A Coq proof of the correctness of X25519 in TweetNaCl

Peter Schwabe, Benoît Viguier, Timmy Weerwag, Freek Wiedijk

Crypto Working Group
November 29\textsuperscript{th}, 2019

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

**Definition**

RFC

\[ s[\{n\}] \leftarrow n \in \mathbb{N}, \]
\[ s[\{p\}] \leftarrow P \in E(F_2) \]

**Pre:**

- \( s[n] \leftarrow n \in \mathbb{N} \)
- \( s[p] \leftarrow P \in E(F_2) \)

**Post:**

- \( s[q] \leftarrow RFC(n, P) \)

**Specification**

\[ \{\text{Pre}\} \text{ Prog } \{\text{Post}\} \]

**Proof**

\[ \text{Pre} \implies RFC(n, P) = n \cdot P \]

\[ \checkmark \]
Prelude
Diffie-Hellman with Elliptic Curves

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

Alice

random $a \in \mathbb{N}$

$A = a \cdot P$

$S = a \cdot B = (a \cdot b) \cdot P$

Bob

random $b \in \mathbb{N}$

$B = b \cdot P$

$S = b \cdot A = (a \cdot b) \cdot P$
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) $\{P, Q\} \mapsto P + Q$
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) $\{P, Q\} \mapsto P + Q$

Operations on $\mathbb{P}$

(1) $xDBL : x(P) \mapsto x([2]P)$
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) $\times \text{DBL} : x(P) \mapsto x([2]P)$
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) $x\text{DBL} : x(P) \mapsto x([2]P)$
**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. $xDBL : x(P) \mapsto x([2]P)$
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) $x\text{DBL} : x(P) \mapsto x([2]P)$
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) $x_{\text{DBL}} : x(P) \mapsto x([2]P)$
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) $x_{\text{DBL}} : x(P) \mapsto x([2]P)$
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. \( \text{xDBL}: x(P) \mapsto x([2]P) \)
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) $\text{xDBL} : x(P) \mapsto x([2]P)$

(2) $\{x(P), x(Q)\} \mapsto \{x(P + Q), x(P - Q)\}$
**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) $\times\text{DBL} : x(P) \mapsto x([2]P)$

(2) $\times\text{ADD} : \Big\{x(P), x(Q), x(P - Q)\Big\} \mapsto x(P + Q)$
Algorithm 1 Montgomery ladder for scalar mult.

Input:  $x$-coordinate $x_P$ of a point $P$, scalar $n$ with $n < 2^m$

Output:  $x$-coordinate $x_Q$ of $Q = n \cdot P$

$Q = (X_Q : Z_Q) \leftarrow (1 : 0)$
$R = (X_R : Z_R) \leftarrow (x_P : 1)$

for $k := m$ down to 1 do
    $(Q, R) \leftarrow \text{CSWAP}((Q, R), k^{\text{th}} \text{ bit of } n)$
    $Q \leftarrow \text{xDBL}(Q)$
    $R \leftarrow \text{xADD}(x_P, Q, R)$
    $(Q, R) \leftarrow \text{CSWAP}((Q, R), k^{\text{th}} \text{ bit of } n)$
end for

return $X_Q/Z_Q$
A quick overview of TweetNaCl
```c
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);
        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);
    pack25519(q, a);
    return 0;
```
256-bits integers do not fit into a 64-bits containers...

256 bits number

16 × 16 bits limbs

typedef long long i64;
typedef i64 gf[16];
#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) # Subtraction
{  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
Formalizing X25519 from RFC 7748
The specification of X25519 in RFC 7748 is formalized by RFC in Coq.

