Radbound University

Verification of TweetNaCl's Curve25519

Peter Schwabe, **Benoît Viguier**, Timmy Weerwag, Freek Wiedijk

Journée GT Méthodes Formelles pour la Sécurité
March 18^{th}, 2019

Institute for Computing and Information Sciences – Digital Security
Radboud University, Nijmegen

Overview

Prelude

Formalization of Elliptic Curves

A quick overview of TweetNaCl

From C to Coq

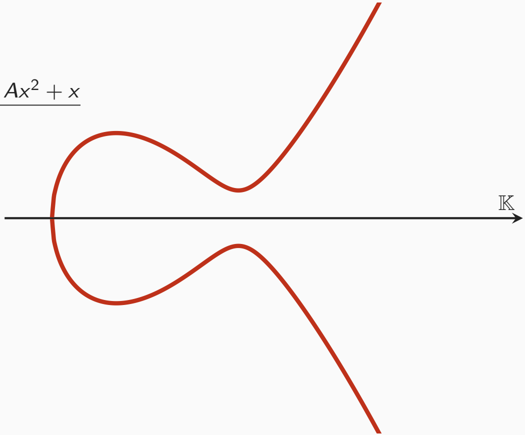
Crypto_Scalarmult n P.x = ([n]P).x ?

Prelude

Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

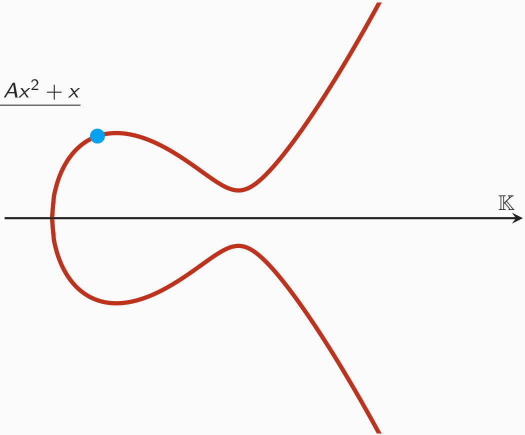
(2) $\{P, Q\} \mapsto P + Q$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

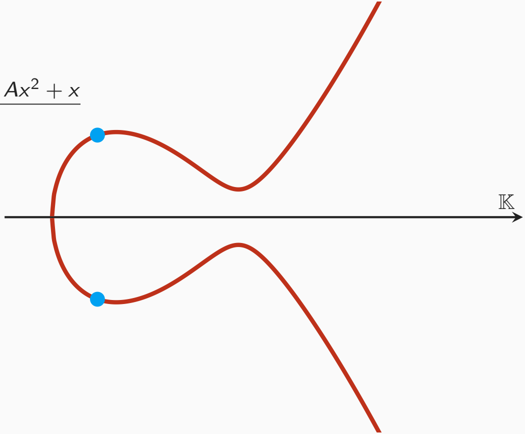
(2) $\{P, Q\} \mapsto P + Q$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

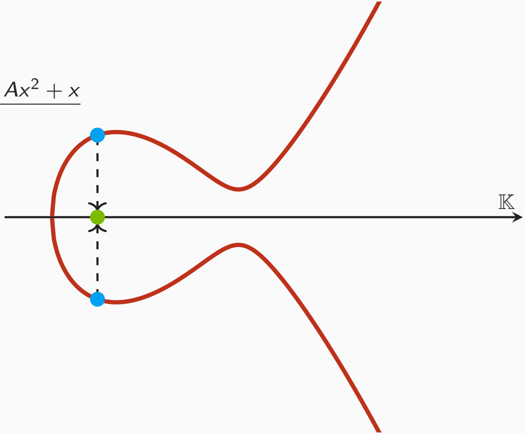
(2) $\{P, Q\} \mapsto P + Q$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

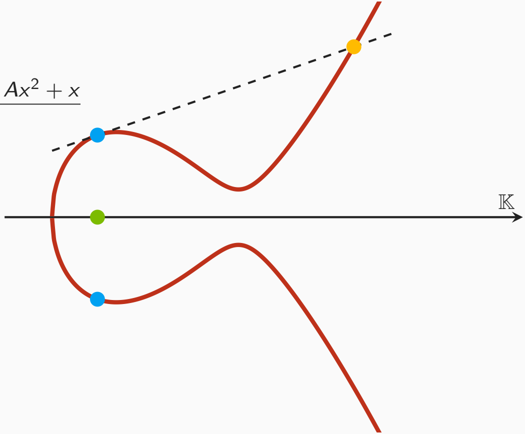
(2) $\{P, Q\} \mapsto P + Q$



<u>*Operations on* $E : By^2 = x^3 + Ax^2 + x$</u>

(1) $P \mapsto [2]P$

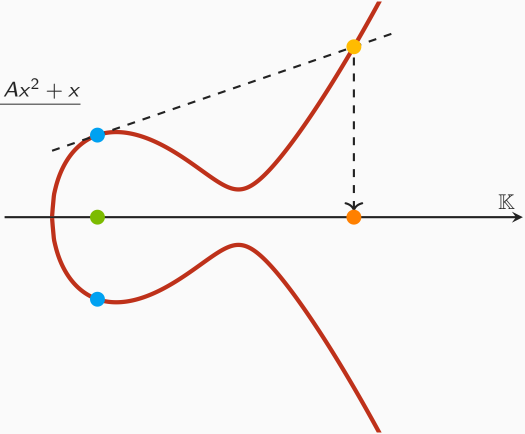
(2) $\{P, Q\} \mapsto P + Q$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

