

The Hidden Number Problem

Sage Binder, Eric Ren, and Katherine Kosaian

June 26, 2025

Abstract

In this entry, we formalize the Hidden Number Problem (HNP), originally introduced by Boneh and Venkatesan in 1996 [2]. Intuitively, the HNP involves demonstrating the existence of an algorithm (the “adversary”) which can compute (with high probability) a hidden number α given access to a bit-leaking oracle. Originally developed to establish the security of Diffie–Hellman key exchange, the HNP has since been used not only for protocol security but also in cryptographic attacks, including notable ones on DSA and ECDSA.

Additionally, the HNP makes use of an instance of Babai’s nearest plane algorithm [1], which solves the approximate closest vector problem. Thus, building on the LLL algorithm [5] (which has already been formalized [4, 3, 6]), we formalize Babai’s algorithm, which itself is of independent interest. Our formalizations of Babai’s algorithm and the HNP adversary are executable, setting up potential future work, e.g. in developing formally verified instances of cryptographic attacks.

Note, our formalization of Babai’s algorithm here is an updated version of a previous AFP entry of ours. The updates include a tighter error bound, which is required for our HNP proof.

Contents

1	Locale setup for Babai	3
2	Coordinates	5
3	Lattice Lemmas	7
4	Lemmas on closest distance	8
5	More linear algebra lemmas	8
6	Coord-Invariance	10
7	Bound on distance to target, with epsilon factor.	11

8	SPMF and PMF helper lemmas	14
9	General helper lemmas	19
9.1	Casting lemmas	21
10	HNP adversary locale	22
11	HNP locales	23
11.0.1	Arithmetic locale	23
11.1	Main HNP locale	24
11.2	Uniqueness lemma	24
11.2.1	Lattice construction and lemmas	25
11.2.2	dist-p definition and lemmas	29
11.2.3	Uniqueness lemma argument	30
11.2.4	Main uniqueness lemma statement	32
11.3	Main theorem	33
11.3.1	Oracle definition	33
11.3.2	Apply Babai lemmas to adversary	33
11.3.3	Main theorem statement	34
12	Some MSB instantiations and lemmas	34
12.1	Bit-shift MSB	34
12.2	MSB-p	35
13	This theory demonstrates an example of the executable adversary.	36

```

theory Babai-Algorithm-Updated
  imports
    LLL-Basis-Reduction.LLL-Impl
    HOL.Archimedean-Field
    HOL-Analysis.Inner-Product

begin

fun calculate-c:: rat vec  $\Rightarrow$  rat vec list  $\Rightarrow$  nat  $\Rightarrow$  int where
  calculate-c s L1 n = round ((s  $\cdot$  (L1!((dim-vec s) - n))) / (sq-norm-vec (L1!((dim-vec s) - n))))

fun update-s:: rat vec  $\Rightarrow$  rat vec list  $\Rightarrow$  rat vec list  $\Rightarrow$  nat  $\Rightarrow$  rat vec where
  update-s sn M Mt n = ((rat-of-int (calculate-c sn Mt n))  $\cdot_v$  M!((dim-vec sn) - n))

fun babai-help:: rat vec  $\Rightarrow$  rat vec list  $\Rightarrow$  rat vec list  $\Rightarrow$  nat  $\Rightarrow$  rat vec where
  babai-help s M Mt 0 = s |
  babai-help s M Mt (Suc n) = (let B = (babai-help s M Mt n) in B - (update-s B M Mt (Suc n)))

```

This assumes an LLL-reduced input and outputs a short vector of the form $v + t$, with $v \in L$

```
fun babai-of-LLL :: rat vec  $\Rightarrow$  rat vec list  $\Rightarrow$  rat vec where
  babai-of-LLL s M = babai-help s M (gram-schmidt (dim-vec s) M) (dim-vec s)
```

This begins with a non-reduced basis and outputs a vector v in L which is close to t

```
fun full-babai :: int vec list  $\Rightarrow$  rat vec  $\Rightarrow$  rat  $\Rightarrow$  int vec
where full-babai fs target  $\alpha$  =
  map-vec int-of-rat
    (babai-of-LLL
      (uminus target)
      (LLL.RAT (LLL-Impl.reduce-basis  $\alpha$  fs))
      + target)
```

end

```
theory Babai-Correctness-Updated
imports Babai-Algorithm-Updated
          LLL-Basis-Reduction.LLL-Impl
          Jordan-Normal-Form.DL-Rank
          BenOr-Kozen-Reif.More-Matrix
```

begin

This theory contains the proof of correctness of the algorithm. The main theorem is “theorem babai-correct”, under the locale “babai-with-assms”. To use the theorem, one needs to show that lattice, the vectors in the lattice basis, and the target vector all have the same dimension, that the lattice basis vectors are linearly independent, and that the lattice basis is LLL-weakly-reduced for some parameter $\alpha \geq 4/3$.

