Coppersmith's Method

Katherine Kosaian and Yong Kiam Tan

June 13, 2024

Abstract

We formalize *Coppersmith's method*, an algorithm for finding small (in magnitude) roots of univariate integer polynomials mod M. Coppersmith's method has important applications in cryptography and is used in various attacks on the RSA algorithm for public-key cryptography. We also formalize a related (more lightweight) result with slightly weaker bounds; we split out the generic mathematical results underlying both this lightweight result and Coppersmith's method into a dedicated locale, which could be used to prove other "Coppersmith-like" results. Our work builds on the existing formalization of the LenstraLenstraLovász (LLL) algorithm for lattice basis reduction [2], and our formalization adds a determinant bound on the length of the short vector produced by the LLL algorithm.

There are many resources on Coppersmith's method that we found useful in the course of our development. Some that we recommend include Chapter 19 of a textbook by Galbraith [3], Section 17.3 of a textbook by Trappe and Washington [4], and the following set of lecture notes [1].

Contents

1 Preliminary results for LLL

In this file, we prove some additional results involving lattices and a bound for LLL.

theory *More-LLL*
imports
  LLL-Basis-Reduction.LLL
begin

context *vec-module* **begin**

lemma *in-lattice-ofD*:
  assumes *x*: $x \in$ *lattice-of a*
  assumes *a*: *set a* $\subseteq$ *carrier-vec n*
  obtains *v* **where**
    $v \in$ *carrier-vec* (*length a*)
    mat-of-cols n a $*_v$ *map-vec of-int* (v::*int vec*) = x
⟨*proof*⟩

lemma *exists-list-all2*:
  assumes $\forall x.\ (x \in set\ xs \longrightarrow (\exists y.\ P\ x\ y))$
  obtains *ys* **where** *list-all2 P xs ys*
  ⟨*proof*⟩

lemma *subset-lattice-ofD*:
  assumes *xs*: $set\ xs \subseteq lattice\text{-}of\ a\ set\ xs \subseteq carrier\text{-}vec\ n$
  assumes *a*: $set\ a \subseteq carrier\text{-}vec\ n$
  obtains *vs* **where**
    $vs \in carrier\text{-}mat\ (length\ a)\ (length\ xs)$
    $mat\text{-}of\text{-}cols\ n\ a * map\text{-}mat\ of\text{-}int\ vs = mat\text{-}of\text{-}cols\ n\ xs$
⟨*proof*⟩

lemma *mk-coeff-list-nth*:
  assumes $i < length\ ls\ distinct\ ls$
  shows $mk\text{-}coeff\ ls\ f\ (ls\ !\ i) = f\ i$
  ⟨*proof*⟩

This next lemma is trivial when the cols are not distinct.

lemma *mat-mul-non-zero-col-lin-dep*:
  assumes *A*: $A \in carrier\text{-}mat\ n\ y\ distinct\ (cols\ A)$
  assumes *U*: $U \in carrier\text{-}mat\ y\ z$
  assumes *i*: $i < z\ col\ U\ i \neq 0_v\ y$
  assumes *eqz*: $A * U = 0_m\ n\ z$
  shows $lin\text{-}dep\ (set\ (cols\ A))$
⟨*proof*⟩

lemma *lin-indpt-mul-eq-ident*:
  assumes *a*: $distinct\ a\ lin\text{-}indpt\ (set\ a)$
    $set\ a \subseteq carrier\text{-}vec\ n\ length\ a = m$
  assumes *u*: $u \in carrier\text{-}mat\ m\ m$
  assumes *e*: $mat\text{-}of\text{-}cols\ n\ a = mat\text{-}of\text{-}cols\ n\ a * u$
  shows $u = 1_m\ m$
⟨*proof*⟩

end

Set up a locale: vec_module where the ring has characteristic zero

locale *ring-char-0-vec-module* = *vec-module f-ty n* **for**
  $f\text{-}ty{::}'a{::}\ \{comm\text{-}ring\text{-}1,\ ring\text{-}char\text{-}0\}\ itself$
  and *n*
begin

This next lemma shows that different basis a, b of the same lattice have the same Gram determinant.

lemma *lattice-of-eq-gram-det-eq*:
  fixes $a\ b{::}'a\ vec\ list$
  assumes *a*: $distinct\ a\ lin\text{-}indpt\ (set\ a)\ set\ a \subseteq carrier\text{-}vec\ n$
  assumes *b*: $set\ b \subseteq carrier\text{-}vec\ n\ length\ a = length\ b$

assumes *lat-eq*: *lattice-of a* = *lattice-of b*
defines *A*: *A* ≡ *mat-of-cols n a*
defines *B*: *B* ≡ *mat-of-cols n b*
shows *det* $(A^T * A)$ = *det* $(B^T * B)$
⟨*proof*⟩

lemma *lattice-of-eq-gram-det-rows-eq*:
fixes *a b*::′*a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n*
assumes *b*: *set b* ⊆ *carrier-vec n* *length a* = *length b*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
defines *A*: *A* ≡ *mat-of-rows n a*
defines *B*: *B* ≡ *mat-of-rows n b*
shows *det* $(A * A^T)$ = *det* $(B * B^T)$
⟨*proof*⟩

lemma *lattice-of-eq-sq-det-eq*:
fixes *a b*::′*a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n length a* = *n*
assumes *b*: *set b* ⊆ *carrier-vec n length b* = *n*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
shows (*det* (*mat-of-cols n a*))^*2* = (*det* (*mat-of-cols n b*))^*2*
⟨*proof*⟩

lemma *lattice-of-eq-sq-det-rows-eq*:
fixes *a b*::′*a vec list*
assumes *a*: *distinct a lin-indpt* (*set a*) *set a* ⊆ *carrier-vec n length a* = *n*
assumes *b*: *set b* ⊆ *carrier-vec n length b* = *n*
assumes *lat-eq*: *lattice-of a* = *lattice-of b*
shows (*det* (*mat-of-rows n a*))^*2* = (*det* (*mat-of-rows n b*))^*2*
⟨*proof*⟩

end

context *LLL-with-assms*
begin

This next lemma bounds the size of the shortest vector by the determinant.

lemma *short-vector-det-bound*:
assumes *m*: $m \neq 0$
assumes *k*: $k \leq m$
shows
(*rat-of-int* ‖*short-vector*‖²) ^ *k* ≤
α ^(*k* ∗ (*k*−*1*) *div 2*) ∗ *rat-of-int* (*gs.Gramian-determinant reduce-basis k*)
⟨*proof*⟩

lemma *square-Gramian-determinant-eq-det-square*:
assumes *sq*:*n* = *m*

shows *gs.Gramian-determinant fs-init m* =
(*det* (*mat-of-rows m fs-init*))^*2*
⟨*proof*⟩

end

end

2 Algorithm for Coppersmith's Method

In this file, we formalize the algorithm for Coppersmith's method. We follow the descriptions in Chapter 19 of "Mathematics of Public Key Cryptography" by Steven Galbraith and Section 17.3 of "Introduction to Cryptography with Coding Theory" by Trappe and Washington. We first formalize an algorithm for a "lightweight" version of Coppersmith's method, and then formalize Coppersmith's method itself. Correctness proofs for these algorithms are in "Towards_Coppersmith.thy" and "Coppersmith.thy".

theory *Coppersmith-Algorithm*

imports *LLL-Factorization.LLL-Factorization*
begin

This next definition forms the vector associated to a polynomial as in (19.1) of Galbraith.

definition *vec-associated-to-poly*:: (*'a*::{*ring*,*power*}) *poly* ⇒ *'a* ⇒ *'a vec* **where**
vec-associated-to-poly p X = *vec* (*degree p* + *1*) (λ*j*. (*coeff p j*) ∗ *X*^*j*)

abbreviation *vec-associated-to-int-poly*:: *int poly* ⇒ *int* ⇒ *int vec* **where**
vec-associated-to-int-poly ≡ *vec-associated-to-poly*
abbreviation *vec-associated-to-real-poly*:: *real poly* ⇒ *real* ⇒ *real vec* **where**
vec-associated-to-real-poly ≡ *vec-associated-to-poly*

definition *int-poly-associated-to-vec*:: *int vec* ⇒ *nat* ⇒ *int poly* **where**
int-poly-associated-to-vec v X = (
let newvec = *vec* (*dim-vec v*) (λ*j*. *floor* (((*v* $ *j*)::*real*)/(*X*^*j*))) *in*
∑ *i*<(*dim-vec v*). *monom* (*newvec* $ *i*) *i*
)

2.1 Lightweight method similar to Coppersmith

In this section, we start with a more lightweight version of Coppersmith which doesn't achieve the full bounds, but is built on similar ideas.

