Radboud University

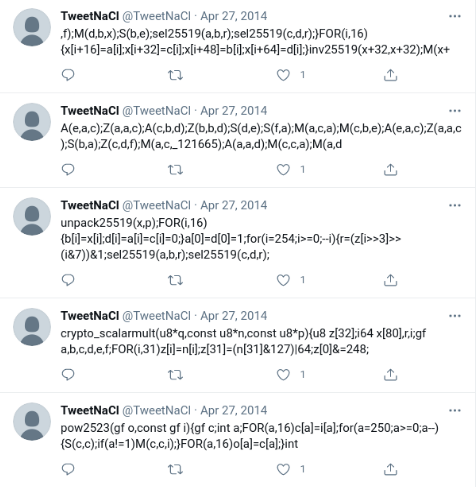
A Coq proof of the correctness of X25519 in TweetNaCl

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

34th IEEE Computer Security Foundations Symposium
June 24th, 2021

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

What is TweetNaCl?



TweetNaCl @TweetNaCl · Apr 27, 2014

```
,f);M(d,b,x);S(b,e);sel25519(a,b,r);sel25519(c,d,r);}FOR(i,16)
{x[i+16]=a[i];x[i+32]=c[i];x[i+48]=b[i];x[i+64]=d[i];}inv25519(x+32,x+32);M(x+
```

1

TweetNaCl @TweetNaCl · Apr 27, 2014

```
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
```

1

TweetNaCl @TweetNaCl · Apr 27, 2014

```
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;sel25519(a,b,r);sel25519(c,d,r);
```

1

TweetNaCl @TweetNaCl · Apr 27, 2014

```
crypto_scalarmult(u8*q,const u8*n,const u8*p){u8 z[32];i64 x[80],r,i;gf
a,b,c,d,e,f;FOR(i,31)z[i]=n[i];z[31]=(n[31]&127)|64;z[0]&=248;
```

1

TweetNaCl @TweetNaCl · Apr 27, 2014

```
pow2523(gf o,const gf i){gf c;int a;FOR(a,16)c[a]=i[a];for(a=250;a>=0;a--)
{S(c,c);if(a!=1)M(c,c,i);}FOR(a,16)o[a]=c[a];}int
```

1

- Small NaCl cryptographic library.
- 100 tweets (of 140 chars.)
- *Easily* auditable.

What is TweetNaCl?

```
int crypto_scalarmult(u8 *q,const u8 *n,const u8 *p)
{
  u8 z[32];
  int r,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;
  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;
    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;
}
```

- Small NaCl cryptographic library.
- 100 tweets (of 140 chars.)
- *Easily* auditable.

X22519: Diffie-Hellman-Merkle key exchange over Curve25519

Curve25519: new Diffie-Hellman speed records

Daniel J. Bernstein *

djb@cr.yp.to

Abstract. This paper explains the design and implementation of a high-security elliptic-curve-Diffie-Hellman function achieving record-setting speeds: e.g., 832457 Pentium III cycles (with several side benefits: free key compression, free key validation, and state-of-the-art timing-attack protection), more than twice as fast as other authors' results at the same conjectured security level (with or without the side benefits).

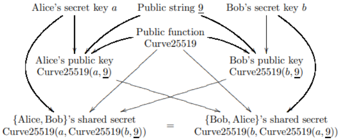
Keywords: Diffie-Hellman, elliptic curves, point multiplication, new curve, new software, high conjectured security, high speed, constant time, short keys

1 Introduction

This paper introduces and analyzes Curve25519, a state-of-the-art elliptic-curve-Diffie-Hellman function suitable for a wide variety of cryptographic applications. This paper uses Curve25519 to obtain new speed records for high-security Diffie-Hellman computations.

Here is the high-level view of Curve25519: Each Curve25519 user has a 32-byte secret key and a 32-byte public key. Each set of two Curve25519 users has a 32-byte shared secret used to authenticate and encrypt messages between the two users.

Medium-level view: The following picture shows the data flow from secret keys through public keys to a shared secret.