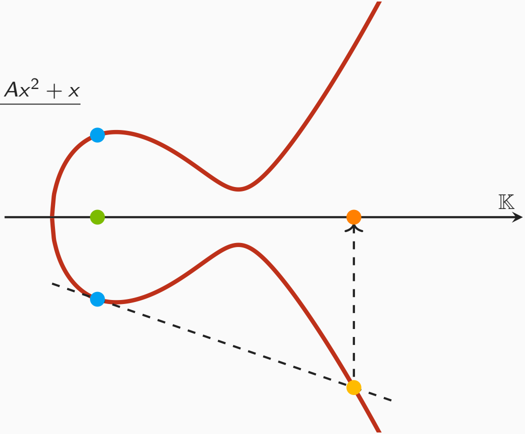
(2) $\{P, Q\} \mapsto P + Q$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$



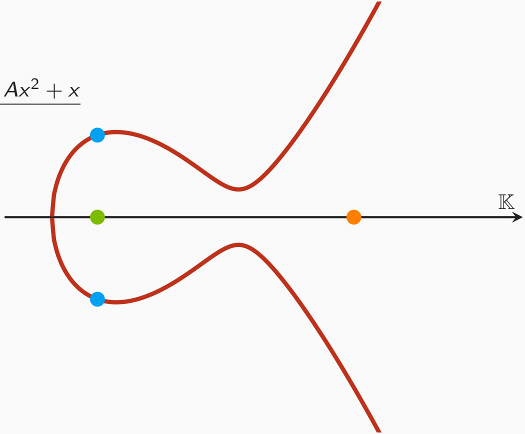
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



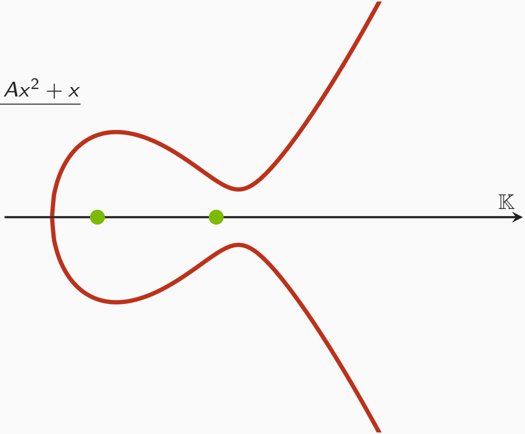
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



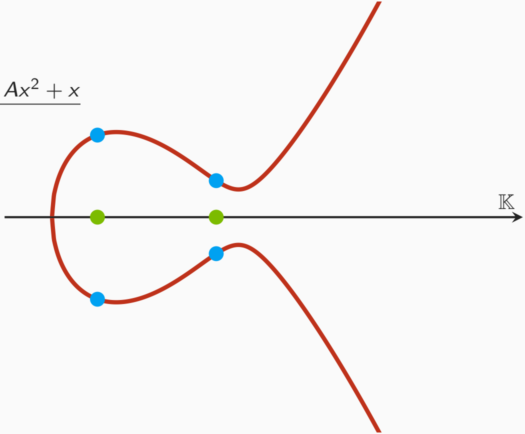
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



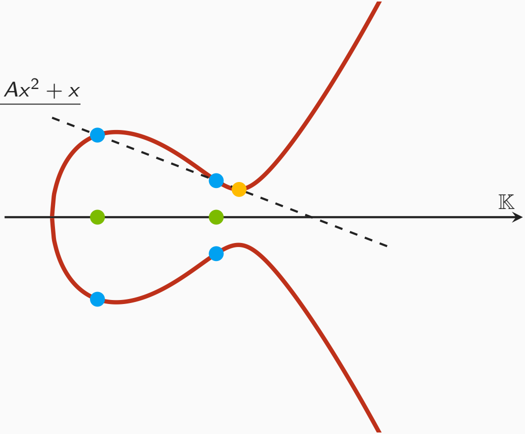
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



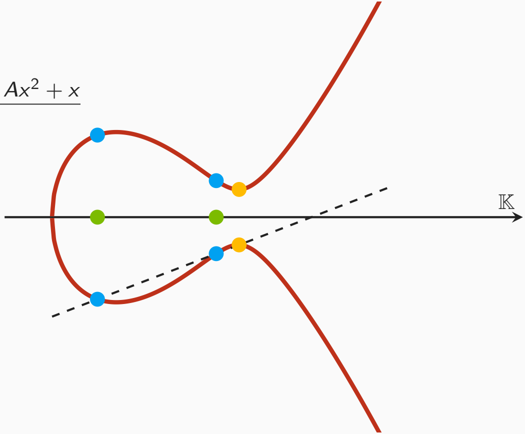
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