More formally:

```coq
Definition RFC (n: list Z) (p: list Z) : list Z :=
  let k := decodeScalar25519 n in
  let u := decodeUCoordinate p in
  let t := montgomery_rec
    255 (* iterate 255 times *)
    k (* clamped n *)
    1 (* x_2 *)
    u (* x_3 *)
    0 (* z_2 *)
    1 (* z_3 *)
    0 (* dummy *)
    0 (* dummy *)
  u (* x_1 *)
  in
  let a := get_a t in
  let c := get_c t in
  let o := ZPack25519 (Z.mul a (ZInv25519 c))
  in encodeUCoordinate o.
```
Fixpoint montgomery_rec (m : nat) (z : T')
(a: T) (b: T) (c: T) (d: T) (e: T) (f: T) (x: T) :
(* a: x2  b: x3  c: z2  d: z3  x: x1 *)
(T * T * T * T * T * T) :=
match m with
| 0%nat ⇒ (a,b,c,d,e,f)
| S n ⇒
  let r := Getbit (Z.of_nat n) z in
  (* swap ← k_t *)
  let (a, b) := (Sel25519 r a b, Sel25519 r b a) in
  (* (x2, x3) = cswap(swap, x2, x3) *)
  let (c, d) := (Sel25519 r c d, Sel25519 r d c) in
  (* (z2, z3) = cswap(swap, z2, z3) *)
  let e := a + c in
  (* A = x2 + z2 *)
  let a := a - c in
  (* B = x2 - z2 *)
  let c := b + d in
  (* C = x3 + z3 *)
  let b := b - d in
  (* D = x3 - z3 *)
  let d := e * 2 in
  (* AA = A^2 *)
  let f := a * 2 in
  (* BB = B^2 *)
  let c := b * e in
  (* DA = D * A *)
  let e := a + c in
  (* x3 = (DA + CB)^2 *)
  let a := a - c in
  (* z3 = x1 * (DA - CB)^2 *)
  let b := a * 2 in
  (* z3 = x1 * (DA - CB)^2 *)
  let c := d - f in
  (* E = AA - BB *)
  let a := c * C_121665 in
  (* z2 = E * (AA + a24 * E) *)
  let a := a + d in
  (* z2 = E * (AA + a24 * E) *)
  let c := c * a in
  (* z2 = E * (AA + a24 * E) *)
  let a := d + f in
  (* z2 = AA * BB *)
  let d := b * x in
  (* z3 = x1 * (DA - CB)^2 *)
  let b := e * 2 in
  (* x3 = (DA + CB)^2 *)
  let (a, b) := (Sel25519 r a b, Sel25519 r b a) in
  (* (x2, x3) = cswap(swap, x2, x3) *)
  let (c, d) := (Sel25519 r c d, Sel25519 r d c) in
  (* (z2, z3) = cswap(swap, z2, z3) *)
montgomery_rec n z a b c d e f x end.
Let \( \text{ZofList} : \mathbb{Z} \rightarrow \text{list } \mathbb{Z} \rightarrow \mathbb{Z} \), a function given \( n \) and a list \( l \) returns its little endian decoding with radix \( 2^n \).

\[
\text{Fixpoint ZofList} \{n: \mathbb{Z}\} (a:\text{list } \mathbb{Z}) : \mathbb{Z} :=
\begin{array}{l}
\text{match } a \text{ with} \\
| [] \Rightarrow 0 \\
| h :: q \Rightarrow h + 2^n \times \text{ZofList} q \\
\end{array}
\]

Let \( \text{ListofZ32} : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{list } \mathbb{Z} \), given \( n \) and \( a \) returns \( a \)'s little-endian encoding as a list with radix \( 2^n \).

\[
\text{Fixpoint ListofZn_fp} \{n: \mathbb{Z}\} (a: \mathbb{Z}) (f:\text{nat}) : \text{list } \mathbb{Z} :=
\begin{array}{l}
\text{match } f \text{ with} \\
| 0\\text{nat} \Rightarrow [] \\
| \text{S} \text{ fuel} \Rightarrow (a \text{ mod } 2^n) :: \text{ListofZn_fp} (a/2^n) \text{ fuel} \\
\end{array}
\]

\[
\text{Definition ListofZ32} \{n: \mathbb{Z}\} (a: \mathbb{Z}) : \text{list } \mathbb{Z} :=
\text{ListofZn_fp} n a 32.
\]
ListofZ32 and ZofList are inverse to each other.

Lemma ListofZ32_ZofList_Zlength: forall (l:list Z),
  Forall (\lambda x \Rightarrow 0 \leq x < 2^n) l \rightarrow
  Zlength l = 32 \rightarrow
  ListofZ32 n (ZofList n l) = l.
Qed.

With those tools at hand, we formally define the decoding and encoding as specified in the RFC.

Definition decodeScalar25519 (l: list Z) : Z :=
  ZofList 8 (clamp l).

Definition decodeUCoordinate (l: list Z) : Z :=
  ZofList 8 (upd_nth 31 l
           (Z.land (nth 31 l 0) 127)).