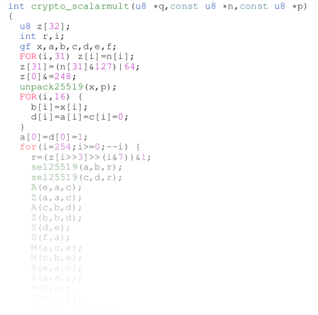


Implementation guidelines in RFC 7748

```
Langley, et al.              Informational                      [Page 8]

RFC 7748              Elliptic Curves for Security          January 2016

   x_1 = u
   x_2 = 1
   z_2 = 0
   x_3 = u
   z_3 = 1
   swap = 0

   For t = bits-1 down to 0:
       k_t = (k >> t) & 1
       swap ^= k_t
       // Conditional swap; see text below.
       (x_2, x_3) = cswap(swap, x_2, x_3)
       (z_2, z_3) = cswap(swap, z_2, z_3)
       swap = k_t

       A = x_2 + z_2
       AA = A^2
       B = x_2 - z_2
       BB = B^2
       E = AA - BB
       C = x_3 + z_3
       D = x_3 - z_3
       DA = D * A
       CB = C * B
       x_3 = (DA + CB)^2
       z_3 = x_1 * (DA - CB)^2
       x_2 = AA * BB
       z_2 = E * (AA + a24 * E)

   // Conditional swap; see text below.
   (x_2, x_3) = cswap(swap, x_2, x_3)
   (z_2, z_3) = cswap(swap, z_2, z_3)
   Return x_2 * (z_2^(p - 2))
```

Overview

TweetNaCl.c

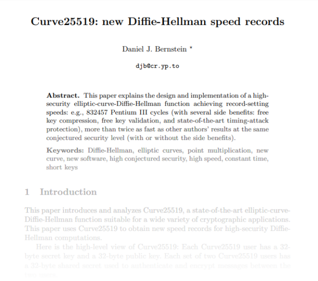


```
int crypto_scalarmult(u8 *q,const u8 *n,const u8 *p)
{
  u8 z[32];
  int r,i;
  gf x,a,b,c,d,e,f;
  FOR(i,31) z[i]=n[i];
  z[31]=(n[31]&127)|64;
  z[0]&=248;
  unpack25519(x,p);
  FOR(i,16) {
    b[i]=x[i];
    d[i]=a[i]=c[i]=0;
  }
  a[0]=d[0]=1;
  for(i=254;i>=0;--i) {
    r=(z[i>>3]>>(i&7))&1;
    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);
```

RFC 7748

```
Langley, et al.              Informational                  [Page 8]

RFC 7748               Elliptic Curves for Security         January 2016

   x_1 = u
   x_2 = 1
   z_2 = 0
   x_3 = u
   z_3 = 1
   swap = 0

   For t = bits-1 down to 0:
       k_t = (k >> t) & 1
       swap ^= k_t
       // Conditional swap; see text below.
       (x_2, x_3) = cswap(swap, x_2, x_3)
       (z_2, z_3) = cswap(swap, z_2, z_3)
       swap = k_t

       A = x_2 + z_2
       AA = A^2
       B = x_2 - z_2
       BB = B^2
       E = AA - BB
       C = x_3 + z_3
       D = x_3 - z_3
       DA = D * A
       CB = C * B
       x_3 = (DA + CB)^2
       z_3 = x_1 * (DA - CB)^2
       x_2 = AA * BB
       z_2 = E * (AA + a24 * E)

   // Conditional swap; see text below.
   (x_2, x_3) = cswap(swap, x_2, x_3)
   (z_2, z_3) = cswap(swap, z_2, z_3)
   Return x_2 * (z_2^(p - 2))
```

Maths



Curve25519: new Diffie-Hellman speed records

Daniel J. Bernstein *

djb@cr.yp.to

Abstract. This paper explains the design and implementation of a high-security elliptic-curve-Diffie-Hellman function achieving record-setting speeds: e.g., 832457 Pentium III cycles (with several side benefits: free key compression, free key validation, and state-of-the-art timing-attack protection), more than twice as fast as other authors' results at the same conjectured security level (with or without the side benefits).