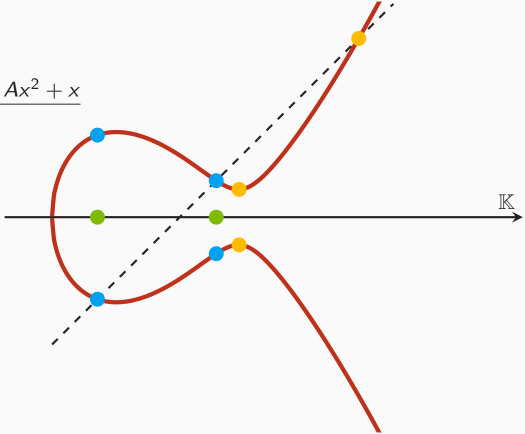
Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$

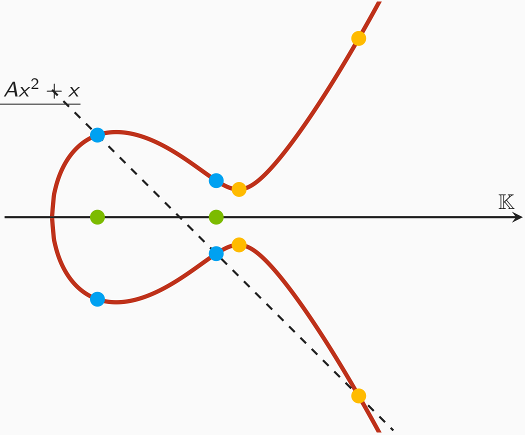


Elliptic Curves 101

Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$



Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$

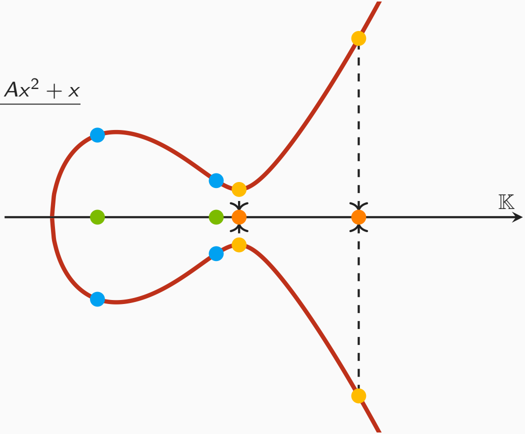
<u>*Operations on* $E : By^2 = x^3 + Ax^2 + x$</u>

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

<u>*Operations on* $\mathbb{P}$</u>

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$



Operations on $E : By^2 = x^3 + Ax^2 + x$

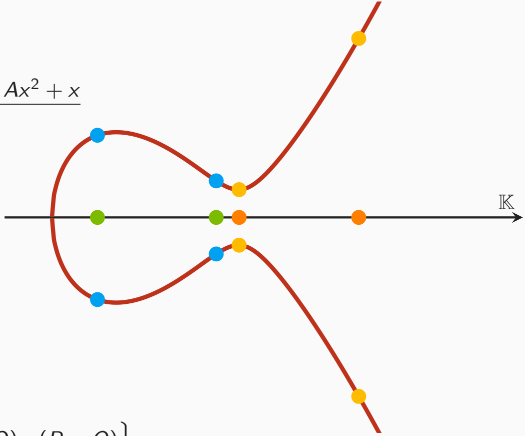
(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$

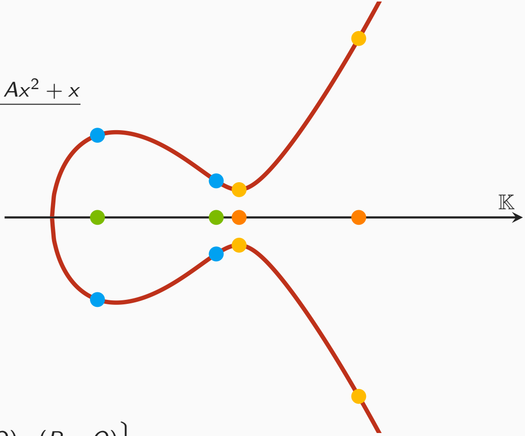
(2) $\left\{\mathbf{x}(P), \mathbf{x}(Q)\right\} \mapsto \left\{\mathbf{x}(P+Q), \mathbf{x}(P-Q)\right\}$