1 Locale setup for Babai

```
locale babai =
  fixes M :: int vec list
  fixes t :: rat vec
  fixes  $\alpha$  :: rat
  assumes length-M: length M = dim-vec t
begin
```

```
abbreviation n where n  $\equiv$  length M
sublocale LLL n n M  $\alpha$  (proof)
```

```
abbreviation coset::rat vec set where coset $\equiv$ {(map-vec rat-of-int x)-t|x. x $\in$ L}
abbreviation Mt where Mt  $\equiv$  gram-schmidt n (RAT M)
```

```
definition s :: nat  $\Rightarrow$  rat vec where
  s i = babai-help (uminus t) (RAT M) Mt i
```

definition *closest-distance-sq*:: *real* **where**

closest-distance-sq = *Inf* {*real-of-rat* (*sq-norm* *x*::*rat*) | *x*. *x* ∈ *coset*}

end

Locale setup with additional assumptions required for main theorem. Contains an arbitrary extra constant epsilon which appears in the final bound in this locale, giving a theorem quantified over all epsilon > 1. The next locale, “babai-with-assms,” uses this theorem to prove a theorem without epsilon.

locale *babai-with-assms-epsilon* = *babai* +

fixes *mat-M* *mat-M-inv*:: *rat* *mat*

fixes *epsilon*::*real*

assumes *basis*: *lin-indep* *M*

defines *mat-M* ≡ *mat-of-cols* *n* (*RAT* *M*)

defines *mat-M-inv* ≡

(*if* (*invertible-mat* *mat-M*) *then* *SOME* *B*. (*inverts-mat* *B* *mat-M*) ∧ (*inverts-mat* *mat-M* *B*) *else* (*0*_{*m*} *n* *n*))

assumes *inv*:*invertible-mat* *mat-M*

assumes *reduced*:*weakly-reduced* *M* *n*

assumes *non-trivial*:*0* < *n*

assumes *alpha*:*α* ≥ 4/3

assumes *epsilon*:*epsilon* > 1

begin

lemma *dim-vecs-in-M*:

shows ∀ *v* ∈ *set* *M*. *dim-vec* *v* = *length* *M*

⟨*proof*⟩

lemma *inv1*:*mat-M* * *mat-M-inv* = *1*_{*m*} *n*

⟨*proof*⟩

lemma *inv2*:*mat-M-inv* * *mat-M* = *1*_{*m*} *n*

⟨*proof*⟩

sublocale *rats*: *vec-module* *TYPE*(*rat*) *n*⟨*proof*⟩

lemma *M-dim*: *dim-row* *mat-M* = *n* *dim-col* *mat-M* = *n*

⟨*proof*⟩

lemma *M-inv-dim*: *dim-row* *mat-M-inv* = *n* *dim-col* *mat-M-inv* = *n*

⟨*proof*⟩

lemma *babai-to-help*:

shows *s* *n* = *babai-of-LLL* (*uminus* *t*) (*RAT* *M*)

⟨*proof*⟩

2 Coordinates

This section sets up the use of the lattice basis and its GS orthogonalization as coordinate systems and some properties of that coordinate system. The important lemma here is coord-invariance, which shows that after step i of the algorithm, all coordinates (in both systems) after $n-i$ are invariant.