Keywords: Diffie-Hellman, elliptic curves, point multiplication, new curve, new software, high conjectured security, high speed, constant time, short keys

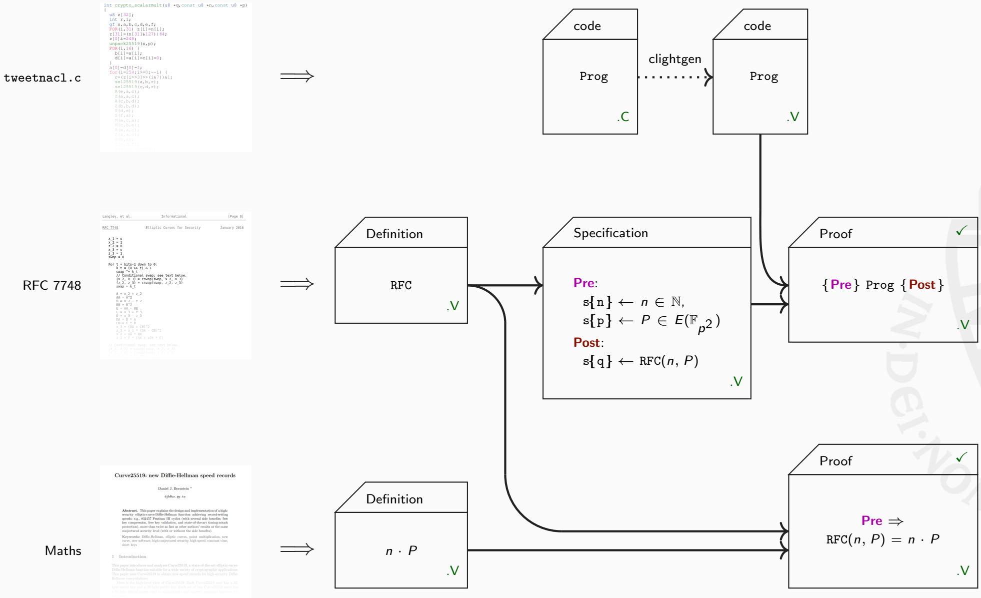
1 Introduction

This paper introduces and analyzes Curve25519, a state-of-the-art elliptic-curve-Diffie-Hellman function suitable for a wide variety of cryptographic applications. This paper uses Curve25519 to obtain new speed records for high-security Diffie-Hellman computations.

Here is the high-level view of Curve25519: Each Curve25519 user has a 32-byte secret key and a 32-byte public key. Each set of two Curve25519 users has a 32-byte shared secret used to authenticate and encrypt messages between the two users.

- We formalize RFC 7748 in Coq.
- We prove that TweetNaCl correctly implements RFC 7748.
- We prove that RFC 7748 matches X25519.

Overview



tweetnacl.c
RFC 7748
Maths
Curve25519: new Diffie-Hellman speed records
code
Prog
.C
clightgen
code
Prog
.V
Definition
RFC
.V
Specification
Pre:
s{n} ← n ∈ ℕ,
s{p} ← P ∈ E(𝔽p2)
Post:
s{q} ← RFC(n, P)
.V
Proof ✓
{Pre} Prog {Post}
.V
Definition
n · P
.V
Proof ✓
Pre ⇒
RFC(n, P) = n · P
.V

Formalizing X25519 from RFC 7748

RFC 7748 in Coq

The specification of X25519 in RFC 7748 is formalized by `RFC` in Coq.

