**theory** *Elliptic-Test*  
**imports**  
*Elliptic-Locale*  
*HOL-Number-Theory.Residues*  
**begin**

### 4.1 Specialized Definitions for Prime Fields

**definition** *mmult* ::  $\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$  (**infixl** \*\*1 70)  
**where**  $x \ **_m \ y = x * y \ \text{mod } m$

**definition** *madd* ::  $\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$  (**infixl** ++1 65)  
**where**  $x \ \ ++_m \ y = (x + y) \ \text{mod } m$

**definition** *msub* ::  $\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$  (**infixl** --1 65)  
**where**  $x \ \ --_m \ y = (x - y) \ \text{mod } m$

**definition** *mpow* ::  $\text{int} \Rightarrow \text{int} \Rightarrow \text{nat} \Rightarrow \text{int}$  (**infixr**  $\overset{\sim}{\sim}$ 1 80)  
**where**  $x \ \ \overset{\sim}{\sim}_m \ n = x \ ^n \ \text{mod } m$

**lemma** (in *residues*) *res-of-natural-eq*:  $\langle n \rangle_{\mathbb{N}} = \text{int } n \ \text{mod } m$   
**by** (*induct n*)  
*(simp-all add: of-natural-def res-zero-eq res-one-eq res-add-eq mod-add-right-eq)*

**lemma** (in *residues*) *res-of-integer-eq*:  $\langle i \rangle = i \ \text{mod } m$   
**by** (*simp add: of-integer-def res-of-natural-eq res-neg-eq mod-minus-eq*)

**lemma** (in *residues*) *res-pow-eq*:  $x \ [\overset{\sim}{\sim}] \ (n::\text{nat}) = x \ ^n \ \text{mod } m$   
**using** *m-gt-one*

**by** (*induct n*)  
(*simp-all add: res-one-eq res-mult-eq mult-ac mod-mult-right-eq*)

**lemma** (*in residues*) *res-sub-eq*:  $(x \bmod m) \ominus (y \bmod m) = (x \bmod m - y \bmod m) \bmod m$   
**by** (*simp add: minus-eq res-neg-eq res-add-eq mod-minus-eq mod-add-eq mod-diff-eq*)

**definition** *mpdouble* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int ppoint*  $\Rightarrow$  *int ppoint* **where**

*mpdouble* *m a p* =  
(*let* (*x, y, z*) = *p*  
in  
if *z* = 0 then *p*  
else  
let  
*l* = 2 mod *m* \*\**m* *y* \*\**m* *z*;  
*n* = 3 mod *m* \*\**m* *x*  $\widetilde{m}$  2 ++*m* *a* \*\**m* *z*  $\widetilde{m}$  2  
in  
(*l* \*\**m* (*n*  $\widetilde{m}$  2 --*m* 4 mod *m* \*\**m* *x* \*\**m* *y* \*\**m* *l*),  
*n* \*\**m* (6 mod *m* \*\**m* *x* \*\**m* *y* \*\**m* *l* --*m* *n*  $\widetilde{m}$  2) --*m*  
2 mod *m* \*\**m* *y*  $\widetilde{m}$  2 \*\**m* *l*  $\widetilde{m}$  2,  
*l*  $\widetilde{m}$  3))

**definition** *mpadd* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int ppoint*  $\Rightarrow$  *int ppoint*  $\Rightarrow$  *int ppoint* **where**

*mpadd* *m a p1 p2* =  
(*let*  
(*x1, y1, z1*) = *p1*;  
(*x2, y2, z2*) = *p2*  
in  
if *z1* = 0 then *p2*  
else if *z2* = 0 then *p1*  
else  
let  
*d1* = *x2* \*\**m* *z1*;  
*d2* = *x1* \*\**m* *z2*;  
*l* = *d1* --*m* *d2*;  
*n* = *y2* \*\**m* *z1* --*m* *y1* \*\**m* *z2*  
in  
if *l* = 0 then  
if *n* = 0 then *mpdouble* *m a p1*  
else (0, 0, 0)  
else  
let *h* = *n*  $\widetilde{m}$  2 \*\**m* *z1* \*\**m* *z2* --*m* (*d1* ++*m* *d2*) \*\**m* *l*  $\widetilde{m}$  2  
in  
(*l* \*\**m* *h*,  
(*d2* \*\**m* *l*  $\widetilde{m}$  2 --*m* *h*) \*\**m* *n* --*m* *l*  $\widetilde{m}$  3 \*\**m* *y1* \*\**m* *z2*,  
*l*  $\widetilde{m}$  3 \*\**m* *z1* \*\**m* *z2*))

**lemma** (*in residues*) *pdouble-residue-eq*: *pdouble* *a p* = *mpdouble* *m a p*  
**by** (*simp only: pdouble-def mpdouble-def*)