Definition encodeUCoordinate (x: Z) : list Z :=
  ListofZ32 8 x.
From C to Coq
Hoare logic 101

\{\text{Pre}\} \text{ Prog} \{\text{Post}\}

where \text{Pre} and \text{Post} are assertions and Prog is a fragment of code.

“when the precondition \text{Pre} is met, executing Prog will yield postcondition \text{Post}”.

Sequent Rule in Hoare logic:

\[
\text{Hoare-Seq} \quad \frac{\{P\} \text{C}_1\{Q\} \quad \{Q\} \text{C}_2\{R\}}{\{P\} \text{C}_1; \text{C}_2\{R\}}
\]
Fixpoint Low.A (a b : list Z) : list Z :=
  match a, b with
  | [] , q ⇒ q
  | q, [] ⇒ q
  | h1::q1, h2::q2 ⇒ (Z.add h1 h2) :: Low.A q1 q2
  end.
Notation "a ⊞ b" := (Low.A a b) (at level 60).

Corollary A_correct:
  forall (a b: list Z),
  ZofList 16 (a ⊞ b) = (ZofList 16 a) + (ZofList 16 b).
Qed.

Lemma A_bound_len:
  forall (m1 n1 m2 n2: Z) (a b: list Z),
  length a = length b →
  Forall (λ x ⇒ m1 < x < n1) a →
  Forall (λ x ⇒ m2 < x < n2) b →
  Forall (λ x ⇒ 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.
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 *)
  Forall (λ x ↦ -2^62 < x < 2^62) a;
  Forall (λ x ↦ amin < x < amax) a;
  Forall (λ x ↦ -2^62 < x < 2^62) b;
  Forall (λ x ↦ 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 (λ x ↦ amin + bmin < x < amax + bmax) (Low.A a b)
  Zlength (A a b) = 16;
)
LOCAL()
SEP (sh{v_o} ← (lg16)− mVI64 (Low.A a b);
    sh{v_a} ← (lg16)− mVI64 a;
    sh{v_b} ← (lg16)− mVI64 b).

sv A(gf o,const gf a,const gf b) {
  int i;
  FOR(i,16) o[i]=a[i]+b[i];
}
(1) We define Low.A; Low.M; Low.Sq; Low.Zub; Unpack25519; clamp; Pack25519; Inv25519; car25519 to have the same behavior as the low level C code.

(2) We define Crypto_Scalarmult with Low.A; Low.M; Low.Sq; Low.Zub; Unpack25519; clamp; Pack25519; Inv25519; car25519; montgomery_rec.

(3) We prove that Low.M; Low.A; Low.Sq; Low.Zub; Unpack25519; clamp; Pack25519; Inv25519; car25519 have the same behavior over list Z as their equivalent over Z with :GF (in $\mathbb{Z}_{2^{255}-19}$).

(4) We prove that Crypto_Scalarmult performs the same computation as RFC.

Lemma Crypto_Scalarmult_RFC_eq :
   forall (n: list Z) (p: list Z),
   Zlength n = 32 →
   Zlength p = 32 →
   Forall ($\lambda$ x ⇒ 0 ≤ x ∧ x < $2^8$) n →
   Forall ($\lambda$ x ⇒ 0 ≤ x ∧ x < $2^8$) p →
   Crypto_Scalarmult n p = RFC n p.
Qed.
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 (\lambda x \mapsto 0 \leq x < 2^{8}) p;
    Forall (\lambda x \mapsto 0 \leq 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 (\lambda x \mapsto 0 \leq x < 2^{8}) (RFC n p);
    Zlength (RFC n p) = 32)
LOCAL(temp ret_temp (Vint Int.zero))
SEP (sh[ {v_q}]←(uch32)− mVI (RFC n p);
    sh[ {v_n}]←(uch32)− mVI n;
    sh[ {v_p}]←(uch32)− mVI p;
    Ews{ c121665 }←(lg16)− mVI64 c_{121665}
The implementation of X25519 in TweetNaCl (crypto_scalarmult) matches the specifications of RFC 7748 (RFC).

More formally:

**Theorem** body_crypto_scalarmult:

(* VST boiler plate. *)

semax_body

(* Clight translation of TweetNaCl. *)

Vprog

(* Hoare triples for function calls. *)

Gprog

(* function we verify. *)

f_crypto_scalarmult_curve25519_tweet

(* Our Hoare triple, see below. *)

crypto_scalarmult_spec.
Formalization of Elliptic Curves
Formal definition of a point

\textbf{Inductive point (K: Type) : Type :=}
(* A point is either at Infinity *)
| EC_Inf : point K
(* or (x, y) *)
| EC_In : K \rightarrow K \rightarrow point K.