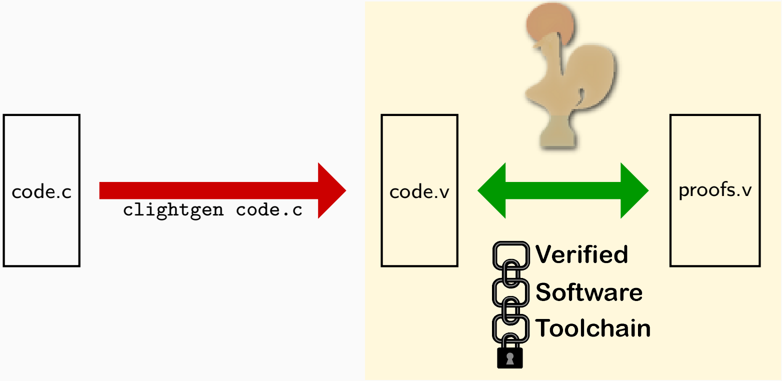
More formally:

```
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    (* x2                *)
    u    (* x3                *)
    0    (* z2                *)
    1    (* z3                *)
    0    (* dummy             *)
    0    (* dummy             *)
    u    (* x1                *) 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: x_2   b: x_3   c: z_2   d: z_3   x: x_1   *)
(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                 (* k_t = (k >> t) & 1            *)
    (* swap <- k_t *)
  let (a, b) := (Sel25519 r a b, Sel25519 r b a) in (* (x_2, x_3) = cswap(swap, x_2, x_3) *)
  let (c, d) := (Sel25519 r c d, Sel25519 r d c) in (* (z_2, z_3) = cswap(swap, z_2, z_3) *)
  let e := a + c in                                  (* A = x_2 + z_2                  *)
  let a := a - c in                                  (* B = x_2 - z_2                  *)
  let c := b + d in                                  (* C = x_3 + z_3                  *)
  let b := b - d in                                  (* D = x_3 - z_3                  *)
  let d := e ^2 in                                   (* AA = A^2                       *)
  let f := a ^2 in                                   (* BB = B^2                       *)
  let a := c * a in                                  (* CB = C * B                     *)
  let c := b * e in                                  (* DA = D * A                     *)
  let e := a + c in                                  (* x_3 = (DA + CB)^2              *)
  let a := a - c in                                  (* z_3 = x_1 * (DA - CB)^2        *)
  let b := a ^2 in                                   (* z_3 = x_1 * (DA - CB)^2        *)
  let c := d - f in                                  (* E = AA - BB                    *)
  let a := c * C_121665 in                           (* z_2 = E * (AA + a24 * E)       *)
  let a := a + d in                                  (* z_2 = E * (AA + a24 * E)       *)
  let c := c * a in                                  (* z_2 = E * (AA + a24 * E)       *)
  let a := d * f in                                  (* x_2 = AA * BB                  *)
  let d := b * x in                                  (* z_3 = x_1 * (DA - CB)^2        *)
  let b := e ^2 in                                   (* x_3 = (DA + CB)^2              *)
  let (a, b) := (Sel25519 r a b, Sel25519 r b a) in (* (x_2, x_3) = cswap(swap, x_2, x_3) *)
  let (c, d) := (Sel25519 r c d, Sel25519 r d c) in (* (z_2, z_3) = cswap(swap, z_2, z_3) *)
  montgomery_rec n z a b c d e f x
end.
```

From C to Coq

Proving with VST



Hoare Triple of crypto_scalarmult

```
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) (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
```

TweetNaCl implements correctly the RFC

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
   (* Global variables used in the code. *)
   Vprog
   (* Hoare triples for function calls. *)
   Gprog
   (* Clight AST of the 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

```
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

Definition

Let $a \in \mathbb{K}\backslash\{-2, 2\}$, *and* $b \in \mathbb{K}\backslash\{0\}$. *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* $\mathcal{O}$, *"at infinity".*

```
(* 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) :=
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.
```

```
Definition  neg (p: point 𝕂) :=
  if p is (| x, y |) then (| x, − y |) else ∞.