This next definition constructs the g_i's

definition *g-i-vec*:: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec* **where**
g-i-vec M X i n = *vec n* (λ*j*. *if j* = *i then M*∗*X*^*i else 0*)

This next definition should be called with degree d = p - 1

definition *form-basis-helper*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list* **where**
form-basis-helper p M X d = *map* (λ*i*. *g-i-vec M X i* (*degree p* + *1*)) [*0*..<*d* + *1*]

definition *form-basis*:: *int poly* => *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list* **where**
form-basis p M X d = (*form-basis-helper p M X d*) @ [*vec-associated-to-int-poly p X*]

definition *first-vector*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *int vec* **where**
first-vector p M X = (
let cs-basis = *form-basis p M X* (*degree p* − *1*) *in*
(*short-vector 2 cs-basis*)
)

definition *towards-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *int poly* **where**
towards-coppersmith p M X = *int-poly-associated-to-vec* (*first-vector p M X*) *X*

2.2 Full Coppersmith

In this section, we develop the full Coppersmith's algorithm.

definition *vec-associated-to-int-poly-padded*:: *nat* ⇒ *int poly* ⇒ *nat* ⇒ *int vec* **where**
vec-associated-to-int-poly-padded n p X = *vec n* (λ*j*. (*coeff p j*)∗*X*^*j*)

definition *row-of-coppersmith-matrix*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec*
where *row-of-coppersmith-matrix p M X h i j* =
vec-associated-to-int-poly-padded ((*degree p*)∗*h*) (*smult* (((*M*^(*h*−*1*−*j*)))) (*p*^*j*∗(*monom 1 i*))) *X*

definition *form-basis-coppersmith-aux*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list*
where *form-basis-coppersmith-aux p M X h j* = (*map* (λ*i*. *row-of-coppersmith-matrix p M X h i j*) [*0*..<*degree p*])

definition *form-basis-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int vec list*
where *form-basis-coppersmith p M X h* = *concat* (*map* (λ*j*. *form-basis-coppersmith-aux p M X h j*) [*0*..< (*h*::*nat*)])

definition *calculate-h-coppersmith-aux*:: *int poly* ⇒ *real* ⇒ *int* **where**
calculate-h-coppersmith-aux p e = (*let d* = *degree p in* ((*ceiling* (((*d*−*1*)/(*d*∗(*e*::*real*)) + *1*)/*d*))::*int*))

definition *calculate-h-coppersmith*:: *int poly* ⇒ *real* ⇒ *nat* **where**
calculate-h-coppersmith p e = (*nat* (*calculate-h-coppersmith-aux p e*))

Note that we pass 2 as a parameter to the LLL algorithm. Any bound $> 4/3$ would work.

definition *first-vector-coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *real* ⇒ *int vec* **where**
first-vector-coppersmith p M X epsilon = (

let cs-basis = form-basis-coppersmith p M X (calculate-h-coppersmith p epsilon)
in
short-vector 2 cs-basis)

definition *coppersmith*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *real* ⇒ *int poly* **where**
coppersmith p M X epsilon =
int-poly-associated-to-vec (*first-vector-coppersmith p M X epsilon*) *X*

end

3 Howgrave-Graham's theorem

In this file, we prove a result due to Howgrave-Graham on small-enough roots of polynomials mod M (see Theorem 19.1.2 in "Mathematics of Public Key Cryptography" by Galbraith).

theory *Howgrave-Graham*

imports *Coppersmith-Algorithm*
HOL−Analysis.L2-Norm
LLL-Basis-Reduction.Norms
begin

abbreviation *euclidean-norm-int-vec*::*int vec* ⇒ *real*
where *euclidean-norm-int-vec v* ≡ *sqrt* (*sq-norm-vec v*)

abbreviation *euclidean-norm-real-vec*::*real vec* ⇒ *real*
where *euclidean-norm-real-vec v* ≡ *sqrt* (*sq-norm-vec v*)

lemma *euclidean-norm-int-vec-eq*:
shows *euclidean-norm-int-vec v* = *sqrt* (∑ *i*<(*dim-vec v*). (*v*$*i*)^*2*)
⟨*proof*⟩

lemma *euclidean-norm-real-vec-eq*:
shows *sqrt* (*sq-norm-vec v*) = *sqrt* (∑ *i*<(*dim-vec v*). (*v*$*i*)^*2*)
⟨*proof*⟩

lemma *euclidean-norm-gteq0*:
shows *euclidean-norm-real-vec* (*a*::*real vec*) ≥ *0*
euclidean-norm-int-vec (*c*::*int vec*) ≥ *0*
⟨*proof*⟩

lemma *dim-vec-vec-associated-to-poly*[*simp*]:
shows *dim-vec* (*vec-associated-to-poly F X*) = *degree F* + *1*
⟨*proof*⟩

lemma *Cauchy-Schwarz-sum*:
fixes *n*:: *nat*
fixes *x*:: *nat* ⇒ *real*

shows $(\sum i{\le}n.\ x\ i) \le sqrt\ ((n{+}1) * (\sum i{\le}n.\ (x\ i)\hat{}2))$
⟨*proof*⟩

lemma *abs-mult-sum*:
fixes *f g*:: *nat* ⇒ *real*
fixes *n*:: *nat*
shows $abs(\sum i{\le}n.\ (f\ i){*}(g\ i))$
$\le (\sum i{\le}n.\ (abs\ (f\ i)){*}(abs\ (g\ i)))$
⟨*proof*⟩

lemma *sum-helper*:
fixes *g h*:: *nat* ⇒ *real*
assumes $\forall\ i \le n.\ f\ i \ge 0$
assumes $\forall i \le n.\ g\ i \le h\ i$
shows $(\sum i{\le}n.\ (f\ i){*}\ (g\ i)) \le (\sum i{\le}n.\ (f\ i){*}\ (h\ i))$
⟨*proof*⟩

theorem *Howgrave-Graham*:
fixes *F* :: *real poly*
fixes *M X* :: *nat*
fixes *x0 k* :: *int*
assumes *M-gt*: $M > 0$
assumes *root-mod-M*: *poly F* (*real-of-int x0*) = *k* ∗ *M*
assumes *root-bound*: *abs x0* ≤ *X*
assumes *norm-bound*: *sqrt* (‖*vec-associated-to-poly F X*‖^2) < *M* / *sqrt* (*degree F* + *1*)
shows *poly F x0* = *0*
⟨*proof*⟩

abbreviation *int-poly-to-real-poly*:: *int poly* ⇒ *real poly*
where *int-poly-to-real-poly F* ≡ *map-poly real-of-int F*

lemma *int-poly-to-real-poly-same-norm*:
fixes *X* :: *nat*
shows *euclidean-norm-int-vec* (*vec-associated-to-int-poly F X*) =
euclidean-norm-real-vec (*vec-associated-to-real-poly* (*int-poly-to-real-poly F*) *X*)
⟨*proof*⟩