\textbf{Notation } "\infty" := (@EC_Inf _).
\textbf{Notation } "(| x , y |)" := (@EC_In x y).

(* Get the x coordinate of p or 0 *)
\textbf{Definition} point\_x0 (p : point K) :=
if p is (| x, _ |) then x else 0.

\textbf{Notation } "p.x" := (point\_x0 p).
Formal definition of a curve

Definition

Let \( a \in \mathbb{K}\{−2, 2\} \), and \( b \in \mathbb{K}\{0\} \). The elliptic curve \( M_{a,b} \) is defined by the equation:

\[
by^2 = x^3 + ax^2 + x,
\]

\( M_{a,b}(\mathbb{K}) \) is the set of all points \((x, y) \in \mathbb{K}^2\) satisfying the \( M_{a,b} \) along with an additional formal point \( O \), “at infinity”.

\[
(* \; B \; y = x^3 + A \; x^2 + x \; *)
\]
Record mcuType := \{ A: \mathbb{K}; B: \mathbb{K}; _: B \neq 0; _: A^2 \neq 4 \}

\[
(* \; is \; a \; point \; p \; on \; the \; curve? \; *)
\]
Definition oncurve (p : point \mathbb{K}) :=
if p is (| x, y |)
  then cB * y^2 == x^3 + cA * x^2 + x
  else true.

\[
(* \; We \; define \; a \; point \; on \; a \; curve \; as \; a \; point \; and \; the \; proof \; that \; it \; is \; on \; the \; curve \; *)
\]
Inductive mc : Type := MC p of oncurve p.
Formal definition of the operations over a curve

Definition $\text{neg} (p: \text{point } \mathbb{K}) :=$
if $p$ is $(| x, y |)$ then $(| x, -y |)$ else $\infty$.

Definition $\text{add} (p_1, p_2: \text{point } \mathbb{K}) :=$
match $p_1$, $p_2$ with
| $\infty$, $-$ $\Rightarrow$ $p_2$
| $-$, $\infty$ $\Rightarrow$ $p_1$
| $(| x_1, y_1 |), (| x_2, y_2 |)$ $\Rightarrow$
if $x_1 = x_2$ then
if $(y_1 = y_2)$ && $(y_1 \neq 0)$ then ... $\infty$
else
$$s := \frac{(y_2 - y_1)}{(x_2 - x_1)}$$
let $x_s := s^2 \cdot B - A - x_1 - x_2$ in
$$(-s \cdot (x_s - x_1) - y_1)$$
end

Notation "$-x" := (\text{neg } x)$.
Notation "$x + y" := (\text{add } x y)$.
Notation "$x - y" := (x + (-y))$.
We define $\chi$ and $\chi_0$ to return the $x$-coordinate of points on a curve.

**Definition**

Let $\chi$ and $\chi_0$:

- $\chi : M_{a,b}(\mathbb{K}) \to \mathbb{K} \cup \{\infty\}$
  such that $\chi(O) = \infty$ and $\chi((x, y)) = x$.
- $\chi_0 : M_{a,b}(\mathbb{K}) \to \mathbb{K}$
  such that $\chi_0(O) = 0$ and $\chi_0((x, y)) = x$.

Montgomery curves make use of projective coordinates. Points are represented with triples $(X : Y : Z)$, with the exception of $(0 : 0 : 0)$

For all $\lambda \neq 0$, the triples $(X : Y : Z)$ and $(\lambda X : \lambda Y : \lambda Z)$ represent the same point.

For $Z \neq 0$, the projective point $(X : Y : Z)$ corresponds to the point $(X/Z, Y/Z)$ on the affine plane.