Operations on $E : By^2 = x^3 + Ax^2 + x$

(1) $P \mapsto [2]P$

(2) $\{P, Q\} \mapsto P + Q$



Operations on $\mathbb{P}$

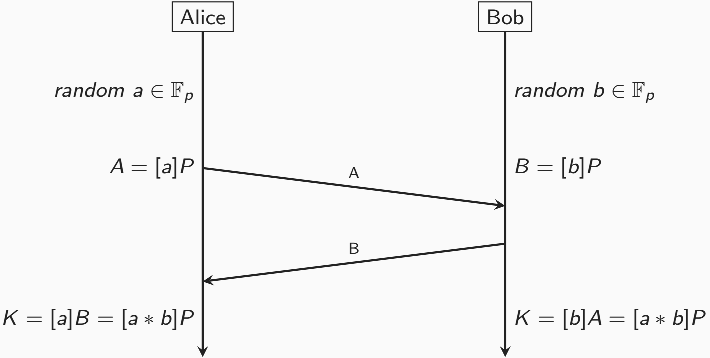
(1) $\mathbf{x}(P) \mapsto \mathbf{x}([2]P)$

(2) $\left\{\mathbf{x}(P), \mathbf{x}(Q)\right\} \mapsto \left\{\mathbf{x}(P+Q), \mathbf{x}(P-Q)\right\}$

$\implies \left\{\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P-Q)\right\} \mapsto \mathbf{x}(P+Q)$

Diffie-Hellman with Elliptic Curves

Public parameter: point P, curve E over $\mathbb{F}_p$



Formalization of Elliptic Curves

Formal definition of a point

```
Inductive point (𝕂: Type) : Type :=
  (* A point is either at Infinity *)
  | EC_Inf : point 𝕂
  (* or (x,y) *)
  | EC_In : 𝕂 → 𝕂 → point 𝕂.

Notation "∞" := (@EC_Inf _).
Notation "(| x , y |)" := (@EC_In _ x y).
```

```
(* Get the x coordinate of p or 0 *)
Definition point_x0 (p : point 𝕂) :=
  if p is (| x, _ |) then x else 0.

Notation "p.x" := (point_x0 p).
```

A Formal Library for Elliptic Curves in the Coq Proof Assistant – Evmorfia-Iro Bartzia, Pierre-Yves Strub
https://hal.inria.fr/hal-01102288

Formal definition of a curve

```
(* Definition of a curve in its Montgomery form *)
(* B y = x^3 + A x^2 + x *)
Record mcuType := {
  A: 𝕂;
  B: 𝕂;
  _ : B ≠ 0;
  _ : A^2 ≠ 4
}

(* is a point p on the curve? *)
Definition oncurve (p: point 𝕂) : bool :=
  match p with
  | ∞ ⇒ true
  | (| x , y |) ⇒ B * y^2 == x^3 + A * x^2 + x
  end.

(* We define a point on a curve as a point
   and the proof that it is on the curve *)
Inductive mc : Type :=
  MC p of oncurve p.
```

Montgomery ladder

```
Definition cswap (c : ℕ) (a b : 𝕂) :=
  if c == 1 then (b, a) else (a, b).

Fixpoint opt_montgomery_rec (n m : ℕ) (x a b c d : 𝕂) : 𝕂 :=
  if m is m.+1 then
    let (a, b) := cswap (bitn n m) a b in
    let (c, d) := cswap (bitn n m) c d in
    let e := a + c in
    let a := a - c in
    let c := b + d in
    let b := b - d in
    let d := e^2 in
    let f := a^2 in
    let a := c * a in
    let c := b * e in
    let e := a + c in
    let a := a - c in
    let b := a^2 in
    let c := d - f in
    let a := c * ((A - 2) / 4) in
    let a := a + d in
    let c := c * a in
    let a := d * f in
    let d := b * x in
    let b := e^2 in
    let (a, b) := cswap (bitn n m) a b in
    let (c, d) := cswap (bitn n m) c d in
    opt_montgomery_rec n m x a b c d
  else
    a / c.

Definition opt_montgomery (n m : ℕ) (x : 𝕂) : 𝕂 :=
  opt_montgomery_rec n m x 1 x 0 1.
```

```
Lemma opt_montgomery_ok :
  forall  (n m: ℕ) (xp : 𝕂) (P : mc M),
  n < 2^m
  → xp ≠ 0

  → P.x = xp
    (* if xp is the x coordinate of P *)

  → opt_montgomery n m xp = ([n]P).x
    (* opt_montgomery n m xp is the x coordinate of [n]P *)
  .
```

($\mathbb{K} = \mathbb{F}_{2^{255}-19}$ *)*
($A = 486662$ *)*
($B = 1$ *)*
(Curve25519 : $B * y^2 = x^3 + A * x^2 + x$ *)*
($y^2 = x^3 + 486662 * x^2 + x$ *)*

```
Definition curve25519_ladder n x = opt_montgomery n 255 x.

Lemma curve25519_ladder_ok :
  forall (n: ℕ) (xp : 𝔽_{2^255−19}) (P : mc Curve25519),
  n < 2^255
  → xp ≠ 0

  → P.x = xp
    (* if xp is the x coordinate of P *)

  → curve25519_ladder n xp = ([n]P).x
    (* curve25519_ladder n xp is the x coordinate of [n]P *)
  .
```