*madd-def mmult-def msub-def mpow-def res-zero-eq res-add-eq res-mult-eq res-of-integer-eq  
res-pow-eq res-sub-eq)*

**lemma** (in *residues*) *padd-residue-eq*:  $padd\ a\ p_1\ p_2 = mpadd\ m\ a\ p_1\ p_2$   
**by** (*simp only*: *padd-def mpadd-def pdouble-residue-eq  
madd-def mmult-def msub-def mpow-def res-zero-eq res-add-eq res-mult-eq res-of-integer-eq  
res-pow-eq res-sub-eq Let-def*)

**fun** *fast-ppoint-mult* ::  $int \Rightarrow int \Rightarrow nat \Rightarrow int\ ppoint \Rightarrow int\ ppoint$   
**where**

*fast-ppoint-mult*  $m\ a\ n\ p =$   
 (if  $n = 0$  then  $(0, 0, 0)$   
 else if  $n \bmod 2 = 0$  then  $mpdouble\ m\ a\ (fast-ppoint-mult\ m\ a\ (n\ div\ 2)\ p)$   
 else  $mpadd\ m\ a\ p\ (mpdouble\ m\ a\ (fast-ppoint-mult\ m\ a\ (n\ div\ 2)\ p))$ )

**lemma** *fast-ppoint-mult-0* [*simp*]:  $fast-ppoint-mult\ m\ a\ 0\ p = (0, 0, 0)$   
**by** *simp*

**lemma** *fast-ppoint-mult-even* [*simp*]:  
 $n \neq 0 \implies n \bmod 2 = 0 \implies$   
 $fast-ppoint-mult\ m\ a\ n\ p = mpdouble\ m\ a\ (fast-ppoint-mult\ m\ a\ (n\ div\ 2)\ p)$   
**by** *simp*

**lemma** *fast-ppoint-mult-odd* [*simp*]:  
 $n \neq 0 \implies n \bmod 2 \neq 0 \implies$   
 $fast-ppoint-mult\ m\ a\ n\ p = mpadd\ m\ a\ p\ (mpdouble\ m\ a\ (fast-ppoint-mult\ m\ a\ (n\ div\ 2)\ p))$   
**by** *simp*

**declare** *fast-ppoint-mult.simps* [*simp del*]

**locale** *residues-prime-gt2* = *residues-prime* +  
**assumes** *gt2*:  $2 < p$

**sublocale** *residues-prime-gt2* < *ell-field*  
**using** *gt2*  
**by** *unfold-locales (simp add: res-of-integer-eq res-zero-eq)*

**lemma** (in *residues-prime-gt2*) *fast-ppoint-mult-closed*:  
**assumes**  $a \in carrier\ R\ b \in carrier\ R\ on-curvep\ a\ b\ q$   
**shows**  $on-curvep\ a\ b\ (fast-ppoint-mult\ (int\ p)\ a\ n\ q)$   
**using** *assms*  
**proof** (*induct int p a n q rule: fast-ppoint-mult.induct*)  
**case**  $(1\ a\ n\ q)$   
**show** ?*case*  
**proof** (*cases n = 0*)  
**case** *True*  
**then show** ?*thesis using m-gt-one*  
**by** (*simp add: on-curvep-infinity [simplified res-zero-eq] res-carrier-eq*)

```

next
  case False
  with 1 show ?thesis
    by (cases  $n \bmod 2 = 0$ )
      (simp-all add: padd-residue-eq [symmetric] pdouble-residue-eq [symmetric]
        padd-closed pdouble-closed)
qed
qed

lemma (in residues-prime-gt2) point-mult-residue-eq:
  assumes  $a \in \text{carrier } R$   $b \in \text{carrier } R$  on-curvep  $a$   $b$   $q$  nonsingular  $a$   $b$ 
  shows proj-eq (ppoint-mult  $a$   $n$   $q$ ) (fast-ppoint-mult (int  $p$ )  $a$   $n$   $q$ )
proof -
  from assms
  have point-mult  $a$   $n$  (make-affine  $q$ ) = make-affine (fast-ppoint-mult (int  $p$ )  $a$   $n$ 
 $q$ )
  proof (induct int  $p$   $a$   $n$   $q$  rule: fast-ppoint-mult.induct)
    case (1  $a$   $n$   $q$ )
    show ?case
    proof (cases  $n = 0$ )
      case True
      then show ?thesis by (simp add: make-affine-infinity [simplified res-zero-eq])
    next
      case False
      have point-mult  $a$   $n$  (make-affine  $q$ ) =
        point-mult  $a$  ( $n \text{ div } 2 * 2 + n \bmod 2$ ) (make-affine  $q$ )
        by simp
      also from 1
      have ... = add  $a$  (point-mult  $a$  2 (point-mult  $a$  ( $n \text{ div } 2$ ) (make-affine  $q$ )))
        (point-mult  $a$  ( $n \bmod 2$ ) (make-affine  $q$ ))
        by (simp only: point-mult-mult point-mult-add
          on-curvep-iff-on-curve [symmetric] on-curvep-imp-in-carrierp)
      also have ... = make-affine (fast-ppoint-mult (int  $p$ )  $a$   $n$   $q$ )
        using 1 False
        by (cases  $n \bmod 2 = 0$ )
          (simp-all add: padd-residue-eq [symmetric] pdouble-residue-eq [symmetric]
            add-0-r
              padd-correct pdouble-correct
              fast-ppoint-mult-closed on-curvep-imp-in-carrierp [of  $a$   $b$ ]
              point-mult2-eq-double pdouble-closed
              add-assoc [symmetric] add-comm add-comm' on-curvep-iff-on-curve
                [symmetric])
      finally show ?thesis .
    qed
  qed
  with assms show ?thesis
  by (simp add: make-affine-proj-eq-iff fast-ppoint-mult-closed
    ppoint-mult-correct on-curvep-imp-in-carrierp [of  $a$   $b$ ])
qed