definition *lattice-coord* :: *rat vec* \Rightarrow *rat vec*
where *lattice-coord* $a = \text{mat-}M\text{-inv} \cdot_v a$

lemma *dim-preserve-lattice-coord*:
fixes $v::\text{rat vec}$
assumes $\text{dim-vec } v = n$
shows $\text{dim-vec } (\text{lattice-coord } v) = n$ $\langle \text{proof} \rangle$

lemma *vec-to-col*:
assumes $i < n$
shows $(\text{RAT } M)!i = \text{col mat-}M\ i$
 $\langle \text{proof} \rangle$

lemma *unit*:
assumes $i < n$
shows $\text{lattice-coord } ((\text{RAT } M)!i) = \text{unit-vec } n\ i$
 $\langle \text{proof} \rangle$

lemma *linear*:
fixes $i::\text{nat}$
fixes $v1::\text{rat vec}$
and $v2::\text{rat vec}$
and $q::\text{rat}$
assumes $\text{dim-vec } v1 = n$
assumes $\text{dim-vec } v2 = n$
assumes $0 \leq i$
assumes $\text{dim-vec } i < n$
shows $(\text{lattice-coord } (v1 + (q \cdot_v v2)))!i = (\text{lattice-coord } v1)!i + q * ((\text{lattice-coord } v2)!i)$
 $\langle \text{proof} \rangle$

lemma *sub-s*:
fixes $i::\text{nat}$
assumes $0 \leq i$
assumes $i < n$
shows $s (\text{Suc } i) = (s\ i) -$
 $((\text{rat-of-int } (\text{calculate-c } (s\ i)\ Mt\ (\text{Suc } i))) \cdot_v (\text{RAT } M)!((\text{dim-vec } (s\ i)) - (\text{Suc } i)))$
 $\langle \text{proof} \rangle$

lemma *M-locale-1*:
shows $\text{gram-schmidt-fs-Rn } n\ (\text{RAT } M)$
 $\langle \text{proof} \rangle$

lemma *M-locale-2*:

shows *gram-schmidt-fs-lin-indpt* n (*RAT* M)

<proof>

lemma *more-dim*: *length* (*RAT* M) = n

<proof>

lemma *Mt-gso-connect*:

fixes $j::nat$

assumes $j < n$

shows $Mt!j = gs.gso\ j$

<proof>

lemma *access-index-M-dim*:

assumes $0 \leq i$

assumes $i < n$

shows *dim-vec* (*map of-int-hom.vec-hom* $M\ !\ i$) = n

<proof>

lemma *s-dim*:

fixes $i::nat$

assumes $i \leq n$

shows *dim-vec* ($s\ i$) = $n \wedge (s\ i) \in carrier-vec\ n$

<proof>

lemma *dim-vecs-in-Mt*:

fixes $i::nat$

assumes $i < n$

shows *dim-vec* ($Mt!i$) = n

<proof>

lemma *upper-tri*:

fixes $i::nat$

and $j::nat$

assumes $j > i$

assumes $j < n$

shows $((RAT\ M)!i) \cdot (Mt!j) = 0$

<proof>

lemma *one-diag*:

fixes $i::nat$

assumes $0 \leq i$

assumes $i < n$

shows $((RAT\ M)!i) \cdot (Mt!i) = sq-norm\ (Mt!i)$

<proof>

lemma *coord-invariance*:

fixes $j::nat$

fixes $k::nat$

fixes $i::nat$
assumes $k \leq j$
assumes $j+i \leq n$
assumes $k > 0$
shows $(lattice\text{-}coord\ (s\ (j+i)))\$(n-k) = (lattice\text{-}coord\ (s\ j))\$(n-k)$
 $\wedge (s\ (j+i)) \cdot Mt!(n-k) = (s\ j) \cdot Mt!(n-k)$
 $\langle proof \rangle$

lemma *small-orth-coord*:

fixes $i::nat$
assumes $1 \leq i$
assumes $i \leq n$
shows $abs\ ((s\ i) \cdot Mt!(n-i)) \leq (sq\text{-}norm\ (Mt!(n-i))) * (1/2)$
 $\langle proof \rangle$

lemma *lattice-carrier*: $L \subseteq carrier\text{-}vec\ n$
 $\langle proof \rangle$