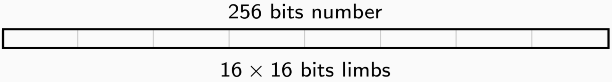
A quick overview of TweetNaCl

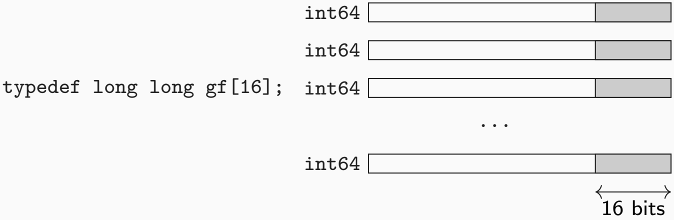
crypto_scalarmult

```
int crypto_scalarmult(u8 *q,const u8 *n,const u8 *p)
{
  u8 z[32]; i64 r; int i; gf x,a,b,c,d,e,f;
  FOR(i,31) z[i]=n[i];
  z[31]=(n[31]&127)|64; z[0]&=248;                 #   Clamping of n
  unpack25519(x,p);
  FOR(i,16) { b[i]=x[i]; d[i]=a[i]=c[i]=0; }
  a[0]=d[0]=1;
  for(i=254;i>=0;--i) {
    r=(z[i>>3]>>(i&7))&1;                          #   i^th bit of n
    sel25519(a,b,r);
    sel25519(c,d,r);
    A(e,a,c);                                      #
    Z(a,a,c);                                      #
    A(c,b,d);                                      #
    Z(b,b,d);                                      #
    S(d,e);                                        #
    S(f,a);                                        #
    M(a,c,a);                                      #  Montgomery Ladder
    M(c,b,e);                                      #
    A(e,a,c);                                      #
    Z(a,a,c);                                      #
    S(b,a);                                        #
    Z(c,d,f);                                      #
    M(a,c,_121665);                                #
    A(a,a,d);                                      #
    M(c,c,a);                                      #
    M(a,d,f);                                      #
    M(d,b,x);                                      #
    S(b,e);                                        #
    sel25519(a,b,r);
    sel25519(c,d,r);
  }
  inv25519(c,c); M(a,a,c);                         #   a / c
  pack25519(q,a);
  return 0;
}
```

Number representation

256-bits integers do not fit into a 64-bits containers...



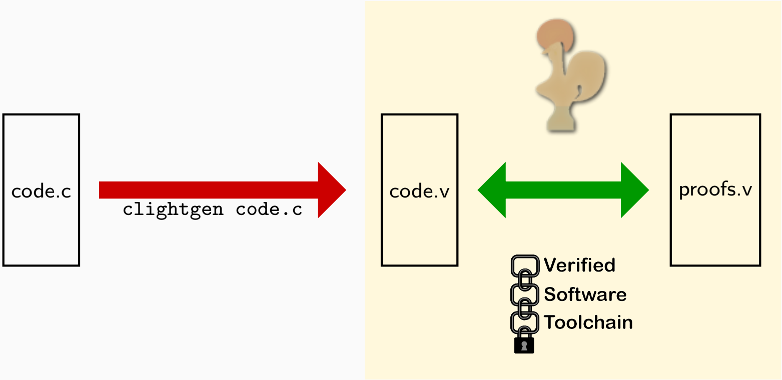


Basic Operations

```
#define FOR(i,n) for (i = 0;i < n;++i)
#define sv static void
typedef long long i64;
typedef i64 gf[16];


sv A(gf o,const gf a,const gf b)    # Addition
{
  int i;
  FOR(i,16) o[i]=a[i]+b[i];         # carrying is done separately
}


sv Z(gf o,const gf a,const gf b)    # Zubstraction
{
  int i;
  FOR(i,16) o[i]=a[i]-b[i];         # carrying is done separately
}


sv M(gf o,const gf a,const gf b)    # Multiplication (school book)
{
  i64 i,j,t[31];
  FOR(i,31) t[i]=0;
  FOR(i,16) FOR(j,16) t[i+j] = a[i]*b[j];
  FOR(i,15) t[i]+=38*t[i+16];
  FOR(i,16) o[i]=t[i];
  car25519(o);                      # carrying
  car25519(o);                      # carrying
}
```

From C to Coq

Proving with VST



Specification: ZofList

```
Variable n: ℤ.
Hypothesis Hn: n > 0.

(*
  in C we have gf[16] here we consider a list of integers (list ℤ)
  of length 16 in this case.

  ZofList converts a list ℤ into its ℤ value
  assume a radix: 2^n
*)
Fixpoint ZofList (a : list ℤ) : ℤ :=
  match a with
  | [] ⇒ 0
  | h :: q ⇒ h + 2^n * ZofList q
  end.

Notation "ℤ.of_list A" := (ZofList A).
```

