Radbound University

Curve25519: Proving datatypes with a rooster

Benoît Viguier MSc

(λ x y. x@y.nl) benoit viguier

https://www.viguier.nl

DS Lunch talk

9th December 2016

Institute for Computing and Information Sciences – Digital Security

Radboud University Nijmegen

Overview

A quick overview of TweetNaCl

From C to Coq

car25519

A quick overview of TweetNaCl

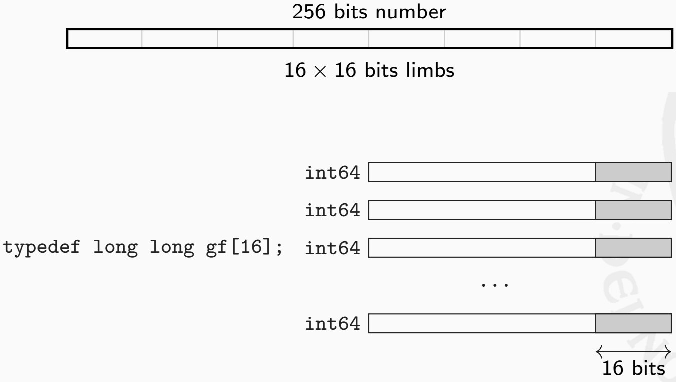
Context

```
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);                    #  The steps and order
  S(f,a);                    #  of the operations
  M(a,c,a);                  #  have been proved
  M(c,b,e);                  #  by Timmy Weerwag
  A(e,a,c);                  #
  Z(a,a,c);                  #
  S(b,a);                    #  The use of datatypes
  Z(c,d,f);                  #  (number representation)
  M(a,c,_121665);            #  is not proven (yet).
  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);
}
```

Code 1: crypto_scalarmult

Datatype (or number representation)

256 bits integers does not fit into a 64 bits containers…



```
#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
{
  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
}
```

Code 2: Basic Operations

What needs to be done

We need to prove:

(1) that the operations (A,Z,M) are what they are supposed to be with repect to the number representation

(2) that the operations (A,Z,M) are correct in $GF(2^{255} - 19)$.

(3) the absence of possible [over/under]flows.

What needs to be done

We need to prove:

(1) that the operations (A,Z,M) are what they are supposed to be with repect to the number representation

(2) that the operations (A,Z,M) are correct in $GF(2^{255} - 19)$.

(3) the absence of possible [over/under]flows.

Is this enough ?

What needs to be done

We need to prove:

(1) that the operations (A,Z,M) are what they are supposed to be with repect to the number representation

(2) that the operations (A,Z,M) are correct in $GF(2^{255} - 19)$.

(3) the absence of possible [over/under]flows.

Is this enough ?

No! We also need to prove the soundness of `car25519`.

```
sv car25519(gf o)
{
  int i;
  i64 c;
  FOR(i,16) {
    o[i]+=(1LL<<16);
    c=o[i]>>16;
    o[(i+1)*(i<15)]+=c-1+37*(c-1)*(i==15);
    o[i]-=c<<16;
  }
}

# unpacked version:
sv car25519(gf o)
{
  int i;
  i64 c;
  FOR(i,15) {
    o[i]+=(1LL<<16);    # add 2^16
    c=o[i]>>16;         # get the carry (bits > 16)
    o[(i+1)]+=c-1;      # propagate to the next limb
    o[i]-=c<<16;        # remove the carry
  }
  o[15]+=(1LL<<16);     # add 2^16
  c=o[15]>>16;          # get the carry (bits > 16)
  o[0]+=38*(c-1);       # propagate to the first limb
  o[15]-=c<<16;         # remove the carry
}
```

Code 3: car25519

```
sv pack25519(u8 *o,const gf n)
{
  int i,j,b;
  gf m,t;
  FOR(i,16) t[i]=n[i];
  car25519(t);
  car25519(t);
  car25519(t);
  ...
```