3 Lattice Lemmas

lemma *lattice-sum-close*:

fixes $u::int\ vec$ **and** $v::int\ vec$
assumes $u \in L\ v \in L$
shows $u+v \in L$
 $\langle proof \rangle$

lemma *lattice-smult-close*:

fixes $u::int\ vec$ **and** $q::int$
assumes $u \in L$
shows $q \cdot_v u \in L$

$\langle proof \rangle$

lemma *smult-vec-zero*:

fixes $v :: 'a::ring\ vec$
shows $0 \cdot_v v = 0_v\ (dim\text{-}vec\ v)$
 $\langle proof \rangle$

lemma *coset-s*:

fixes $i::nat$
assumes $i \leq n$
shows $s\ i \in coset$
 $\langle proof \rangle$

lemma *subtract-coset-into-lattice*:

fixes $v::rat\ vec$
fixes $w::rat\ vec$
assumes $v \in coset$
assumes $w \in coset$

shows $(v-w) \in \text{of-int-hom.vec-hom}' L$
<proof>

lemma *t-in-coset*:

shows $u \text{ minus } t \in \text{coset}$
<proof>

4 Lemmas on closest distance

lemma *closest-distance-sq-pos*: $\text{closest-distance-sq} \geq 0$
<proof>

definition *witness*:: $\text{rat vec} \Rightarrow \text{rat} \Rightarrow \text{bool}$

where *witness* v *eps-closest* = $(\text{sq-norm } v \leq \text{eps-closest} \wedge v \in \text{coset} \wedge \text{dim-vec } v = n)$

definition *close-condition*:: $\text{rat} \Rightarrow \text{bool}$

where *close-condition* *eps-closest* \equiv
 $(\text{if } \text{closest-distance-sq} = 0 \text{ then } 0 \leq \text{real-of-rat } \text{eps-closest}$
 $\text{else } \text{real-of-rat } (\text{eps-closest}) > \text{closest-distance-sq}$
 $\wedge (\text{real-of-rat } (\text{eps-closest}) \leq \text{epsilon} * \text{closest-distance-sq})$

lemma *close-rat*:

obtains *eps-closest*:: rat
where *close-condition* *eps-closest*
<proof>

definition *eps-closest*:: rat

where *eps-closest* = $(\text{if } \exists r. \text{close-condition } r \text{ then } \text{SOME } r. \text{close-condition } r$
 $\text{else } 0)$

lemma *eps-closest-lemma*: *close-condition* *eps-closest*
<proof>

lemma *rational-tri-ineq*:

fixes $v::\text{rat vec}$
fixes $w::\text{rat vec}$
assumes $\text{dim-vec } v = \text{dim-vec } w$
shows $(\text{sq-norm } (v+w)) \leq 4 * (\text{Max } \{(\text{sq-norm } v), (\text{sq-norm } w)\})$
<proof>

lemma *witness-exists*:

shows $\exists v. \text{witness } v \text{ eps-closest}$
<proof>

5 More linear algebra lemmas

lemma *carrier-Ms*:

shows $\text{mat-}M \in \text{carrier-mat } n \ n \ \text{mat-}M\text{-inv} \in \text{carrier-mat } n \ n$

$\langle proof \rangle$

lemma *carrier-L*:

fixes $v::rat\ vec$

assumes $dim-vec\ v = n$

shows $lattice-coord\ v \in carrier-vec\ n$

$\langle proof \rangle$

lemma *sumlist-index-commute*:

fixes $Lst::rat\ vec\ list$

fixes $i::nat$

assumes $set\ Lst \subseteq carrier-vec\ n$

assumes $i < n$

shows $(gs.sumlist\ Lst)\$i = sum-list\ (map\ (\lambda j. (Lst!j)\$i)\ [0..<(length\ Lst)])$

$\langle proof \rangle$

lemma *mat-mul-to-sum-list*:

fixes $A::rat\ mat$

fixes $v::rat\ vec$

assumes $dim-vec\ v = dim-col\ A$

assumes $dim-row\ A = n$

shows $A*_v\ v = gs.sumlist\ (map\ (\lambda j. v\$j \cdot_v\ (col\ A\ j))\ [0..<dim-col\ A])$

$\langle proof \rangle$

lemma *recover-from-lattice-coord*:

fixes $v::rat\ vec$

assumes $dim-vec\ v = n$

shows $v = gs.sumlist\ (map\ (\lambda i. (lattice-coord\ v)\$i \cdot_v\ (RAT\ M)!i)\ [0..<n])$

$\langle proof \rangle$

lemma *sumlist-linear-coord*:

fixes $Lst::int\ vec\ list$

assumes $\bigwedge i. i < length\ Lst \implies dim-vec\ (Lst!i) = n$

shows $lattice-coord\ (map-vec\ rat-of-int\ (sumlist\ Lst)) = gs.sumlist\ (map\ lattice-coord\ (RAT\ Lst))$