```



**definition** *mmake-affine* :: *int* ⇒ *int ppoint* ⇒ *int point* **where**

```

mmake-affine q p =
  (let (x, y, z) = p
   in if z = 0 then Infinity else
     let (a, b) = bezout-coefficients z q
     in Point (a **q x) (a **q y))

```

**lemma** (in *residues-prime*) *make-affine-residue-eq*:

```

assumes in-carrierp q
shows make-affine q = mmake-affine (int p) q
proof (cases q)
  case (fields x y z)
    show ?thesis
    proof (cases z = 0)
      case True
        with fields show ?thesis by (simp add: make-affine-def mmake-affine-def
res-zero-eq)
      next
        case False
          show ?thesis
          proof (cases bezout-coefficients z (int p))
            case (Pair a b)
              with fields False assms have ¬ int p dvd z
              by (auto simp add: in-carrierp-def res-carrier-eq prime-imp-coprime zdvd-not-zless)
              with p-prime have coprime (int p) z
              by (auto intro: prime-imp-coprime)
              then have coprime z (int p)
              by (simp add: ac-simps)
              then have fst (bezout-coefficients z (int p)) * z +
snd (bezout-coefficients z (int p)) * int p = 1
              by (simp add: bezout-coefficients-fst-snd)
              with m-gt-one have fst (bezout-coefficients z (int p)) * z mod int p = 1
              by (auto dest: arg-cong [of - - λx. x mod int p])
              then have z ⊗ (fst (bezout-coefficients z (int p)) mod int p) = 1
              by (simp add: res-mult-eq res-one-eq mult.commute mod-mult-right-eq)
              with fields assms have inv z = fst (bezout-coefficients z (int p)) mod int p
              by (simp add: inverse-unique in-carrierp-def res-carrier-eq)
              with fields Pair False show ?thesis
              by (simp add: make-affine-def mmake-affine-def res-zero-eq m-div-def
res-mult-eq mmult-def mod-mult-right-eq mult.commute)
            qed
          qed
        qed

```

**definition** *mon-curve* :: *int* ⇒ *int* ⇒ *int* ⇒ *int point* ⇒ *bool* **where**

```

mon-curve m a b p = (case p of
  Infinity ⇒ True
  | Point x y ⇒ 0 ≤ x ∧ x < m ∧ 0 ≤ y ∧ y < m ∧

```



## References

- [1] D. E. Fu and J. A. Solinas. Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2. RFC 5903, Internet Engineering Task Force (IETF), June 2010. Available online at <https://tools.ietf.org/html/rfc5903>.
- [2] C. F. Kerry and P. D. Gallagher. Digital Signature Standard (DSS). Federal Information Processing Standards (FIPS) Publication 186-4, Information Technology Laboratory, National Institute of Standards and Technology (NIST), July 2013. Available online at <https://doi.org/10.6028/NIST.FIPS.186-4>.
- [3] A. Mahboubi and B. Grégoire. Proving Equalities in a Commutative Ring Done Right in Coq. In J. Hurd and T. Melham, editors, *TPHOLS 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113, Oxford, United Kingdom, August 2005. Springer.
- [4] L. Théry. Proving the group law for elliptic curves formally. Technical Report RT-0330, INRIA, March 2007. Available online at <https://hal.inria.fr/inria-00129237>, Coq sources available at <http://coqprime.gforge.inria.fr>.