Now we restate the result over int polys.

lemma *Howgrave-Graham-int-poly*:
fixes *F*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
assumes *M-gt*: $M > 0$
assumes *root-mod-M*: *poly F x0 mod M* = *0*
assumes *root-bound*: *abs x0* ≤ *X*
assumes *norm-bound*: *sqrt* (*sq-norm-vec* (*vec-associated-to-int-poly F X*)) < *M* / *sqrt* (*degree F* + *1*)

shows *poly F x0 = 0*
⟨*proof*⟩

end

4 Coppersmith Generic

In this file, we develop the generic argument behind Coppersmith's method.

theory *Coppersmith-Generic*

imports *Coppersmith-Algorithm*
Howgrave-Graham
More-LLL
begin

hide-const (**open**) *module.smult*

4.1 Some matrix properties

This definition should only be used for lists of vectors that correspond to square matrices

definition *vec-list-to-square-mat*:: *'a vec list* ⇒ *'a mat*
where *vec-list-to-square-mat L = mat-of-rows* (*length L*) *L*

This lemma is similar to upper_triangular_imp_distinct, from the "Jordan_Normal_Form.Matrix" library by Thiemann and Yamada.

lemma *lower-triangular-imp-distinct*:
assumes *A*: *A* ∈ *carrier-mat n n*
and *tri*: ⋀*i j. i < j* ⟹ *j < n* ⟹ *A* $$ (*i*,*j*) = *0*
and *diag*: *0* ∉ *set* (*diag-mat A*)
shows *distinct* (*rows A*)
⟨*proof*⟩

lemma *lower-triangular-imp-det-eq-0-iff*:
fixes *A* :: *'a* :: *idom mat*
assumes *A* ∈ *carrier-mat n n* **and** ⋀*i j. i < j* ⟹ *j < n* ⟹ *A* $$ (*i*,*j*) = *0*
shows *det A = 0* ⟷ *0* ∈ *set* (*diag-mat A*)
⟨*proof*⟩

This next lemma is similar to upper_triangular_imp_lin_indpt_rows, from the "Jordan_Normal_Form.VS_Connect" library by Thiemann and Yamada.

lemma (**in** *idom-vec*) *lower-triangular-imp-lin-indpt-rows*:
assumes *A*: *A* ∈ *carrier-mat n n*
and *lower-tri*: ⋀*i j. i < j* ⟹ *j < n* ⟹ *A* $$ (*i*,*j*) = *0*
and *diag*: *0* ∉ *set* (*diag-mat A*)
shows *lin-indpt* (*set* (*rows A*))

⟨*proof*⟩

lemma *g-i-vec-ith-element*:
assumes *degree p* $\geq$ *1*
assumes *i* $<$ *degree p*
assumes *M* $>$ *0*
assumes *X* $>$ *0*
shows (*g-i-vec M X i* (*degree p* + *1*)) $ *i* = *M*∗*X*^*i*
⟨*proof*⟩

lemma *ith-row-form-basis-helper*:
assumes *i* $\leq$ *d*
shows ((*form-basis-helper p M X d*)!*i*) = *g-i-vec M X i* (*degree p* + *1*)
⟨*proof*⟩

lemma *no-zeros-on-diagonal-helper*:
assumes *degree p* $\geq$ *1*
assumes *i* $<$ *degree p*
assumes *M* $>$ *0*
assumes *X* $>$ *0*
shows ((*form-basis-helper p M X* (*degree p* − *1*))!*i*)$*i* = *M*∗*X*^*i*
⟨*proof*⟩

4.2 Casting lemmas

lemma *casted-distinct-is-distinct*:
fixes *vecs*:: *int vec list*
assumes *distinct-vecs*: *distinct vecs*
shows *distinct* ((*map of-int-vec vecs*)::*rat vec list*)
⟨*proof*⟩

Copy pasted from VS_Connect, which only has it for field coefficients

lemma (**in** *vec-module*) *finsum-dim*:
shows *f* ∈ *A* → *carrier-vec n* ⟹
dim-vec (*finsum V f A*) = *n*
⟨*proof*⟩

lemma (**in** *vec-module*) *finsum-index*:
assumes *i*: *i* $<$ *n*
and *f*: *f* ∈ *X* → *carrier-vec n*
and *X*: *X* ⊆ *carrier-vec n*
shows *finsum V f X* $ *i* = *sum* (λ*x*. *f x* $ *i*) *X*
⟨*proof*⟩

lemma *is-int-rat-mul-of-int*:
assumes *snd* (*quotient-of y*) *dvd x*
shows *is-int-rat* (*of-int x* ∗ *y*)
⟨*proof*⟩

lemma *casting-lin-comb-helper-set*:

assumes *vs*: *vs* ⊆ *carrier-vec n*
assumes *ldi*: *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) *n*)
(*of-int-vec* ‘ *vs*)
shows *module.lin-dep class-ring* (*module-vec TYPE*(*int*) *n*) *vs*
⟨*proof*⟩

lemma *casting-lin-comb-helper*:
assumes *dim-vecs*: ⋀*v*. *v* ∈ *set vecs* ⟹ *dim-vec v* = *len-vec*
assumes *module.lin-indpt class-ring* (*module-vec TYPE*(*int*) (*len-vec*)) (*set vecs*)
shows ¬ (*module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*len-vec*))
(*set* ((*map of-int-vec vecs*)::*rat vec list*)))
⟨*proof*⟩

lemma *casting-lin-comb-helper-set-2*:
assumes *vs*: *vs* ⊆ *carrier-vec n*
assumes *ldi*: *module.lin-dep class-ring* (*module-vec TYPE*(*int*) *n*) *vs*
shows *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) *n*)
(*of-int-vec* ‘ *vs*)
⟨*proof*⟩

lemma *casting-lin-comb-helper-2*:
assumes *dim-vecs*: ⋀*v*. *v* ∈ *set vecs* ⟹ *dim-vec v* = *len-vec*
assumes ¬ (*module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*len-vec*))
(*set* ((*map of-int-vec vecs*)::*rat vec list*)))
shows *module.lin-indpt class-ring* (*module-vec TYPE*(*int*) (*len-vec*)) (*set vecs*)
⟨*proof*⟩

4.3 Properties of associated polynomial

lemma *f-representation*:
shows *f* = (∑ *i*<*degree f*. *monom* (*poly.coeff f i*) *i*) +
monom (*lead-coeff f*) (*degree f*)
⟨*proof*⟩

lemma *vec-associated-to-int-poly-inverse*:
assumes *X* > *0*
fixes *f*:: *int poly*
shows *f* = *int-poly-associated-to-vec* (*vec-associated-to-int-poly f X*) *X*
⟨*proof*⟩

lemma *int-poly-associated-to-vec-degree-helper-le*:
shows *degree* (∑ *i*≤*n*. *monom* (*f i*) *i*) ≤ *n*
⟨*proof*⟩

lemma *int-poly-associated-to-vec-degree-helper-lt*:
assumes *n* > *0*
shows *degree* (∑ *i*<*n*. *monom* (*f i*) *i*) < *n*
⟨*proof*⟩

lemma *int-poly-associated-to-vec-degree*:
fixes *v*:: *int vec*
assumes *dim-vec v* > *0*
shows *degree* (*int-poly-associated-to-vec v X*) < *dim-vec v*
⟨*proof*⟩

lemma *degree-associated-poly*:
shows *degree* (*int-poly-associated-to-vec v X*) ≤ *dim-vec v*
⟨*proof*⟩

lemma *degree-associated-poly-lt*:
assumes *X* > *0*
assumes *dim-vec v* ≥ *1*
shows *degree* (*int-poly-associated-to-vec v X*) < *dim-vec v*
⟨*proof*⟩

lemma *degree-of-monom-sum-list*:
fixes *ell*:: *int list*
fixes *j*:: *nat*
assumes *ell* ≠ []
shows *degree*
(∑ *i*<*length ell*. *monom*
(*ell* ! *i*) *i*) < *length ell*
⟨*proof*⟩

lemma *coeff-of-monom-sum-list*:
fixes *ell*:: *int list*
fixes *j*:: *nat*
assumes *j* < *length ell*
shows *coeff*
(∑ *i*<*length ell*. *monom*
(*ell* ! *i*) *i*) *j* = *ell* ! *j*
⟨*proof*⟩

lemma *coeff-of-monom-sum*:
fixes *a*:: *int vec*
assumes *i* < *dim-vec a*
shows *coeff*
(∑ *i*<*dim-vec a*. *monom*
(*a* $ *i*) *i*) *i* = *a* $ *i*
⟨*proof*⟩