Likewise the point $(X, Y)$ on the affine plane corresponds to $(X : Y : 1)$ on the projective plane.
Lemma

Let $M_{a,b}$ be a Montgomery curve such that $a^2 - 4$ is not a square in $\mathbb{K}$, and let $X_1, Z_1, X_2, Z_2, X_4, Z_4 \in \mathbb{K}$, such that $(X_1, Z_1) \neq (0, 0)$, $(X_2, Z_2) \neq (0, 0)$, $X_4 \neq 0$ and $Z_4 \neq 0$. Define

\[ X_3 = Z_4((X_1 - Z_1)(X_2 + Z_2) + (X_1 + Z_1)(X_2 - Z_2))^2 \]
\[ Z_3 = X_4((X_1 - Z_1)(X_2 + Z_2) - (X_1 + Z_1)(X_2 - Z_2))^2, \]

then for any point $P_1$ and $P_2$ in $M_{a,b}(\mathbb{K})$ such that $X_1/Z_1 = \chi(P_1)$, $X_2/Z_2 = \chi(P_2)$, and $X_4/Z_4 = \chi(P_1 - P_2)$, we have $X_3/Z_3 = \chi(P_1 + P_2)$.

Remark: These definitions should be understood in $\mathbb{K} \cup \{\infty\}$. If $x \neq 0$ then we define $x/0 = \infty$. 
Lemma

Let $M_{a,b}$ be a Montgomery curve such that $a^2 - 4$ is not a square in $\mathbb{K}$, and let $X_1, Z_1 \in \mathbb{K}$, such that $(X_1, Z_1) \neq (0, 0)$. Define

$$c = (X_1 + Z_1)^2 - (X_1 - Z_1)^2$$
$$X_3 = (X_1 + Z_1)^2(X_1 - Z_1)^2$$
$$Z_3 = c \left( (X_1 + Z_1)^2 + \frac{a - 2}{4} \times c \right),$$

then for any point $P_1$ in $M_{a,b}(\mathbb{K})$ such that $X_1/Z_1 = \chi(P_1)$, we have $X_3/Z_3 = \chi(2P_1)$. 
Correctness of the Montgomery ladder

By combining the Montgomery ladder with the previous formula, we define a ladder \texttt{opt\_montgomery} (in which \(K\) has not been fixed yet).

**Hypothesis**

\[ a^2 - 4 \text{ is not a square in } K. \]

We prove its correctness.

**Theorem**

For all \(n, m \in \mathbb{N}, x \in K, P \in M_{a,b}(K), \) if \(\chi_0(P) = x\) then \texttt{opt\_montgomery} returns \(\chi_0(n \cdot P)\)

\[
\text{Theorem opt\_montgomery\_ok} (n \text{ m: nat}) (x : K) : \\
\quad \text{n < } 2^m \rightarrow \\
\quad \forall (p : \text{mc M}), p\#x0 = x \\
\quad \quad \quad \ast \text{ if } x \text{ is the } x\text{-coordinate of } P \ast \\
\quad \quad \rightarrow \text{opt\_montgomery n m x = (p + n)\#x0.} \\
\quad \quad \quad \ast \text{ opt\_montgomery n m xp is the } x\text{-coordinate of } n \cdot P \ast.
\]

Qed.
Problem!

\[ a^2 - 4 \text{ is a square in } \mathbb{F}_{p^2} \]
Curve25519 ladder

\[ (* a^2 - 4 is not a square in \mathbb{F}_{2^{255-19}}. *) \]

**Fact a_not_square:** \( \forall x : \mathbb{F}_{2^{255-19}}, x^2 \neq (\text{Zmodp.pi 486662})^2 - 4. \)

\[ (* 2 is not a square in \mathbb{F}_{2^{255-19}}. *) \]

**Fact two_not_square:** \( \forall x : \mathbb{F}_{2^{255-19}}, x^2 \neq 2. \)

We now consider \( M_{486662,1}(\mathbb{F}_p) \) and \( M_{486662,2}(\mathbb{F}_p) \), one of its quadratic twists.

**Definition**

We instantiate `opt_montgomery` in two specific ways:

– `Curve25519_Fp(n, x)` for \( M_{486662,1}(\mathbb{F}_p) \).
– `Twist25519_Fp(n, x)` for \( M_{486662,2}(\mathbb{F}_p) \).

`Curve25519_Fp(n, x)` and `Twist25519_Fp(n, x)` do not depend on \( b \).

**Lemma curve_twist_eq:** \( \forall n x, \)

\[
\text{curve25519_Fp_ladder n x} = \text{twist25519_Fp_ladder n x}.
\]

Qed.
We derive the following two lemmas:

**Lemma**

For all \( x \in \mathbb{F}_p, \ n \in \mathbb{N}, \ P \in \mathbb{F}_p \times \mathbb{F}_p \), such that \( P \in M_{486662,1}(\mathbb{F}_p) \) and \( \chi_0(P) = x \).