Specification: Addition

```
Fixpoint A (a b : list ℤ) : list ℤ :=
  match a,b with
  | [], q ⇒ q
  | q,[] ⇒ q
  | h1::q1,h2::q2 ⇒ (Z.add h1 h2) :: A q1 q2
  end.
Notation "a ⊞ b" := (A a b) (at level 60).


Corollary A_correct:
  forall (a b: list ℤ),
    ℤ.of_list (a ⊞ b) = (ℤ.of_list a) + (ℤ.of_list b).
Qed.


Lemma A_bound_len:
  forall (m1 n1 m2 n2: ℤ) (a b: list ℤ),
    length a = length b →
    Forall (λx ⇒ m1 < x < n1) a →
    Forall (λx ⇒ m2 < x < n2) b →
      Forall (λx ⇒ m1 + m2 < x < n1 + n2) (a ⊞ b).
Qed.

Lemma A_length_16:
  forall (a b: list ℤ),
  length a = 16 →
  length b = 16 →
    length (a ⊞ b) = 16.
Qed.
```

Verification: Addition (with VST)

```
sv A(gf o,const gf a,const gf b)
{
  int i;
  FOR(i,16) o[i]=a[i]+b[i];
}
```

```
Definition A_spec :=
DECLARE _A
WITH
  v_o: val, v_a: val, v_b: val,
  sh : share,
  o : list val,
  a : list Z, amin : Z, amax : Z,
  b : list Z, bmin : Z, bmax : Z,

(*-----------------------------------------*)
PRE [ _o OF (tptr tlg), _a OF (tptr tlg), _b OF (tptr tlg) ]
    PROP  (writable_share sh;
          (* For soundness *)                          (* For bounds propagation *)
           Forall (λx ↦ -2^62 < x < 2^62) a;          Forall (λx ↦ amin < x < amax) a;
           Forall (λx ↦ -2^62 < x < 2^62) b;          Forall (λx ↦ bmin < x < bmax) b;

           Zlength a = 16; Zlength b = 16; Zlength o = 16)
    LOCAL (temp _a v_a; temp _b v_b; temp _o v_o)
    SEP   (sh[ v_o ] ←(lg16)— o;
           sh[ v_a ] ←(lg16)— mVI64 a;
           sh[ v_b ] ←(lg16)— mVI64 b)

  (*-----------------------------------------*)
  POST [ tvoid ]
      PROP ( (* Bounds propagation *)
            Forall (λx ↦ amin + bmin < x < amax + bmax) (A a b)
            Zlength (A a b) = 16;
           )
      LOCAL()
      SEP (sh[ v_o ] ←(lg16)— mVI64 (A a b);
           sh[ v_a ] ←(lg16)— mVI64 a;
           sh[ v_b ] ←(lg16)— mVI64 b).
```

Using VST

```
Definition crypto_scalarmult_spec :=
DECLARE _crypto_scalarmult_curve25519_tweet
WITH
  v_q: val, v_n: val, v_p: val, c121665:val,
  sh : share,
  q : list val, n : list Z, p : list Z

(*-----------------------------------------*)
PRE [ _q OF (tptr tuchar), _n OF (tptr tuchar), _p OF (tptr tuchar) ]
    PROP (writable_share sh;
          Forall (λx ↦ 0 ≤ x < 2^8) p;
          Forall (λx ↦ 0 ≤ x < 2^8) n;
          Zlength q = 32; Zlength n = 32; Zlength p = 32 )
    LOCAL(temp _q v_q; temp _n v_n; temp _p v_p; gvar __121665 c121665 )
    SEP  (sh{ v_q }←(uch32)— q;
          sh{ v_n }←(uch32)— mVI n;
          sh{ v_p }←(uch32)— mVI p;
          Ews{ c121665 }←(lg16)— mVI64 c_121665)

(*-----------------------------------------*)
POST [ tint ]
    PROP (Forall (λx ↦ 0 ≤ x < 2^8) (Crypto_Scalarmult n p);
          Zlength (Crypto_Scalarmult n p) = 32)
    LOCAL(temp ret_temp (Vint Int.zero))
    SEP  (sh{ v_q }←(uch32)— mVI (Crypto_Scalarmult n p);
          sh{ v_n }←(uch32)— mVI n;
          sh{ v_p }←(uch32)— mVI p;
          Ews{ c121665 }←(lg16)— mVI64 c_121665
```

Crypto_Scalarmult n P.x = ([n]P).x ?

Generic Operations

```
Class Ops (T T': Type) (Mod: T → T):=
{
  A:   T → T → T;                  (* Addition       over T *)
  M:   T → T → T;                  (* Multiplication over T *)
  Zub: T → T → T;                  (* Substraction   over T *)
  Sq:  T → T;                      (* Squaring       over T *)
  C_0: T;                          (* Constant 0       in T *)
  C_1: T;                          (* Constant 1       in T *)
  C_121665: T;                     (* Constant 121665  in T *)
  Sel25519: ℤ → T → T → T;         (* Select the 2nd or 3rd argument depending of Z *)
  Getbit:   ℤ → T' → ℤ;            (* Return the ith bit of T' *)

  (* Mod conservation *)
  Mod_ZSel25519_eq : forall b p q,  Mod (Sel25519 b p q) = Sel25519 b (Mod p) (Mod q);
  Mod_ZA_eq :        forall p q,    Mod (A p q)          = Mod (A (Mod p) (Mod q));
  Mod_ZM_eq :        forall p q,    Mod (M p q)          = Mod (M (Mod p) (Mod q));
  Mod_ZZub_eq :      forall p q,    Mod (Zub p q)        = Mod (Zub (Mod p) (Mod q));
  Mod_ZSq_eq :       forall p,      Mod (Sq p)           = Mod (Sq (Mod p));

  Mod_red :          forall p,      Mod (Mod p)          = (Mod p)
}.
```