This next lemma required showing that every vector in the lattice has its ith component divisible by X^n (a property of the basis).

lemma *access-entry-in-int-poly-associated-to-vec*:
fixes *x i*:: *nat*
fixes *v*:: *int vec*
assumes *i* < *dim-vec v*
shows (*coeff* (*int-poly-associated-to-vec v X*) *i*) = ⌊*real-of-int* (*v* $ *i*) / *real* (*X*

$\hat{}\ i)\rfloor$
⟨*proof*⟩

lemma *dim-vec-associated-to-int-poly-lteq*:
fixes *v*:: *int vec*
assumes $\mathit{dim\text{-}vec}\ v \geq 1$
assumes $X > 0$
shows $\mathit{dim\text{-}vec}\ (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X) \leq \mathit{dim\text{-}vec}\ v$
⟨*proof*⟩

lemma *dim-vec-associated-to-int-poly-lt-imp-zeros*:
fixes *v*:: *int vec*
fixes *i*:: *nat*
assumes *entries-div*: $\bigwedge j.\ j < \mathit{dim\text{-}vec}\ v \Longrightarrow (\exists\, k.\ (X\hat{}j)*k = (v\ \$\ j))$
assumes *X-gt*: $X > 0$
assumes $\mathit{dim\text{-}vec}\ (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X) < \mathit{dim\text{-}vec}\ v$
assumes *i-gt*: $i \geq \mathit{dim\text{-}vec}\ (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X)$
assumes *i-lt*: $i < \mathit{dim\text{-}vec}\ v$
shows $v\ \$\ i = 0$
⟨*proof*⟩

lemma *int-poly-associated-to-notNil-vec-same-norm-upto*:
assumes $\mathit{dim\text{-}vec}\ v \geq 1$
assumes *X-gt*: $X > 0$
assumes *entries-div*: $\bigwedge j.\ j < \mathit{dim\text{-}vec}\ v \Longrightarrow (\exists\, k.\ (X\hat{}j)*k = (v\ \$\ j))$
assumes $w = (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X)$
shows $(\sum i{<}(\mathit{dim\text{-}vec}\ v).\ (v\$i)\hat{}2) = (\sum i{<}(\mathit{dim\text{-}vec}\ w).\ (v\$i)\hat{}2)$
⟨*proof*⟩

lemma *int-poly-associated-to-notNil-vec-same-entries*:
fixes *i*:: *nat*
fixes *v w*:: *int vec*
assumes *i-lt*: $i < \mathit{dim\text{-}vec}\ w$
assumes $\mathit{dim\text{-}vec}\ v \geq 1$
assumes *X-gt*: $X > 0$
assumes *entries-div*: $\bigwedge j.\ j < \mathit{dim\text{-}vec}\ v \Longrightarrow (\exists\, k.\ (X\hat{}j)*k = (v\ \$\ j))$
assumes *w-is*: $w = (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X)$
shows $((v\ \$\ i) = (w\ \$\ i))$
⟨*proof*⟩

lemma *int-poly-associated-to-nil-vec-same-norm*:
assumes $X > 0$
assumes $\mathit{dim\text{-}vec}\ v = 0$
shows $\mathit{euclidean\text{-}norm\text{-}int\text{-}vec}\ v = \mathit{euclidean\text{-}norm\text{-}int\text{-}vec}\ (\mathit{vec\text{-}associated\text{-}to\text{-}int\text{-}poly}\ (\mathit{int\text{-}poly\text{-}associated\text{-}to\text{-}vec}\ v\ X)\ X)$
⟨*proof*⟩

lemma *int-poly-associated-to-notNil-vec-same-norm*:
assumes *X-gt*: $X > 0$
assumes *dim-vec* $v > 0$
assumes $\bigwedge j.\ j <$ *dim-vec* $v \Longrightarrow (\exists k.\ (X\hat{\ }j)*k = (v\ \$\ j))$
shows *euclidean-norm-int-vec* v = *euclidean-norm-int-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
⟨*proof*⟩

lemma *int-poly-associated-to-vec-same-norm*:
assumes $X > 0$
assumes $\bigwedge j.\ j <$ *dim-vec* $v \Longrightarrow (\exists k.\ (X\hat{\ }j)*k = (v\ \$\ j))$
shows *euclidean-norm-int-vec* v = *euclidean-norm-int-vec* (*vec-associated-to-int-poly* (*int-poly-associated-to-vec* v X) X)
⟨*proof*⟩

lemma *int-poly-associated-to-vec-sum*:
assumes *dim-vec* a = *dim-vec* b
assumes $\bigwedge i.\ i <$ *dim-vec* $a \Longrightarrow$
$a\ \$\ i$ *mod* $X\ \hat{}\ i = 0$
assumes $\bigwedge i.\ i <$ *dim-vec* $b \Longrightarrow$
$b\ \$\ i$ *mod* $X\ \hat{}\ i = 0$
assumes $X > 0$
shows *int-poly-associated-to-vec* $(a+b)$ X = *int-poly-associated-to-vec* a X + *int-poly-associated-to-vec* b X
⟨*proof*⟩

lemma *int-poly-associated-to-vec-constant-mult*:
fixes c:: *nat* ⇒ *int*
fixes v:: *int vec*
assumes $X > 0$
assumes *div-by-X*: $\bigwedge i.\ i <$ *dim-vec* $v \Longrightarrow$
$v\ \$\ i$ *mod* $X\ \hat{}\ i = 0$
shows *int-poly-associated-to-vec* $(c\ i \cdot_v v)$ X =
smult $(c\ i)$ (*int-poly-associated-to-vec* v X)
⟨*proof*⟩

4.4 Generic Coppersmith assumptions locale

The generic properties required are stored in this locale.