Definition  add (p1 p2: point 𝕂) :=
    match p1 , p2 with
    | ∞, _ ⇒ p2                                    (* If one point is infinity *)
    | _, ∞ ⇒ p1                                    (* If one point is infinity *)

    | (| x1, y1 |), (| x2, y2 |) ⇒
      if x1 == x2 then
        if (y1 == y2) && (y1 ≠ 0) then ...                     (* If p1 = p2 *)
        else
          ∞
      else                                                     (* If p1 ≠ p2 *)
        let s := (y2 − y1) / (x2 − x1) in
        let xs := s^2 ∗ B − A − x1 − x2 in
         (| xs, −s ∗ (xs − x1) − y1|)
    end


Notation "− x" := (neg x).
Notation "x + y" := (add x y).
Notation "x − y" := (x + (− y)).
```

Correctness of the Montgomery ladder

Hypothesis

$a^2 - 4$ is not a square in $\mathbb{K}$.

We prove its correctness.

Theorem

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

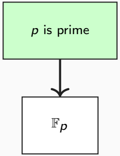
```
Theorem montgomery_ladder_ok (n m: nat) (x : K) :
  n < 2^m →
  forall (p : mc M), p#x0 = x
  (* if x is the x-coordinate of P *)
  → montgomery_ladder n m x = (p *+ n)#x0.
  (* montgomery_ladder n m xp is the x-coordinate of n · P *).
Qed.
```

Proof of correctness of X25519

p is prime

$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$