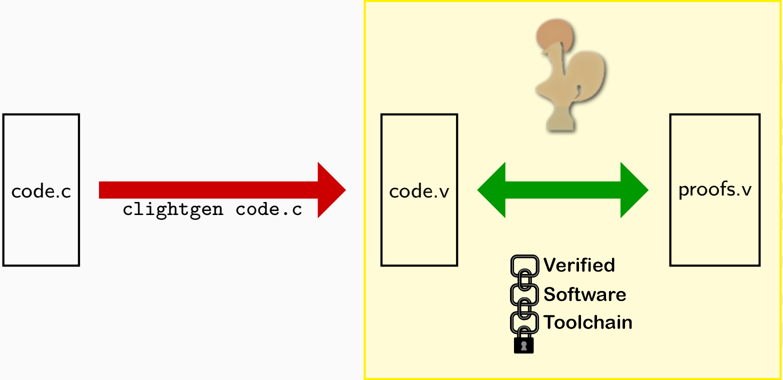
Code 4: car25519 in use

We need to prove that after 3 iterations of `car25519`,

all the values in t are below 2^{16}.

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 convert a list ℤ into it's 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.lst A" := (ZofList A) (at level 65).
```

Code 5: ZofList

Addition

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

Corollary ZsumList_correct:
  ∀ (a b: list ℤ),
    (ℤ.lst a ⊞ b) = (ℤ.lst a) + (ℤ.lst b).
Qed.

Lemma ZsumList_bound_len:
  ∀ (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.
```

Code 6: Addition

Multiplication - specification

```
sv M(gf o,const gf a,const gf b)    # Multiplication
{
  FOR(i,16) FOR(j,16) t[i+j] = a[i]*b[j];   # mult_1
  FOR(i,15) t[i]+=38*t[i+16];               # mult_2
  FOR(i,16) o[i]=t[i];                      # mult_3
 }
```

Code 7: M

```
Fixpoint ZscalarMult (a: ℤ) (b: list ℤ) : list ℤ := match b with
| [] ⇒ []
| h :: q ⇒  a * h :: ZscalarMult a q
end.
Notation "A ∘ B" := (ZscalarMult A B) (at level 60).

Fixpoint mult_1 (a b:list ℤ) : list ℤ := match a, b with
| [],_ ⇒  []
| _,[] ⇒  []
| ha :: qa, hb :: qb ⇒  ha * hb :: (ha ∘ qb) ⊞ (mult_1 qa (hb::qb))
end.

Definition mult_2 (a:list ℤ) : list ℤ := a  ⊞ (38 ∘ (tail 16 a)).
(* where "tail n a" drop the n first elements of a *)

Definition mult_3 (a:list ℤ) : list ℤ := slice 16 a.
(* where "slice n a" keep the n first elements of a *)

Definition M (a b:list ℤ) : list ℤ := mult_3 (mult_2 (mult_1 a b)).
```

Code 8: Multiplication

Multiplication - correctness

```
Notation "A :𝒢ℱ" := (A mod (2^255 - 19)) (at level 40).

Corollary mult1_correct :
  ∀ (a b: list ℤ),
    ℤ.lst mult_1 a b =  (ℤ.lst a) * (ℤ.lst b).
Qed.

Lemma mult_2_correct :
  ∀ (l: list ℤ),
    (ℤ.lst mult_2 l) = (ℤ.lst l) + 38 * ℤ.lst tail 16 l.
Qed.

Lemma reduce_slice_GF:
  ∀ (l: list ℤ),
    ℤ.ℕ length l < 32 →
      (ℤ.lst mult_3 (mult_2 l)) :𝒢ℱ = (ℤ.lst l) :𝒢ℱ.
Qed.

Corollary mult_GF:
  ∀ (a b: list ℤ),
    ℤ.ℕ length a = 16 →
    ℤ.ℕ length b = 16 →
      (ℤ.lst M a b) :𝒢ℱ = (ℤ.lst a) * (ℤ.lst b) :𝒢ℱ.
Qed.
```