locale *coppersmith-assms* = *LLL-with-assms* +
fixes *x0* :: *int*
fixes *M X* :: *nat*
fixes *F* :: *int poly*
assumes *M*: $M > 0$
assumes *X*: $X > 0$
assumes *n*: $n > 0$
assumes *x0*: *poly F x0 mod M* = 0 $|x0| \leq$ *int X*
assumes *lfs*: *length fs-init* $\neq 0$

assumes *Xpoly*: $\bigwedge v\ j.$
  $v \in set\ fs\text{-}init \Longrightarrow j < n \Longrightarrow$
  $v\ \$\ j\ mod\ int\ X\ \hat{}\ j = 0$
assumes *rt*: $\bigwedge v.$
  $v \in set\ fs\text{-}init \Longrightarrow$
  $poly\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ v\ X)\ x0\ mod\ M = 0$
begin

lemma *sumlist-mod*:
  assumes $\bigwedge v.\ v \in set\ xs \Longrightarrow dim\text{-}vec\ v = n$
  assumes *m0*: $\bigwedge v\ j.$
    $v \in set\ xs \Longrightarrow j < n \Longrightarrow$
    $v\ \$\ j\ mod\ int\ X\ \hat{}\ j = 0$
  assumes $i < n$
  shows $M.sumlist\ xs\ \$\ i\ mod\ X\ \hat{}\ i = 0$
⟨*proof*⟩

lemma *poly-associated-to-vec-sumlist*:
  assumes $\bigwedge v.\ v \in set\ xs \Longrightarrow dim\text{-}vec\ v = n$
  assumes $\bigwedge v\ j.\ v \in set\ xs \Longrightarrow$
    $j < dim\text{-}vec\ v \Longrightarrow v\ \$\ j\ mod\ X\ \hat{}\ j = 0$
  shows $int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ (sumlist\ xs)\ X =$
    $(\sum i < length\ xs.\ int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ (xs\ !\ i)\ X)$
  ⟨*proof*⟩

lemma *short-vector-inherit-props*:
  shows $\bigwedge j.\ j < n \Longrightarrow short\text{-}vector\ \$\ j\ mod\ X\ \hat{}\ j = 0$
    $poly\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ short\text{-}vector\ X)\ x0\ mod\ M = 0$
⟨*proof*⟩

lemma *root-poly-short-vector*:
  assumes *bnd*: $real\text{-}of\text{-}int\ \|short\text{-}vector\|^2 < M\hat{}2\ /\ n$
  shows $poly\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ short\text{-}vector\ X)\ x0 = 0$
⟨*proof*⟩

lemma *bnd-raw-imp-short-vec-bound*:
  assumes *bnd-raw*:
    $(real\text{-}of\text{-}rat\ \alpha)\ \hat{}\ (m * (m - 1)\ div\ 2) *$
    $real\text{-}of\text{-}int\ (gs.Gramian\text{-}determinant\ fs\text{-}init\ m) *$
    $(real\ n)\ \hat{}\ m <$
    $M\ \hat{}\ (2 * m)$
  shows $real\text{-}of\text{-}int\ \|short\text{-}vector\|^2 < M\hat{}2\ /\ n$
⟨*proof*⟩

end

end

5 Proof of Lightweight Algorithm

We start by proving the "lightweight algorithm" as a stepping stone to the full algorithm (proved in Coppersmith.thy).

theory *Towards-Coppersmith*

imports *Coppersmith-Generic*
begin

5.1 Basic properties

lemma *dim-form-basis-helper*:
  shows *length* (*form-basis-helper p M X d*) = *d* + *1*
  ⟨*proof*⟩

lemma *dim-form-basis*:
  shows *length* (*form-basis p M X d*) = *d* + *2*
  ⟨*proof*⟩

5.2 Matrix properties

5.2.1 Basic properties and preliminaries

lemma *dim-row-basis-mat*:
  assumes *degree p* ≥ *1*
  shows *dim-row* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) = *degree p* + *1*
⟨*proof*⟩

lemma *dim-col-basis-mat*:
  assumes *degree p* ≥ *1*
  shows*dim-col* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) = (*degree p* + *1*)
⟨*proof*⟩

lemma *matrix-carrier*:
  assumes *degree p* ≥ *1*
  assumes *i* < *degree p* + *1*
  shows *vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*)) ∈ *carrier-mat* (*degree p* + *1*) (*degree p* + *1*)
  ⟨*proof*⟩

lemma *vector-sum-monom*:
  fixes *v*:: *int vec*
  fixes *d i*:: *nat*
  assumes *d* = *dim-vec v*
  assumes *d* > *0*
  assumes *i-lt*: *i* < *d*
  assumes *zero-monom*: ⋀*j*::*nat*. *j* < *d* ⟹ *j* ≠ *i* ⟹ *monom* (*v* $ *j*) *j* = *0*

shows ($\sum$ *i*< *d*. *monom* (*v* $ *i*) *i*) = *monom* (*v* $ *i*) *i*
⟨*proof*⟩

lemma *int-poly-associated-to-g-i-vec*:
assumes *X-gt*: *X* > *0*
assumes *M-gt*: *M* > *0*
assumes *i-lt*: *i* ≤ (*degree p*)
shows *int-poly-associated-to-vec* (*g-i-vec M X i* (*degree p* + *1*)) *X* = *monom M i*
⟨*proof*⟩

lemma *ith-row-form-basis*:
shows *i* ≤ *d* ⟹ ((*form-basis p M X d*)!*i*) = (*form-basis-helper p M X d*)!*i*
((*form-basis p M X d*)!(*d*+*1*)) = *vec-associated-to-int-poly p X*
⟨*proof*⟩

lemma *set-form-basis*:
shows *x*∈ *set* (*form-basis p M X* (*degree p* − *1*)) ⟹ *x* = *vec-associated-to-int-poly p X* ∨
x ∈ *set* (*form-basis-helper p M X* (*degree p* − *1*))
⟨*proof*⟩

lemma *dim-vector-in-basis*:
fixes *i*:: *nat*
assumes *i* < *d* + *2*
shows *dim-vec* ((*form-basis p M X d*) ! *i*) = (*degree p*+*1*)
⟨*proof*⟩

5.2.2 Properties of matrix associated to input

lemma *matrix-row-form-basis-carrier*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p* + *1*
shows *form-basis p M X* (*degree p* − *1*) ! *i* ∈ *carrier-vec* (*degree p* + *1*)
⟨*proof*⟩

lemma *matrix-row-form-basis*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) *i* = (*form-basis p M X* (*degree p* − *1*)) ! *i*
⟨*proof*⟩

lemma *matrix-diagonal-element*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p*
shows *vec-list-to-square-mat*
(*form-basis p M X* (*degree p* − *1*)) $$
(*i*, *i*) = ((*form-basis p M X* (*degree p* − *1*)) ! *i*)$*i*
⟨*proof*⟩

lemma *no-zeros-on-diagonal*:
  assumes *degree p* ≥ *1*
  assumes *M* > *0*
  assumes *X* > *0*
  shows *0* ∉ *set* (*diag-mat* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))))
⟨*proof*⟩

lemma *form-basis-helper-is-lower-triangular*:
  fixes *i j*:: *nat*
  assumes *i* < *j*
  assumes *j* < (*degree p* + *1*)
  shows ((*form-basis-helper p M X* (*degree p* − *1*))!*i*)$*j* = *0*
⟨*proof*⟩

lemma *form-basis-is-lower-triangular*:
  fixes *i j*::*nat*
  assumes *i-lt*: *i* < *j*
  assumes *j-lt*: *j* < (*degree p* + *1*)
  shows (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) $$ (*i*,*j*) = *0*
⟨*proof*⟩

lemma *form-basis-distinct*:
  assumes *degree p* ≥ *1*
  assumes *M* > *0*
  assumes *X* > *0*
  shows *distinct* (*form-basis p M X* (*degree p* − *1*))
⟨*proof*⟩

lemma *det-of-matrix*:
  fixes *M X*:: *nat*
  assumes *M* > *0*
  assumes *X* > *0*
  assumes *degree p* ≥ *1*
  assumes *d-is*: *d* = *degree p*
  assumes *n-is*: *n* = (∑ *i*≤*d*. *i*)
  assumes *monic-poly*: *coeff p d* = *1*
  shows *det* (*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) =
    M^(*degree p*)∗*X*^*n*
⟨*proof*⟩