Generic Montgomery Ladder

```
Context {T : Type}.
Context {T' : Type}.
Context {Mod : T → T}.
Context {O : Ops T T' Mod}.

Fixpoint montgomery_rec (m : ℕ) (z : T') (a b c d e f x : T) : (T * T * T * T * T * T) :=
  match m with
  | 0 ⇒ (a,b,c,d,e,f)
  | S n ⇒
      let r := Getbit (ℤ.of_nat n) z in
      let (a, b) := (Sel25519 r a b, Sel25519 r b a) in
      let (c, d) := (Sel25519 r c d, Sel25519 r d c) in
      let e := A a c in
      let a := Zub a c in
      let c := A b d in
      let b := Zub b d in
      let d := Sq e in
      let f := Sq a in
      let a := M c a in
      let c := M b e in
      let e := A a c in
      let a := Zub a c in
      let b := Sq a in
      let c := Zub d f in
      let a := M c C_121665 in
      let a := A a d in
      let c := M c a in
      let a := M d f in
      let d := M b x in
      let b := Sq e in
      let (a, b) := (Sel25519 r a b, Sel25519 r b a) in
      let (c, d) := (Sel25519 r c d, Sel25519 r d c) in
      montgomery_rec n z a b c d e f x
    end.
```

Operations Equivalence

```
Class Ops_Mod_P {T T' U:Type}
                {Mod:U → U} {ModT:T → T}
                `(Ops T T' ModT) `(Ops U U Mod)  :=
{
P:  T  →  U;     (* Projection from T  to U *)
P': T' →  U;     (* Projection from T' to U *)

A_eq:        forall a b, Mod (P (A a b))   = Mod (A (P a) (P b));
M_eq:        forall a b, Mod (P (M a b))   = Mod (M (P a) (P b));
Zub_eq:      forall a b, Mod (P (Zub a b)) = Mod (Zub (P a) (P b));
Sq_eq:       forall a,   Mod (P (Sq a))    = Mod (Sq (P a));

C_121665_eq: P C_121665 = C_121665;
C_0_eq:      P C_0 = C_0;
C_1_eq:      P C_1 = C_1;

Sel25519_eq: forall b p q,  Mod (P (Sel25519 b p q)) = Mod (Sel25519 b (P p) (P q));
Getbit_eq:   forall i p,    Getbit i p = Getbit i (P' p);
}.
```

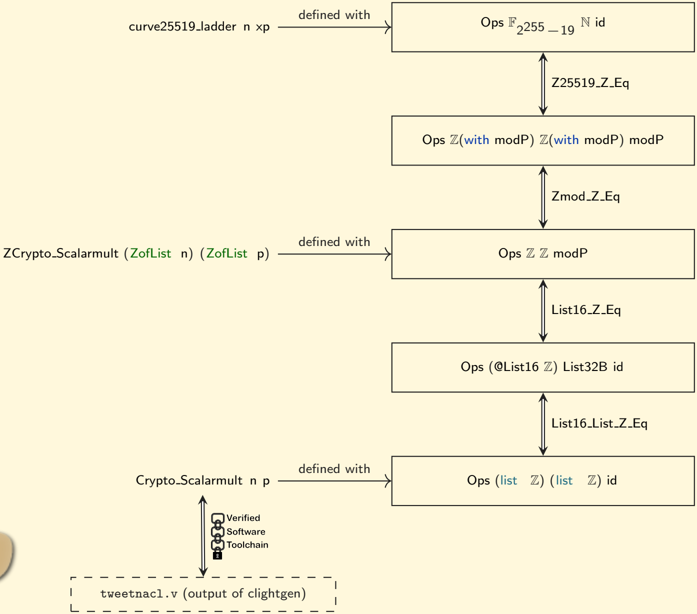
Generic Montgomery Equivalence

```
Context {T:    Type}.
Context {T':   Type}.
Context {U:    Type}.
Context {ModT: T → T}.
Context {Mod:  U → U}.
Context {TO:   Ops T T' ModT}.
Context {UO:   Ops U U  Mod}.
Context {UTO: @Ops_Mod_P T T' U Mod ModT TO UO}.


(* montgomery_rec over T is equivalent to montgomery_rec over U *)
Corollary montgomery_rec_eq_a: forall (n:ℕ) (z:T') (a b c d e f x: T),
  Mod (P (get_a (montgomery_rec n z a b c d e f x))) =                        (* over T *)
  Mod (get_a (montgomery_rec n (P' z) (P a) (P b) (P c) (P d) (P e) (P f) (P x))).  (* over U *)
Qed.

Corollary montgomery_rec_eq_c: forall (n:ℕ) (z:T') (a b c d e f x: T),
  Mod (P (get_c (montgomery_rec n z a b c d e f x))) =                        (* over T *)
  Mod (get_c (montgomery_rec n (P' z) (P a) (P b) (P c) (P d) (P e) (P f) (P x))).  (* over U *)
Qed.
```