Code 9: Multiplication — proof of correctness

```
Lemma ZscalarMult_bound_const:
  ∀ (m2 n2 a: ℤ) (b: list ℤ),
    0 < a →
    Forall (λ x ⇒  m2 < x < n2) b →
    Forall (λ x ⇒  a * m2 < x < a * n2) (a ∘ b).
Qed.

Lemma mult_1_bound:
  ∀ (m1 n1 m2 n2 m3 n3: ℤ) (a b: list ℤ),
    (λ x ⇒  m1 < x < n1) 0 →
    (λ x ⇒  m2 < x < n2) 0 →
    Forall (λ x ⇒  m1 < x < n1) a →
    Forall (λ x ⇒  m2 < x < n2) b →
    m3 = Zmin (Zlength a) (Zlength b) * min_prod m1 n1 m2 n2 →
    n3 = Zmin (Zlength a) (Zlength b) * max_prod m1 n1 m2 n2 →
      Forall (λ x ⇒  m3 < x < n3) (mult_1 a b).
Admitted.

Lemma mult_2_bound:
  ∀ (m1 n1: ℤ) (a: list ℤ),
    (λ x ⇒  m1 < x < n1) 0 →
    Forall (λ x ⇒  m1 < x < n1) a →
      Forall (λ x ⇒  m1 + 38 * m1 < x < n1 + 38 * n1) (mult_2 a).
Qed.

Lemma mult_3_bound:
  ∀ (m1 n1: ℤ) (a: list ℤ),
    Forall (λ x ⇒  m1 < x < n1) a →
      Forall (λ x ⇒  m1 < x < n1) (mult_3 a).
Qed.
```

Code 10: Multiplication — Proofs of bounds

Multiplication - bounds

```
Lemma mult_bound:
  ∀ (m1 n1 m2 n2 m3 n3: Z) (a b: list Z),
    (λ x ⇒ m1 < x < n1) 0 →
    (λ x ⇒ m2 < x < n2) 0 →
    Forall (λ x ⇒ m1 < x < n1) a →
    Forall (λ x ⇒ m2 < x < n2) b →
    m3 = 39 * Z.min (Zlength a) (Zlength b) * min_prod m1 n1 m2 n2 →
    n3 = 39 * Z.min (Zlength a) (Zlength b) * max_prod m1 n1 m2 n2 →
      Forall (λ x ⇒ m3 < x < n3) (M a b).
Admitted.
```

Code 11: M — Proofs of bounds

What can we deduce from this ?

$$
\begin{aligned}
39 \times 16 \times (2^x)^2 &< 64 \times 16 \times (2^x)^2 \\
64 \times 16 \times (2^x)^2 &< 2^{62} \\
2^6 \times 2^4 \times (2^x)^2 &< 2^{62} \\
x &< 26
\end{aligned}
$$

Thus we will avoid any over/underflows if the inputs are within the $]-2^{26}, 2^{26}[$ ranges:

```
Lemma mult_bound_strong:
  ∀ (a b: list Z),
    (length a = 16)%ℕ →
    (length b = 16)%ℕ →
    Forall (λ x ⇒  -2^26 < x < 2^26) a →
    Forall (λ x ⇒  -2^26 < x < 2^26) b →
      Forall (λ x ⇒  -2^62 < x < 2^62) (M a b).
Admitted.
```

Code 12: M — Proofs of bounds

car25519

```
FOR(i,15) {
  o[i]+=(1LL<<16);    # add 2^16
  c=o[i]>>16;         # get the carry (bits > 16)
  o[(i+1)]+=c-1;      # propagate to the next limb
  o[i]-=c<<16;        # remove the carry
}
```