5.2.3 Properties of casted matrix

lemma *dim-row-basis-of-int-mat*:

assumes *degree p* ≥ *1*
shows *dim-row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) = *degree p* + *1*
⟨*proof*⟩

lemma *dim-col-basis-of-int-mat*:
assumes *degree p* ≥ *1*
shows*dim-col* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) = (*degree p* + *1*)
⟨*proof*⟩

lemma *of-int-matrix-row-form-basis-carrier*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p* + *1*
shows (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))) ! *i* ∈ *carrier-vec* (*degree p* + *1*)
⟨*proof*⟩

lemma *of-int-matrix-carrier*:
assumes *degree p* ≥ *1*
assumes *i* < *degree p* + *1*
shows *vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))) ∈ *carrier-mat* (*degree p* + *1*) (*degree p* + *1*)
⟨*proof*⟩

lemma *of-int-matrix-row-form-basis*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) *i* = (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*))) ! *i*
⟨*proof*⟩

lemma *of-int-matrix-row-form-basis-var*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < *degree p* + *1*
shows *row* (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) *i* = *of-int-vec* ((*form-basis p M X* (*degree p* − *1*)) ! *i*)
⟨*proof*⟩

lemma *of-int-mat-element*:
fixes *i j*::*nat*
assumes *degree p* ≥ *1*
assumes *i* < (*degree p* + *1*)
assumes *j* < (*degree p* + *1*)
shows (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) $$ (*i*,*j*) =
of-int ((*vec-list-to-square-mat* (*form-basis p M X* (*degree p* − *1*))) $$ (*i*,*j*))
⟨*proof*⟩

lemma *of-int-mat-is-lower-triangular*:
  fixes *i j::nat*
  assumes *degree p* ≥ 1
  assumes $i < j$
  assumes $j <$ (*degree p* + *1*)
  shows (*vec-list-to-square-mat* (*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))) $$ (*i*,*j*) = *0*
  ⟨*proof*⟩

lemma *of-int-mat-no-zeros-on-diagonal*:
  assumes *p-gt*: *degree p* ≥ 1
  assumes $M > 0$
  assumes $X > 0$
  shows *0* ∉ *set* (*diag-mat* (*vec-list-to-square-mat* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))::*rat vec list*)))
⟨*proof*⟩

lemma *of-int-mat-form-basis-distinct*:
  assumes *degree p* ≥ 1
  assumes $M > 0$
  assumes $X > 0$
  shows *distinct* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))::*rat vec list*)

⟨*proof*⟩

lemma *of-int-form-basis-lin-ind*:
  assumes $M > 0$
  assumes $X > 0$
  assumes *degree p* ≥ 1
  shows ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*rat*) (*degree p* + *1*))
    (*set* ((*map of-int-vec* (*form-basis p M X* (*degree p* − *1*)))::*rat vec list*))
⟨*proof*⟩

5.3 Top-level proof

lemma *towards-coppersmith*:
  fixes *p f*:: *int poly*
  fixes *M X*:: *nat*
  fixes *x0*:: *int*
  assumes *zero-mod-M*: *poly p x0 mod M* = *0*
  assumes *d-is*: *d* = *degree p*
  assumes *d-gt*: $d > 0$
  assumes *monic-poly*: *coeff p d* = *1*
  assumes *X-gt*: $X > 0$
  assumes *X-lt*: *X* < *1*/(*sqrt 2*)∗*1*/(*root d* (*d*+*1*))∗(*root* (*d*∗(*d*+*1*)) *M*)^*2*
  assumes *M-gt*: $M > 0$
  assumes *x0-le*: *abs x0* ≤ *X*
  assumes *f-is*: *f* = *towards-coppersmith p M X*
  shows *poly f x0* = *0*

⟨*proof*⟩

lemma *towards-coppersmith-pretty*:
  fixes *p f*:: *int poly*
  fixes *M X*:: *nat*
  fixes *x0*:: *int*
  defines *d* ≡ *degree p*
  defines *f* ≡ *towards-coppersmith p M X*
  assumes *monic-poly*: *monic p*
  assumes *d* > *0* **and** *M* > *0* **and** *X* > *0*
  assumes *zero-mod-M*: *poly p x0 mod M* = *0*
  assumes *X-lt*: *X* < *1*/(*sqrt 2*)∗*1*/(*root d* (*d*+*1*))∗(*root* (*d*∗(*d*+*1*)) *M*)^*2*
  assumes *x0-le*: *abs x0* ≤ *X*
  shows *poly f x0* = *0*
  ⟨*proof*⟩

end

6 Proof of Coppersmith's Method

In this file, we prove the full version of Coppersmith's method.

theory *Coppersmith*
imports *Coppersmith-Generic*
begin

6.1 Preliminaries and setup

lemma *calculate-h-coppersmith-aux-gteq1*:
  fixes *e*::*real*
  assumes *degree p* > *1*
  assumes *e* > *0*
  shows *calculate-h-coppersmith-aux p e* ≥ *1*
⟨*proof*⟩

lemma *calculate-h-coppersmith-aux-gt1*:
  assumes *deg-gt*: *degree p* > *1*
  assumes *e* > *0*
  assumes *e-lt2*: *e* < *1*/(*real* (*degree p*))
  shows *calculate-h-coppersmith-aux p e* > *1*
⟨*proof*⟩

6.1.1 Dimension properties of matrix

lemma *dim-vector-vec-associated-to-int-poly-padded*:
  shows *dim-vec* (*vec-associated-to-int-poly-padded n p X*) = *n*
  ⟨*proof*⟩

lemma *dim-vector-row-of-coppersmith-matrix*:
  shows *dim-vec* (*row-of-coppersmith-matrix p M X h i j*) = (*degree p*)∗*h*

⟨*proof*⟩

lemma *dim-vector-form-basis-coppersmith-aux*:
fixes *i*:: *nat*
assumes $i < (degree\ p)$
shows *dim-vec* ((*form-basis-coppersmith-aux p M X h j*) ! *i*) = (*degree p*)∗*h*
⟨*proof*⟩

lemma *length-form-basis-coppersmith-aux*:
shows *length* (*form-basis-coppersmith-aux p M X h j*) = *degree p*
⟨*proof*⟩

lemma *length-form-basis-coppersmith*:
fixes *h*:: *nat*
assumes *h-gt*: $h > 0$
shows *length* (*form-basis-coppersmith p M X h*) = (*degree p*)∗*h*
⟨*proof*⟩

lemma *concat-property-helper*:
assumes $j < h$
shows *concat* (*map* (λ*i*. *f i*) [*0*..<*h*]) = *concat* (*map* (λ*i*. *f i*) [*0*..<*j*]) @ *concat* (*map* (λ*i*. *f i*) [*j*..<*h*])
⟨*proof*⟩

lemma *concat-equal-lists-length*:
fixes *f*:: *nat* ⇒ *int vec list*
fixes *i j d*:: *nat*
assumes *len-is*: ⋀*i*. *i* < *h* ⟹ *length* (*f i*) = *d*
shows *length* (*concat* (*map* (λ*i*. *f i*) [*0*..<*h*])) = *d*∗*h*
⟨*proof*⟩

lemma *concat-property*:
fixes *f*:: *nat* ⇒ *int vec list*
fixes *i j d*:: *nat*
assumes $h > 0$
assumes *d-gt*: $d > 0$
assumes *r-lt*: *r* < *d*∗*h*
assumes *len-is*: ⋀*i*. *i* < *h* ⟹ *length* (*f i*) = *d*
assumes *j-eq*: *j* = *nat* ⌊*real r* / *real d*⌋
assumes *i-eq*: *i* = *r* − *d* ∗ *j*
shows (*concat* (*map* (λ*i*. *f i*) [*0*..<*h*])) ! *r* = (*f j*) ! *i*
⟨*proof*⟩

lemma *row-of*:
assumes *r-lt*: *r* < (*degree p*)∗*h*
defines *d* ≡ *degree p*
defines *j* ≡ *nat* ⌊*real r* / *real d*⌋
defines *i* ≡ *r* − *d* ∗ *j*
shows (*form-basis-coppersmith p M X h*) ! *r* =