$\langle proof \rangle$

lemma *integral-sum*:

fixes $l::nat$

assumes $\bigwedge j1. j1 < l \implies$

$map\ f\ [0..<l] ! j1 \in \mathbf{Z}$

shows $sum-list$

$(map\ f\ [0..<l]) \in \mathbf{Z}$

$\langle proof \rangle$

lemma *int-coord*:

```

fixes  $i::nat$ 
assumes  $0 \leq i$ 
assumes  $i < n$ 
fixes  $v::int\ vec$ 
assumes  $v \in L$ 
assumes  $dim-vec\ v = n$ 
shows  $(lattice-coord\ (map-vec\ rat-of-int\ v))\ \$i \in \mathbb{Z}$ 
<proof>

```

```

lemma int-coord-for-rat:
fixes  $i::nat$ 
assumes  $0 \leq i$ 
assumes  $i < n$ 
fixes  $v::rat\ vec$ 
assumes  $v \in of-int-hom.vec-hom\ L$ 
assumes  $dim-vec\ v = n$ 
shows  $(lattice-coord\ v)\ \$i \in \mathbb{Z}$ 
<proof>

```

6 Coord-Invariance

This section shows that the algorithm output matches true closest (or near-closest) vector in some trailing coordinates.

definition I where

$$I = (if\ (\{i \in \{0..<n\}. ((sq-norm\ (Mt!i)::rat)) \leq 4 * eps-closest\}::nat\ set) \neq \{\})$$

$$then\ Max\ (\{i \in \{0..<n\}. ((sq-norm\ (Mt!i)::rat)) \leq 4 * eps-closest\}::nat\ set)\ else$$

$$-1)$$

```

lemma I-geq:
shows  $I \geq -1$ 
<proof>

```

```

lemma I-leq:
shows  $I < n$ 
<proof>

```

```

lemma index-geq-I-big:
fixes  $i::nat$ 
assumes  $i > I$ 
assumes  $i < n$ 
shows  $((sq-norm\ (Mt!i)::rat)) > 4 * eps-closest$ 
<proof>

```

```

lemma scalar-prod-gs-from-lattice-coord:
fixes  $i::nat$ 
fixes  $v::rat\ vec$ 
assumes  $dim-vec\ v = n$ 

```

assumes $i < n$
shows $v \cdot Mt!i = \text{sum-list } (\text{map } (\lambda k. (\text{lattice-coord } v) \$ k * (((\text{RAT } M)!k) \cdot Mt!i))$
 $[i..<n])$
 $\langle \text{proof} \rangle$

lemma *correct-coord-help*:

fixes $i::\text{nat}$
assumes $i < (\text{int } n) - I$
assumes $\text{witness } v \text{ (eps-closest)}$
assumes $0 < i$
shows $(\text{lattice-coord } (s \ i)) \$ (n-i) = (\text{lattice-coord } v) \$ (n-i)$
 $\wedge ((s \ i) \cdot Mt!(n-i) = v \cdot Mt!(n-i))$
 $\langle \text{proof} \rangle$

lemma *correct-coord*:

fixes $v::\text{rat vec}$
fixes $k::\text{nat}$
assumes $\text{witness } v \text{ eps-closest}$
assumes $I < k$
assumes $k < n$
shows $(s \ n) \cdot Mt!(k) = v \cdot Mt!(k)$
 $\langle \text{proof} \rangle$

7 Bound on distance to target, with epsilon factor.

This section culminates in a bound (which includes the epsilon factor) on the output's distance to the target vector.

lemma *sq-norm-from-Mt*:

fixes $v::\text{rat vec}$
assumes $v\text{-carr}:v \in \text{carrier-vec } n$
shows $\text{sq-norm } v = \text{sum-list } (\text{map } (\lambda i. (v \cdot Mt!i) \hat{=} 2 / (\text{sq-norm } (Mt!i))) [0..<n])$
 $\langle \text{proof} \rangle$

lemma *basis-decay*:

fixes $i::\text{nat}$
fixes $j::\text{nat}$
assumes $i < n$
assumes $i+j < n$
shows $\text{sq-norm } (Mt!i) \leq \alpha \hat{=} j * \text{sq-norm}(Mt!(i+j))$
 $\langle \text{proof} \rangle$

lemma *basis-decay-cor*:

fixes $i::\text{nat}$
fixes $j::\text{nat}$
assumes $i < n$
assumes $j < n$
assumes $i \leq j$
shows $\text{sq-norm } (Mt!i) \leq \alpha \hat{=} n * \text{sq-norm}(Mt!j)$