Given \( n \) and \( x \), \( \text{Curve25519}_Fp(n, x) = \chi_0(n \cdot P) \).

**Lemma**

For all \( x \in \mathbb{F}_p, \ n \in \mathbb{N}, \ P \in \mathbb{F}_p \times \mathbb{F}_p \) such that \( P \in M_{486662,2}(\mathbb{F}_p) \) and \( \chi_0(P) = x \).

Given \( n \) and \( x \), \( \text{Twist25519}_Fp(n, x) = \chi_0(n \cdot P) \).
As 2 is not a square in \( \mathbb{F}_p \) we have:

**Lemma**

For all \( x \) in \( \mathbb{F}_p \), there exists \( y \) in \( \mathbb{F}_p \) such that \( y^2 = x \) \( \lor \) \( 2y^2 = x \)

Thus:

**Lemma**

For all \( x \in \mathbb{F}_p \), there exists a point \( P \) in \( M_{486662,1}(\mathbb{F}_p) \) or in \( M_{486662,2}(\mathbb{F}_p) \) such that the \( x \)-coordinate of \( P \) is \( x \).

And formally:

**Lemma** \( x \text{.is_on_curve_or_twist} \):

\[
\forall x : \mathbb{F}_{2^{255} - 19}, \\
\exists (p : \text{mc curve25519_mcuType}), p \neq 0 = x \lor \\
\exists (p' : \text{mc twist25519_mcuType}), p' \neq 0 = x.
\]

Qed.
We define the two morphisms:

**Definition**

*Define the functions* $\varphi_c$, $\varphi_t$ *and* $\psi$.

- $\varphi_c : M_{486662,1}(\mathbb{F}_p) \mapsto M_{486662,1}(\mathbb{F}_{p^2})$ *such that* $\varphi((x, y)) = ((x, 0), (y, 0))$.
- $\varphi_t : M_{486662,2}(\mathbb{F}_p) \mapsto M_{486662,1}(\mathbb{F}_{p^2})$ *such that* $\varphi((x, y)) = ((x, 0), (0, y))$.
- $\psi : \mathbb{F}_{p^2} \mapsto \mathbb{F}_p$ *such that* $\psi(x, y) = (x)$.

And prove:

**Lemma**

*For all* $n \in \mathbb{N}$, *for all point* $P \in \mathbb{F}_p \times \mathbb{F}_p$ *on the curve* $M_{486662,1}(\mathbb{F}_p)$ *(respectively on the quadratic twist* $M_{486662,2}(\mathbb{F}_p))$, *we have:*

\[
P \in M_{486662,1}(\mathbb{F}_p) \implies \varphi_c(n \cdot P) = n \cdot \varphi_c(P)
\]

\[
P \in M_{486662,2}(\mathbb{F}_p) \implies \varphi_t(n \cdot P) = n \cdot \varphi_t(P)
\]

Notice that:

\[
\forall P \in M_{486662,1}(\mathbb{F}_p), \quad \psi(\chi_0(\varphi_c(P))) = \chi_0(P)
\]

\[
\forall P \in M_{486662,2}(\mathbb{F}_p), \quad \psi(\chi_0(\varphi_t(P))) = \chi_0(P)
\]
Theorem

For all $n \in \mathbb{N}$, such that $n < 2^{255}$, for all $x \in \mathbb{F}_p$ and $P \in M_{48662,1}(\mathbb{F}_{p^2})$ such that $\chi_0(P) = x$, Curve25519\_Fp(n, x) computes $\chi_0(n \cdot P)$.

which is formalized in Coq as:

\begin{verbatim}
Theorem curve25519_Fp2_ladder_ok:
  forall (n : nat) (x: \mathbb{F}_{2^{255}-19}),
  (n < 2^{255})%nat ->
  forall (p : mc curve25519_Fp2_mcuType),
  p #x0 = Zmodp2.Zmodp2 x 0 ->
  curve25519_Fp_ladder n x = (p ++ n)#x0 /p.
Qed.
\end{verbatim}
RFC is correct

The implementation of X25519 in TweetNaCl computes the $\mathbb{F}_p$-restricted $x$-coordinate scalar multiplication on \( E(\mathbb{F}_{p^2}) \) where $p$ is \( 2^{255} - 19 \) and $E$ is the elliptic curve \( y^2 = x^3 + 486662x^2 + x \).