((*form-basis-coppersmith-aux p M X h j*) ! *i*)
⟨*proof*⟩

lemma *dim-vector-form-basis-coppersmith*:
fixes *i*:: *nat*
assumes *i* < (*degree p*)∗*h*
shows *dim-vec* ((*form-basis-coppersmith p M X h*) ! *i*) = (*degree p*)∗*h*
⟨*proof*⟩

6.1.2 Equivalent Coppersmith matrix

definition *form-coppersmith-matrix*:: *int poly* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int mat*
where *form-coppersmith-matrix p M X h* = *mat* ((*degree p*)∗*h*) ((*degree p*)∗*h*)
(λ(*r*, *c*). (*let d* = *degree p*; *j* = *nat* (*floor* (*r*/*d*)); *i* = (*r* − *d*∗*j*) *in* (*M*^(*h*−*1*−*j*))∗(*coeff* (*p*^*j*∗ (*monom 1 i*)) *c*)∗*X*^*c*))

lemma *matrix-match*:
assumes *r-lt*: *r* < (*degree p*)∗*h*
assumes *c-lt*: *c* < (*degree p*)∗*h*
assumes *h* > *0*
shows (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*))$$(*r*, *c*) = (*form-coppersmith-matrix p M X h*)$$(*r*, *c*)
⟨*proof*⟩

6.1.3 Lower triangular

lemma *form-coppersmith-matrix-is-lower-triangular*:
fixes *r c*::*nat*
assumes *h* > *0*
assumes *r-lt*: *r* < *c*
assumes *c-lt*: *c* < (*degree p*)∗*h*
shows (*form-coppersmith-matrix p M X h*)$$(*r*, *c*) = *0*
⟨*proof*⟩

lemma *form-coppersmith-basis-is-lower-triangular*:
fixes *i j*::*nat*
assumes *h* > *0*
assumes *i-lt*: *i* < *j*
assumes *j-lt*: *j* < (*degree p*)∗*h*
shows (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) $$ (*i*,*j*) = *0*
⟨*proof*⟩

6.1.4 Distinct elements

lemma *coppersmith-matrix-carrier-mat*:
assumes *h* > *0*
shows *vec-list-to-square-mat* (*form-basis-coppersmith p M X h*) ∈ *carrier-mat* ((*degree p*)∗*h*) ((*degree p*)∗*h*)
⟨*proof*⟩

lemma *no-zeros-on-diagonal-coppersmith*:
assumes *degree p* ≥ *1*
assumes *M-gt*: *M* > *0*
assumes *X-gt*: *X* > *0*
assumes *h-gt*: *h* > *0*
shows *0* ∉ *set* (*diag-mat* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)))
⟨*proof*⟩

lemma *form-basis-coppersmith-distinct*:
fixes *M X*::*nat*
assumes *1* ≤ *degree p*
assumes *p-neq*: *p* ≠ *0*
assumes *M-gt*: *M* > *0*
assumes *X-gt*: *X* > *0*
assumes *h-gt*: *h* > *0*
shows *distinct* (*form-basis-coppersmith p M X h*)
⟨*proof*⟩

lemma *matrix-row-form-basis-coppersmith*:
assumes *degree p* ≥ *1*
assumes *i-lt*: *i* < (*degree p*)∗*h*
assumes *h* > *0*
shows *row* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) *i* = (*form-basis-coppersmith p M X h*) ! *i*
⟨*proof*⟩

6.1.5 Linear independence properties

lemma *form-basis-coppersmith-lin-ind*:
assumes *M* > *0*
assumes *X* > *0*
assumes *degree p* ≥ *1*
assumes *h* > *0*
shows ¬ *module.lin-dep class-ring* (*module-vec TYPE*(*int*) ((*degree p*)∗*h*))
(*set* (*form-basis-coppersmith p M X h*))
⟨*proof*⟩

6.1.6 Divisible by X property

lemma *row-of-cs-matrix-div-by-Xpow*:
fixes *M X i j h*::*nat*
assumes *i-lt*: *i* < *degree p*
assumes *j-lt*: *j* < *h*
assumes *j-lt*: *ja* < *h*∗(*degree p*)
shows (*row-of-coppersmith-matrix p M X h i j*) $ *ja mod* (*X* ^ *ja*) = *0*
⟨*proof*⟩

lemma *form-basis-coppersmith-div-by-Xpow*:
fixes *M X*::*nat*

fixes *a*:: *int vec*
assumes *j-lt*: $ja < h*(degree\ p)$
assumes *a-inset*: $a \in set\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h)$
shows $(a\ \$\ ja)\ mod\ (X\ \hat{}\ ja) = 0$
⟨*proof*⟩

6.1.7 Zero mod M property

lemma *row-of-cs-matrix-zero-mod-M*:
fixes *M X i j h*::*nat*
assumes *p-neq*: $p \neq 0$
assumes *M-gt*: $M > 0$
assumes *X-gt*: $X > 0$
assumes *h-gt1*: $h > 1$
assumes *p-zero-mod-M*: $poly\ p\ x0\ mod\ M = 0$
assumes *deg-gt*: $degree\ p > 1$
assumes *i-lt*: $i < degree\ p$
assumes *j-lt*: $j < h$
shows $poly\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ (row\text{-}of\text{-}coppersmith\text{-}matrix\ p\ M\ X\ h\ i\ j)\ X)\ x0\ mod\ M\ \hat{}(h-1) = 0$
⟨*proof*⟩

lemma *form-basis-coppersmith-zero-mod-M*:
fixes *M X*::*nat*
assumes *p-neq*: $p \neq 0$
assumes *M-gt*: $M > 0$
assumes *X-gt*: $X > 0$
assumes *h-gt*: $h > 1$
assumes *p-zero-mod-M*: $poly\ p\ x0\ mod\ M = 0$
assumes *a-inset*: $a \in set\ (form\text{-}basis\text{-}coppersmith\ p\ M\ X\ h)$
assumes *deg-gt*: $degree\ p > 1$
shows $poly\ (int\text{-}poly\text{-}associated\text{-}to\text{-}vec\ a\ X)\ x0\ mod\ M\ \hat{}\ (h - 1) = 0$
⟨*proof*⟩

6.1.8 Determinant of matrix

lemma *determinant-bound-arithmetic-helper*:
fixes *k*:: *nat*
shows $(\prod\ j{<}(w{+}1).\ k\hat{}j) = sqrt\ (k\hat{}(w*(w{+}1)))$
⟨*proof*⟩

lemma *det-of-form-coppersmith-matrix*:
fixes *M X*:: *nat*
assumes $M > 0$
assumes $X > 0$
assumes *d-is*: $d = degree\ p$
assumes *monic-poly*: $coeff\ p\ d = 1$
assumes *h-gt*: $h > 1$
assumes *deg-gt*: $degree\ p > 1$
shows $det\ (form\text{-}coppersmith\text{-}matrix\ p\ M\ X\ h) =$

(*root 2* (*M*^((*h*−*1*)∗*d*∗*h*))) ∗ (*root 2* (*X*^((*d*∗*h*−*1*)∗*d*∗*h*)))
⟨*proof*⟩

lemma *det-of-matrix*:
fixes *M X*:: *nat*
assumes *M* > *0*
assumes *X* > *0*
assumes *d* > *1*
assumes *d-is*: *d* = *degree p*
assumes *monic-poly*: *coeff p d* = *1*
assumes *h-gt*: *h* > *1*
shows *det* (*vec-list-to-square-mat* (*form-basis-coppersmith p M X h*)) =
(*root 2* (*M*^((*h*−*1*)∗*d*∗*h*))) ∗ (*root 2* (*X*^((*d*∗*h*−*1*)∗*d*∗*h*)))
⟨*proof*⟩

6.2 Top-level proof

definition *root-bound*:: *nat* ⇒ *nat* ⇒ *real* ⇒ *real*
where *root-bound M d eps* = *1*/*2*∗(*M powr* (*1*/*d* − *eps*))