<proof>

theorem *babai-correct:*

shows $\text{real-of-rat } ((\text{sq-norm } (s \ n)):\text{rat}) \leq (\text{real-of-rat } ((\text{rat-of-int } n)*\alpha^{\wedge}n) * \text{epsilon} * \text{closest-distance-sq}) \wedge s \ n \in \text{coset}$
<proof>

end

In this section we remove the arbitrary constant epsilon and clean up the assumptions and results.

locale *babai-with-assms* = *babai* +
fixes *mat-M* :: *rat mat*
assumes *basis: lin-indep M*
defines *mat-M* \equiv *mat-of-cols n (RAT M)*
assumes *reduced: weakly-reduced M n*
assumes *non-trivial: 0 < n*
assumes *alpha: $\alpha \geq 4/3$*
begin

sublocale *vec-space TYPE(rat) n* *<proof>*

definition *mat-M-inv* \equiv

(if (invertible-mat mat-M) then SOME B. (inverts-mat B mat-M) \wedge (inverts-mat mat-M B) else (0_m n n))

lemma *carrier-M:*

shows *mat-M* \in *carrier-mat n n*
<proof>

lemma *mat-M-inv-is-inv:*

shows *invertible-mat mat-M*
<proof>

abbreviation *babai-out* **where** *babai-out* \equiv *babai-of-LLL (-t) (RAT M)*

lemma *babai-with-assms-epsilon-connect:*

shows *babai-with-assms-epsilon M t α 2*
<proof>

This shows that the output, which is of the form $v-t$, with v in L , is short

lemma *babai-correct:*

shows $\text{real-of-rat } (\text{sq-norm } (\text{babai-out})) \leq (\text{real-of-rat } ((\text{rat-of-int } n)*\alpha^{\wedge}n) * \text{closest-distance-sq}) \wedge (\text{babai-out}) \in \text{coset}$
<proof>

This shifts the output, which is of the form $v-t$, with $v \in L$, back to v .

abbreviation *vL* **where** *vL* \equiv *map-vec int-of-rat (babai-out + t)*

lemma *shifted-babai-correct*:
shows $vL \in L$
 \wedge *real-of-rat* (*sq-norm* (*map-vec rat-of-int* ($vL - t$))) \leq
real-of-rat ((*rat-of-int* n)* α ^{\wedge} n)**closest-distance-sq*
 \langle *proof* \rangle

end

Here we prove correctness of the full algorithm, which starts with a non-reduced basis and outputs a vector in L close to t .

locale *full-babai-with-assms* =
fixes *target::rat vec*
fixes *n::nat*
fixes *fs-init::int vec list*
fixes $\alpha::rat$
assumes *non-triv:0 < n*
assumes *t-dim:dim-vec target = n*
assumes *LLL-assms:LLL.LLL-with-assms n n fs-init α*
begin

sublocale *vec-space TYPE(rat) n* \langle *proof* \rangle

abbreviation *B::real* **where** $B \equiv$ (*real-of-rat* (α ^{\wedge} n) * (n)) * *babai.closest-distance-sq*
fs-init target
abbreviation *out* **where** $out \equiv$ (*full-babai fs-init target α*)

lemma *LLL-output-babai-assms*:
shows *babai-with-assms* (*map-of-int-hom.vec-hom* (*LLL-Impl.reduce-basis* α *fs-init*))
target α
 \langle *proof* \rangle

lemma *full-babai-correct*:
shows *real-of-rat* (*sq-norm* ((*map-vec rat-of-int* out) - *target*))
 \leq *real-of-rat* (α) ^{\wedge} n * *real n* * *babai.closest-distance-sq fs-init target* \wedge
 $out \in$ *vec-module.lattice-of n fs-init*
 \langle *proof* \rangle
end

Here we prove correctness of the full algorithm, starting with a target vector of $TYPE(int)$.