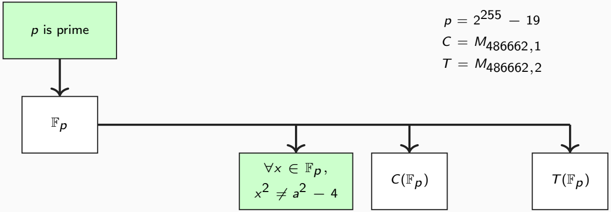
Proof of correctness of X25519



$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$

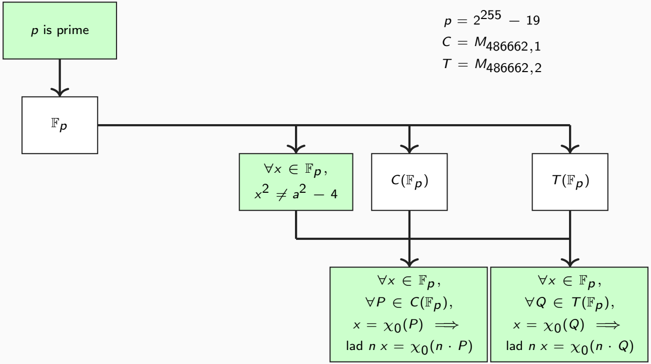
Proof of correctness of X25519

$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$



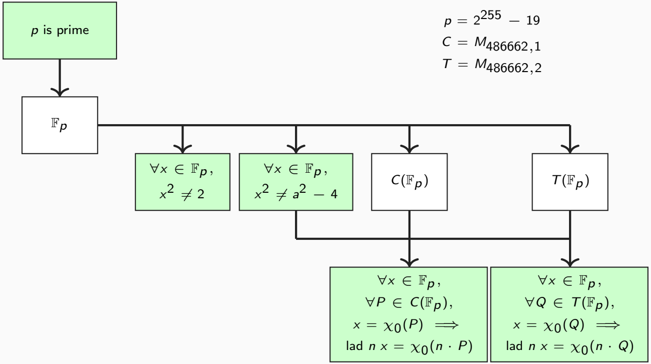
Proof of correctness of X25519

$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$



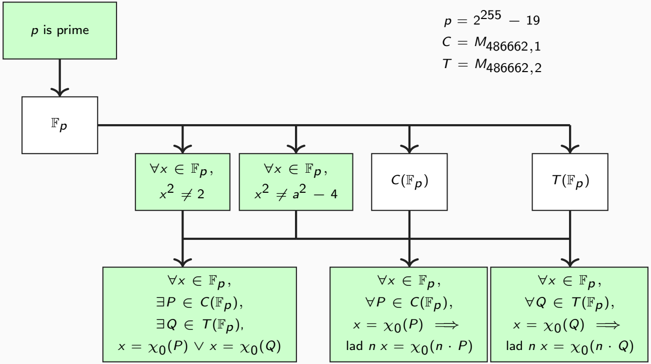
Proof of correctness of X25519

$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$



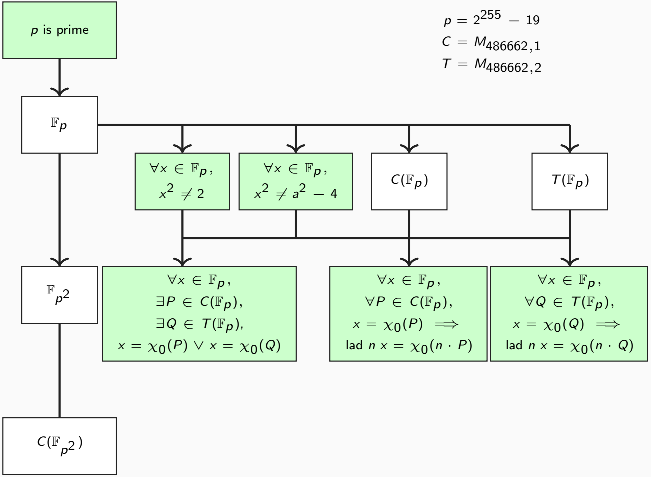
Proof of correctness of X25519

$p = 2^{255} - 19$
$C = M_{486662,1}$
$T = M_{486662,2}$



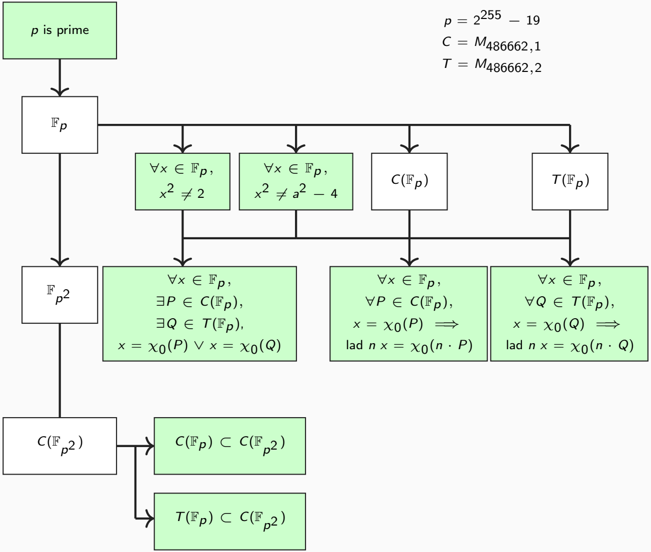
Proof of correctness of X25519

$p = 2^{255} - 19$
$C = M_{486662,1}$
$T = M_{486662,2}$



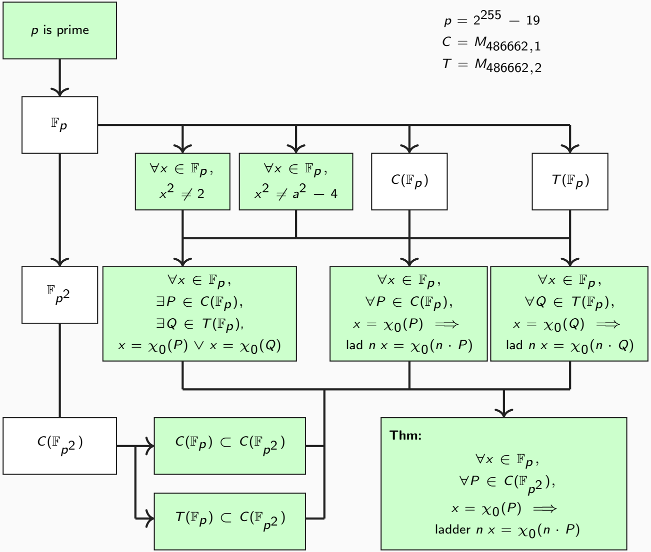
Proof of correctness of X25519

$$p = 2^{255} - 19$$
$$C = M_{486662,1}$$
$$T = M_{486662,2}$$



Proof of correctness of X25519

$p = 2^{255} - 19$
$C = M_{486662,1}$
$T = M_{486662,2}$



Theorem

For all $n \in \mathbb{N}$, such that $n < 2^{255}$, for all $x \in \mathbb{F}_p$ and $P \in M_{486662,1}(\mathbb{F}_{p^2})$ such that $\chi_0(P) = x$, Curve25519_Fp(n, x) computes $\chi_0(n \cdot P)$.

which is formalized in Coq as:

```
Theorem curve25519_Fp2_ladder_ok:
  forall (n : nat) (x: 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.
```

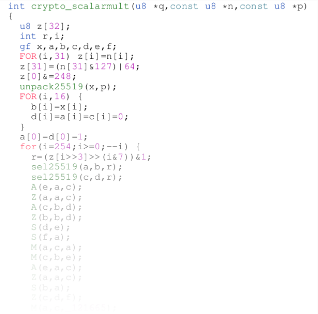
RFC is correct

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 : list Z)
  (P:mc curve25519_Fp2_mcuType),
  Zlength n = 32 →
  Zlength p = 32 →
  Forall (λ x ⇒ 0 ≤ x ∧ x < 2 ^ 8) n →
  Forall (λ x ⇒ 0 ≤ x ∧ x < 2 ^ 8) p →
  Fp2_x (decodeUCoordinate p) = P#x0 →
  RFC n p =
    encodeUCoordinate
      ((P *+ (Z.to_nat (decodeScalar25519 n))) _x0).
Qed.
```

Conclusion

TweetNaCl.c



```c
int crypto_scalarmult(u8 *q,const u8 *n,const u8 *p)
{
  u8 z[32];
  int r,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;
  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;
    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);
```

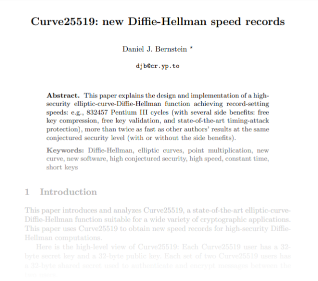
RFC 7748

Langley, et al. Informational [Page 8]

RFC 7748 Elliptic Curves for Security January 2016

```
x_1 = u
x_2 = 1
z_2 = 0
x_3 = u
z_3 = 1
swap = 0

For t = bits-1 down to 0:
    k_t = (k >> t) & 1
    swap ^= k_t
    // Conditional swap; see text below.
    (x_2, x_3) = cswap(swap, x_2, x_3)
    (z_2, z_3) = cswap(swap, z_2, z_3)
    swap = k_t

    A = x_2 + z_2
    AA = A^2
    B = x_2 - z_2
    BB = B^2
    E = AA - BB
    C = x_3 + z_3
    D = x_3 - z_3
    DA = D * A
    CB = C * B
    x_3 = (DA + CB)^2
    z_3 = x_1 * (DA - CB)^2
    x_2 = AA * BB
    z_2 = E * (AA + a24 * E)

// Conditional swap; see text below.
(x_2, x_3) = cswap(swap, x_2, x_3)
(z_2, z_3) = cswap(swap, z_2, z_3)
```

Maths



Curve25519: new Diffie-Hellman speed records

Daniel J. Bernstein *

djb@cr.yp.to

Abstract. This paper explains the design and implementation of a high-security elliptic-curve-Diffie-Hellman function achieving record-setting speeds: e.g., 832457 Pentium III cycles (with several side benefits: free key compression, free key validation, and state-of-the-art timing-attack protection), more than twice as fast as other authors' results at the same conjectured security level (with or without the side benefits).

Keywords: Diffie-Hellman, elliptic curves, point multiplication, new curve, new software, high conjectured security, high speed, constant time, short keys

1 Introduction

This paper introduces and analyzes Curve25519, a state-of-the-art elliptic-curve-Diffie-Hellman function suitable for a wide variety of cryptographic applications. This paper uses Curve25519 to obtain new speed records for high-security Diffie-Hellman computations.

Here is the high-level view of Curve25519: Each Curve25519 user has a 32-byte secret key and a 32-byte public key. Each set of two Curve25519 users has a 32-byte shared secret used to authenticate and encrypt messages between the two users.

- We formalized RFC 7748 in Coq.
- We proved that TweetNaCl correctly implements RFC 7748.
- We proved that RFC 7748 matches X25519 up to the theory of Elliptic Curves.

Thank you.