6.2.1 Arithmetic

lemma *epsilon-bounded-below*:
assumes *d* > *0*
assumes *eps* > *0*
assumes *d*∗*h*−*1* > *0*
assumes *d* = *degree p*
assumes *h* = *calculate-h-coppersmith p eps*
shows *eps* ≥ (*d*−*1*)/(*d*∗(*d*∗*h*−*1*))
⟨*proof*⟩

lemma *z-arith*:
assumes *x*: (*x*::*real*) ≥ *0*
shows *x* / (*1* + *x*) ≤ *ln* (*1* + *x*)
⟨*proof*⟩

lemma *powr-divide-both*:
assumes (*a*::*real*) ≥ *0 x* > *0 b powr y* ≥ *1*
assumes *le*: *a powr x* ≥ *b powr y*
shows *a* ≥ *b powr* (*y* / *x*)
⟨*proof*⟩

lemma *coppersmith-arithmetic-convergence-1*:
fixes *y*:: *real*
assumes *y*: *y* ≥ *1* / *0.18*
shows *0* ≤ *1.414* − (*1* + *y*) *powr* (*1* / *y*)
⟨*proof*⟩

lemma *coppersmith-arithmetic-convergence*:
fixes *x*:: *real*

assumes *x*: *0* < *x* *x* ≤ *0.18*
shows (*1* + (*1*/*x*)) *powr* (*x*) ≤ *sqrt 2*
⟨*proof*⟩

6.2.2 Main results

lemma *coppersmith-finds-small-roots*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *eps*::*real*
assumes *zero-mod-M*: *poly p x0 mod M* = *0*
assumes *d-is*: *d* = *degree p*
assumes *d-gt*: *d* > *1*
assumes *monic-poly*: *coeff p d* = *1*
assumes *X-lt*: *X* < *root-bound M d eps*
assumes *M-gt*: *M* > *0*
assumes *X-gt-zero*: *X* > *0*
assumes *x0-le*: *abs x0* ≤ *X*
assumes *eps-le*: *eps* ≤ *0.18*∗(*d*−*1*)/*d*
assumes *eps-lt*: *eps* < *1*/(*real* (*degree p*))
assumes *eps-gt*: *eps* > *0*
assumes *f-is*: *f* = *coppersmith p M X eps*
shows *poly f x0* = *0*
⟨*proof*⟩

theorem *coppersmith-finds-small-roots-pretty*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *eps*:: *real*
defines *d* ≡ *degree p*
defines *f* ≡ *coppersmith p M X eps*
assumes *monic-poly*: *monic p*
assumes *d* > *1* **and** *M* > *0* **and** *X* > *0*
assumes *zero-mod-M*: *poly p x0 mod M* = *0*
assumes *X-lt*: *X* < *root-bound M d eps*
assumes *x0-le*: *abs x0* ≤ *X*
assumes *eps-le*: *eps* ≤ *0.18* ∗ (*d*−*1*)/*d*
assumes *eps-lt*: *eps* < *1* / *d*
assumes *eps-gt0*: *eps* > *0*
shows *poly f x0* = *0*
⟨*proof*⟩

end

7 Examples of Coppersmith's Method

In this file, we provide some examples of Coppersmith's method, with correctness proofs (when applicable).

theory *Coppersmith-Examples*

imports *Coppersmith*
Towards-Coppersmith
begin

7.1 Example of lightweight method

Following Example 19.1.6 in Galbraith. This example produces [:444, 1, - 20, - 2:], which corresponds to $-2x^3 - 20x^2 + x + 444$, which has 4 as a root. Computing $4^3 + 10 * 4^2 + 5000 * 4 - 222$ produces 20002, which is 0 mod 10001. Note that here, we cannot use our top-level correctness result for the lightweight method to prove correctness. This is because the conditions of this top-level result are not satisfied; however, the method succeeds despite not fulfilling the conditions of this result, because the LLL algorithm can be better than the bound in the result. See Example 19.1.6 in "Mathematics of Public Key Cryptography" by Galbraith for more discussion of this.

value *towards-coppersmith* [:$-$*222*, *5000*, *10*, *1*:] *10001 10*

7.2 Examples of Coppersmith's method

Following Exercise 19.1.12 in Galbraith.
This next example produces 15955575444164700778296 - 86948462676890416832x + 50262448764961319x^2+ 334700479564525x^3 - 611446097378x^4 - 577363178x^5 + 1008850x^6 + 8592x^7, which has 267 as a root, which is a root of the original polynomial mod 2^9.

value *coppersmith* [:*227*, *46976195*, *2^25*$-$*2883584*, *1*:] ((*2^20* + *7*)$*$(*2^21* + *17*)) (*2^9*) *0.089*

We now prove that this example satisfies the conditions of our top-level correctness theorem for Coppersmith's method.

lemma *coppersmith-finds-small-roots-example1*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *k*:: *nat*
defines *p* ≡ [:*227*, *46976195*, *2^25*$-$*2883584*, *1*:]
defines *d* ≡ *degree p*
defines *M* ≡ ((*2^20* + *7*)$*$(*2^21* + *17*))
defines *X* ≡ *2^9*
defines *f* ≡ *coppersmith p M X 0.089*
assumes *x0-le*: |*x0*| ≤ *X*

assumes *zero-mod-M*: *poly p x0 mod M = 0*
shows *poly f x0 = 0*
⟨*proof*⟩

In this next example, we are trying to find small roots less than 3 of $x^3+1000x^2+25+1$ mod 4059. Running "coppersmith" on this example yields [:41858, - 28457, 6100, 376, 150, - 31, - 91, - 28, - 17:], which corresponds to $-17x^8 - 28x^7 - 91x^6 - 31x^5 + 150x^4 + 376x^3 + 6100x^2 - 28457x + 41858$, which has 2 as a root. Plugging in $2^3 + 1000 * 2^2 + 25 * 2 + 1$ indeed yields 4059, which is 0 mod 4059.

value *form-basis-coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3* (*calculate-h-coppersmith* [:*1*, *25*, *1000*, *1*:] *0.10*)
value *reduce-basis 2* (*form-basis-coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3* (*calculate-h-coppersmith* [:*1*, *25*, *1000*, *1*:] *0.10*))
value *coppersmith* [:*1*, *25*, *1000*, *1*:] *4059 3 0.10*

We now prove that this example satisifes the conditions of our top-level correctness theorem for Coppersmith's method.

lemma *coppersmith-finds-small-roots-example2*:
fixes *p f*:: *int poly*
fixes *M X*:: *nat*
fixes *x0*:: *int*
fixes *k*:: *nat*
defines *p* ≡ [:*1*, *25*, *1000*, *1*:]
defines *d* ≡ *degree p*
defines *M* ≡ *4059*
defines *X* ≡ *3*
defines *f* ≡ *coppersmith p M X 0.10*
assumes *x0-le*: |*x0*| ≤ *X*
assumes *zero-mod-M*: *poly p x0 mod M = 0*
shows *poly f x0 = 0*
⟨*proof*⟩

end

References

[1] J. Alperin-Sheriff and C. Peikert. Lattices in cryptography: Lecture 4, Coppersmith, cryptanalysis. Georgia Tech lecture notes, Fall 2023. Online lecture notes available at https://web.eecs.umich.edu/~cpeikert/lic13/lec04.pdf.

[2] R. Bottesch, J. Divasón, M. W. Haslbeck, S. J. C. Joosten, R. Thiemann, and A. Yamada. A verified LLL algorithm. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Basis_Reduction.html, Formal proof development.

[3] S. D. Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 2012.

[4] W. Trappe and L. C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition).* Prentice-Hall, Inc., USA, 2005.