Code 13: car25519 — propagation

```
(*
  getCarry n m       is equivalent to    m >> n
  getResidute n m    is equivalent to    m mod 2^n
*)

Fixpoint Carrying_n (p:ℕ) (a:ℤ) (l:list ℤ) : list ℤ := match p,a,l with
| _,  0,[]      ⇒  []
| _,  a,[]      ⇒  [a]
| 0%ℕ,  a,h::q ⇒  (a + h) :: q
| S p,a,h :: q ⇒  getResidute n (a + h) :: Carrying_n p (getCarry n (a + h)) q
end.

Corollary CarrynPreserve:
  ∀ (m: ℕ) (l: list ℤ),
    ℤ.lst l = ℤ.lst Carrying_n m 0 l.
Qed.
```

Code 14: car25519 — Proofs of correctness

Remark: the add 2^{16} step has been ignored.

car25519 - specification

```
o[15]+=(1LL<<16);     # add 2^16
c=o[15]>>16;          # get the carry (bits > 16)
o[0]+=38*(c-1);       # propagate to the first limb
o[15]-=c<<16;         # remove the carry
```

Code 15: car25519 — back

```
Definition backCarry (l:list Z) : (list Z) :=
  match l with
  | [] ⇒ []
  | h :: q ⇒ let v := nth 15 l 0 in
            (h + 38 * getCarry 16 v) :: slice 14 q ++ [getResidute 16 v]
  end.

Lemma backCarry_25519:
  ∀ (l:list ℤ),
    (length l ≤ 16)%ℕ →
      (ℤ.lst l) :𝒢ℱ = ((ℤ.lst backCarry l) :𝒢ℱ).
Qed.
```

Code 16: car25519 — Proofs of correctness

```
Definition car25519 (l:list ℤ) : list ℤ := backCarry (Carrying_n 16 15 0 l).

Lemma car25519_correct:
  ∀ (l: list ℤ),
    (length l = 16)%ℕ →
      (ℤ.lst l) :𝒢ℱ  = (ℤ.lst car25519 l) :𝒢ℱ.
Qed.

Lemma car25519_bound :
  ∀ (i: ℕ) (l: list ℤ),
    (length l = 16)%ℕ →
    (i ≠  0)%ℕ →
      nth i (car25519 l) 0 < 2 ^ 16.
Qed.
```

Code 17: car25519 — Proofs of correctness

```
Lemma t2256is38 : (2^256 :𝒢ℱ) = (38 :𝒢ℱ).
Proof. compute. reflexivity. Qed.

Definition Zcar25519 (n: ℤ) : ℤ  :=  38 * getCarry 256 n +  getResidute 256 n.

Lemma Zcar25519_correct:
  ∀ (n: ℤ), n :𝒢ℱ = (Zcar25519 n) :𝒢ℱ.
Qed.

Lemma Zcar25519_eq_car25519:
  ∀ (l : list ℤ),
  (length l = 16)%ℕ →
    Zcar25519 (ℤ.lst l) = ℤ.lst (car25519 l).
Qed.
```

Code 18: car25519

car25519 - bounds

```
Lemma ZCarry25519_min:
  ∀ (x: ℤ),
    0 < x →
      0 < Zcar25519 x.
Qed.

Lemma ZCarry25519_sup_bounds:
  ∀ (x: ℤ),
    x < 2 ^ 302 →
    0 < x →
      Zcar25519 x < 2 ^ 257.
Qed.

Lemma Zcarry25519_fixpoint :
  ∀ (x: ℤ),
  0 < x < 2 ^ 256 →
    Zcar25519 x = x.
Qed.

Theorem doubleCar:
  ∀ (x y: ℤ),
    0 ≤  x < 2 ^ 302 →
    y = Zcar25519 x →
      Zcar25519 y < 2 ^ 256.
Qed.
```

Code 19: car25519

What is left ?

Curent and future work:

- finish the proofs on the bounds for the Multiplication.
- redo all the proofs on the carrying including the addition of 2^{16}.
- Prove that the model matches the semantic (`code.v`) using VST (#Princeton).
- Prove that the steps does not yeld to an over/underflow.

Thank you.