lemma *full-babai-correct-int-target*:
assumes *full-babai-with-assms* (*map-vec rat-of-int target*) n *fs-init α*
shows *real-of-int* (*sq-norm* ((*full-babai fs-init* (*map-vec rat-of-int target*) α) -
target))
 \leq (*real-of-rat*(α) ^{\wedge} n * (n)) * *babai.closest-distance-sq fs-init* (*map-vec rat-of-int*
target) \wedge
(*full-babai fs-init* (*map-vec rat-of-int target*) α) \in *vec-module.lattice-of n*
fs-init

<proof>

The literature typically states the main result using 2^n as the approximation constant, rather than $\alpha^n * n$. Here we show that $\alpha^n * n$ is stronger for $\alpha = 4/3$.

lemma *clean-bound*:

fixes $n::nat$

shows $(4/3)^{\wedge}n * n \leq 2^{\wedge}n$

<proof>

end

theory *Misc-PMF*

imports

HOL-Probability.Probability

LLL-Basis-Reduction.LLL-Impl

begin

definition *replicate-spmf* :: $nat \Rightarrow 'b\ pmf \Rightarrow 'b\ list\ spmf$ **where**

replicate-spmf $m\ p = spmf\ of\ pmf\ (replicate\ pmf\ m\ p)$

The preceding *replicate-spmf* definition is copied from *CRYSTALS-Kyber-Security*. We do this to avoid loading the entire library. In fact, we do not need *replicate-spmf* at all for the HNP. However, for each *replicate-pmf* result we prove here, we also prove a corresponding *replicate-spmf* result. The *replicate-spmf* results are here for completeness, but not needed for the HNP.

8 SPMF and PMF helper lemmas

lemma *spmf-eq-element*: $spmf\ (p \gg (\lambda x. return\ spmf\ (x = t)))\ True = spmf\ p\ t$

<proof>

lemma *pmf-true-false*:

fixes $p :: 'a\ pmf$

fixes $P\ Q :: 'a \Rightarrow bool$

defines $a \equiv p \gg (\lambda x. return\ pmf\ (P\ x))$

defines $b \equiv p \gg (\lambda x. return\ pmf\ (\neg P\ x))$

shows $pmf\ a\ True = pmf\ b\ False$

<proof>

lemma *spmf-true-false*:

fixes $p :: 'a\ spmf$

fixes $P\ Q :: 'a \Rightarrow bool$

defines $a \equiv p \gg (\lambda x. return\ spmf\ (P\ x))$

defines $b \equiv p \gg (\lambda x. return\ spmf\ (\neg P\ x))$

shows $spmf\ a\ True = spmf\ b\ False$

<proof>

lemma pmf-subset:
fixes $p :: 'a \text{ pmf}$
fixes $P Q :: 'a \Rightarrow \text{bool}$
assumes $\forall x \in \text{set-pmf } p. P x \longrightarrow Q x$
shows $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (P x))) \text{ True} \leq \text{pmf } (p \gg (\lambda x. \text{return-pmf } (Q x))) \text{ True}$
 $\langle \text{proof} \rangle$

lemma pmf-subset':
fixes $p :: 'a \text{ pmf}$
fixes $P Q :: 'a \Rightarrow \text{bool}$
assumes $\forall x \in \text{set-pmf } p. \neg P x \longrightarrow \neg Q x$
shows $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (P x))) \text{ False} \leq \text{pmf } (p \gg (\lambda x. \text{return-pmf } (Q x))) \text{ False}$
 $\langle \text{proof} \rangle$

lemma spmf-subset:
fixes $p :: 'a \text{ spmf}$
fixes $P Q :: 'a \Rightarrow \text{bool}$
assumes $\forall x \in \text{set-spmf } p. P x \longrightarrow Q x$
shows $\text{spmf } (p \gg (\lambda x. \text{return-spmf } (P x))) \text{ True} \leq \text{spmf } (p \gg (\lambda x. \text{return-spmf } (Q x))) \text{ True}$
 $\langle \text{proof} \rangle$