Instanciating

```
Definition modP (x: ℤ) := x mod 2^255 − 19.

(* Operations over ℤ *)
Instance Z_Ops :      Ops ℤ ℤ modP := {}.

(* Operations over 𝔽_{2^255 − 19} *)
Instance Z25519_Ops : Ops 𝔽_{2^255 − 19} ℕ id := {}.

(* Equivalence between ℤ (with modP) and ℤ *)
Instance Zmod_Z_Eq :   @Ops_Mod_P Z Z Z modP modP Z_Ops Z_Ops :=
{ P := modP; P' := id }.

(* Equivalence between ℤ (with modP) and 𝔽_{2^255 − 19} *)
Instance Z25519_Z_Eq : @Ops_Mod_P Zmodp.type nat Z modP id Z25519_Ops Z_Ops :=
{ P := val; P' := ℤ.of_nat }.

Inductive List16 (T:Type) := Len (l:list T): Zlength l = 16 → List16 T.
Inductive List32B :=         L32B (l:list ℤ): Forall (λx ⇒ 0 ≤ x < 2^8) l → List32B.

(* Operations over List16,List32 *)
Instance List16_Ops : Ops (@List16 ℤ) List32B id := {}.

(* Equivalence between List16,List32 and ℤ *)
Instance List16_Z_Eq : @Ops_Mod_P (@List16 ℤ) (List32B) Z modP id List16_Ops Z_Ops :=
{  P l := (ZofList 16 (List16_to_List l));       P' l := (ZofList 8 (List32_to_List l));  }.

(* Operations over list of ℤ *)
Instance List_Z_Ops : Ops (list ℤ) (list ℤ) id := {}.

(* Equivalence between List16,List32 and list of ℤ *)
Instance List16_List_Z_Eq : @Ops_Mod_P (List16 ℤ) (List32B) (list ℤ) id id List16_Ops List_Z_Ops :=
{  P := List16_to_List;         P' := List32_to_List      }.
```

Full Equivalence



Final Theorems

```
Theorem Crypto_Scalarmult_Eq :
  forall (n p:list ℤ),

  Zlength n = 32 →                                  (* n is a list of 32 unsigned bytes *)
  Forall (λx ⇒ 0 ≤ x ∧ x < 2^8) n →

  Zlength p = 32 →                                  (* p is a list of 32 unsigned bytes *)
  Forall (λx ⇒ 0 ≤ x ∧ x < 2^8) p →

  ZofList 8 (Crypto_Scalarmult n p) =
      val (curve25519_ladder (Z.to_nat (Zclamp (ZofList 8 n)))
                             (Zmodp.pi (modP (ZUnpack25519 (ZofList 8 p))))).

 (* The operations in Crypto_Scalarmult converted to ℤ yield    *)
 (* to the exact same result as the ladder over 𝔽_{2^255 − 19}   *)
```

```
Lemma curve25519_ladder_ok :
   forall (n: ℕ) (xp : 𝔽_{2^255 − 19}) (P : mc Curve25519),
   n < 2^255
   → xp ≠ 0

   → P.x = xp
     (* if xp is the x coordinate of P *)

   → curve25519_ladder n xp = ([n]P).x.
     (* curve25519_ladder n xp is the x coordinate of [n]P *)
```

Thank you.

Addition

```
Definition A_spec :=
DECLARE _A
WITH
  v_o: val, v_a: val, v_b: val,
  sho : share, sha : share, shb : share,
  o : list val,
  a : list Z, amin : Z, amax : Z,
  b : list Z, bmin : Z, bmax : Z,
  k : Z

(*-------------------------------------------*)
PRE [ _o OF (tptr tlg), _a OF (tptr tlg), _b OF (tptr tlg) ]
    PROP  (writable_share sho; readable_share sha; readable_share shb;
          (* For soundness *)
           Forall (λx ↦ -2^62 < x < 2^62) a;
           Forall (λx ↦ -2^62 < x < 2^62) b;
          (* For bounds propagation *)
           Forall (λx ↦ amin < x < amax) a;
           Forall (λx ↦ bmin < x < bmax) b;
           Zlength a = 16; Zlength b = 16; Zlength o = 16)
    LOCAL (temp _a v_a; temp _b v_b; temp _o v_o)
    SEP   (nm_overlap_array_sep_3 sho sha shb o a b v_o v_a v_b k)

  (*-------------------------------------------*)
  POST [ tvoid ]
      PROP ( (* Bounds propagation *)
            Forall (λx ↦ amin + bmin < x < amax + bmax) (A a b)
            Zlength (A a b) = 16;
           )
      LOCAL()
      SEP (nm_overlap_array_sep_3' sho sha shb (mVI64 (A a b)) (mVI64 a) (mVI64 b) v_o v_a v_b k).
```