Theorem RFC_Correct: \( \forall (n \ p : \text{list} \ Z) \)
\( \forall (P: \text{mc curve25519}_{\mathbb{F}_{p^2}-\text{mcuType}}), \)
\( \text{Zlength } n = 32 \rightarrow \)
\( \text{Zlength } p = 32 \rightarrow \)
\( \forall (\lambda x \Rightarrow 0 \leq x \land x < 2 \uparrow 8) \) \( n \rightarrow \)
\( \forall (\lambda x \Rightarrow 0 \leq x \land x < 2 \uparrow 8) \) \( p \rightarrow \)
\( \mathbb{F}_{p^2}x \) \( (\text{decodeUCoordinate } p) = P \# x0 \rightarrow \)
\( \text{RFC } n \ p = \)
\( \text{encodeUCoordinate} \)
\( (((P *+ (Z.\text{to_nat} (\text{decodeScalar25519} n)))) \_x0). \)

Qed.
Thank you.
Equivalences
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; (* Subtraction 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 2\textsuperscript{nd} or 3\textsuperscript{rd} argument depending of Z *)
  Getbit: Z → T' → Z; (* Return the i\textsuperscript{th} 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)
}.
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) := (Sel125519 r a b, Sel125519 r b a) in
  let (c, d) := (Sel125519 r c d, Sel125519 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 e := A a c 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) := (Sel125519 r a b, Sel125519 r b a) in
  let (c, d) := (Sel125519 r c d, Sel125519 r d c) in
  montgomery_rec n z a b c d e f x
end.
Class Ops_Mod_P \{T T' U: Type\} 
\{Mod: U \to U\} \{ModT: T \to T\} 
\{Ops T T' ModT\} \{Ops U U Mod\} := 
\
\{ 
\P: T \to U; (* Projection from T to U *) 
\P': T' \to U; (* Projection from T' to U *) 
\A_eq: \forall a b, \text{Mod} (\P (\A a b)) = \text{Mod} (\A (\P a) (\P b)); 
\M_eq: \forall a b, \text{Mod} (\P (\M a b)) = \text{Mod} (\M (\P a) (\P b)); 
\Zub_eq: \forall a b, \text{Mod} (\P (\Zub a b)) = \text{Mod} (\Zub (\P a) (\P b)); 
\Sq_eq: \forall a, \text{Mod} (\P (\Sq a)) = \text{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, \text{Mod} (\P (\Sel25519 b p q)) = \text{Mod} (\Sel25519 b (\P p) (\P q)); 
\Getbit_eq: \forall i p, \text{Getbit} i p = \text{Getbit} i (\P' p); 
\}.
Generic Montgomery Equivalence

Context \{T: Type\}.
Context \{T': Type\}.
Context \{U: Type\}.
Context \{ModT: T \rightarrow T\}.
Context \{Mod: U \rightarrow 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))) =
Mod (get_a (montgomery_rec n (P z) (P a) (P b) (P c) (P d) (P e) (P f) (P x))). (* over T *)
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))) =
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.
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 $F_{2^{255} - 19}$ nat id := {}.

(* Equivalence between Z (with modP) and Z *)
Instance Zmod_Z_Eq : @Ops_Mod_P Z Z Z modP Z 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 $F_{2^{255} - 19}$ nat id 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 ($\lambda$ x ⇒ 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)); }.

(* 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

curve25519_Fp_ladder n xp \rightarrow \text{defined with} \rightarrow \text{Ops } F_{255-19}^N \text{id}

\text{Ops } \mathbb{Z}(\text{with modP}) \mathbb{Z}(\text{with modP}) \text{ modP}

\text{Z25519_Z_Eq}

\text{Ops } \mathbb{Z} \mathbb{Z} \text{ modP}

\text{Zmod_Z_Eq}

\text{Ops } @\text{List16 } Z \text{ List32B id}

\text{List16_Z_Eq}

\text{Ops } @\text{List16 } Z \text{ List32B id}

\text{List16_List_Z_Eq}

\text{RFC n p}

\text{ZCrypto_Scalarmult } (\text{ZofList n}) (\text{ZofList p}) \rightarrow \text{defined with} \rightarrow Z\text{Crypto_Scalarmult } (\text{ZofList n}) (\text{ZofList p})

\text{uses}

\text{uses}

\text{uses}

\text{Tweetnacl.v (output of clightgen)}