lemma spmf-subset':
fixes $p :: 'a \text{ spmf}$
fixes $P Q :: 'a \Rightarrow \text{bool}$
assumes $\forall x \in \text{set-spmf } p. \neg P x \longrightarrow \neg Q x$
shows $\text{spmf } (p \gg (\lambda x. \text{return-spmf } (P x))) \text{ False} \leq \text{spmf } (p \gg (\lambda x. \text{return-spmf } (Q x))) \text{ False}$
 $\langle \text{proof} \rangle$

lemma pmf-in-set:
fixes $A :: 'a \text{ set}$
fixes $p :: 'a \text{ pmf}$
shows $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (x \in A))) \text{ True} = \text{prob-space.prob } p A$
 $\langle \text{proof} \rangle$

lemma pmf-of-prop:
fixes $P :: 'a \Rightarrow \text{bool}$
fixes $p :: 'a \text{ pmf}$
shows $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (P x))) \text{ True} = \text{prob-space.prob } p \{x \in \text{set-pmf } p. P x\}$
 $\langle \text{proof} \rangle$

lemma spmf-in-set:
fixes $A :: 'a \text{ set}$
fixes $p :: 'a \text{ spmf}$

shows $\text{pmf } (p \gg (\lambda x. \text{return-spmf } (x \in A))) \text{ True} = \text{prob-space.prob } p (\text{Some } 'A)$
 <proof>

lemma *spmf-of-prop*:

fixes $P :: 'a \Rightarrow \text{bool}$

fixes $p :: 'a \text{ spmf}$

shows $\text{pmf } (p \gg (\lambda x. \text{return-spmf } (P x))) \text{ True} = \text{prob-space.prob } p (\text{Some } \{x \in \text{set-spmf } p. P x\})$
 <proof>

lemma *replicate-pmf-events-helper*:

fixes $p :: 'a \text{ pmf}$

fixes $n P$

defines $\text{lhs} \equiv \text{pmf } (\text{pair-pmf } p (\text{replicate-pmf } n p) \gg (\lambda(x, xs). \text{return-pmf } (x \# xs))) \gg (\lambda xs. \text{return-pmf } (P xs)) \text{ True}$

defines $\text{rhs} \equiv \text{pmf } (\text{pair-pmf } p (\text{replicate-pmf } n p) \gg (\lambda(x, xs). \text{return-pmf } (P (x \# xs)))) \text{ True}$

shows $\text{lhs} = \text{rhs}$

<proof>

lemma *replicate-pmf-events*:

fixes $p :: 'a \text{ pmf}$

fixes $n :: \text{nat}$

fixes $E :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$

assumes $\forall i < n. \text{pmf } (p \gg (\lambda x. \text{return-pmf } (E i x))) \text{ True} = A i$

shows $\text{pmf } (\text{replicate-pmf } n p \gg (\lambda xs. \text{return-pmf } (\forall i < n. E i (xs!i)))) \text{ True} = (\prod_{i < n. A i}$

<proof>

lemma *replicate-pmf-same-event*:

fixes $p :: 'a \text{ pmf}$

fixes $n :: \text{nat}$

fixes $E :: 'a \Rightarrow \text{bool}$

assumes $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (E x))) \text{ True} = A$

shows $\text{pmf } (\text{replicate-pmf } n p \gg (\lambda xs. \text{return-pmf } (\forall i < n. E (xs!i)))) \text{ True} = A^{\wedge n}$

<proof>

lemma *replicate-pmf-same-event-leq*:

fixes $p :: 'a \text{ pmf}$

fixes $n :: \text{nat}$

fixes $E :: 'a \Rightarrow \text{bool}$

assumes $\text{pmf } (p \gg (\lambda x. \text{return-pmf } (E x))) \text{ True} \leq A$

shows $\text{pmf } (\text{replicate-pmf } n p \gg (\lambda xs. \text{return-pmf } (\forall i < n. E (xs!i)))) \text{ True} \leq A^{\wedge n}$

<proof>

lemma *replicate-spmf-events*:

fixes $p :: 'a \text{ spmf}$

fixes $n :: \text{nat}$
fixes $E :: \text{nat} \Rightarrow 'a \text{ option} \Rightarrow \text{bool}$
assumes $\forall i < n. (\text{spmf } (p \gg (\lambda x. \text{return-spmf } (E \ i \ x))) \ \text{True} = A \ i)$
shows $\text{spmf } (\text{replicate-spmf } \ n \ p \gg (\lambda xs. \text{return-spmf } (\forall i < n. E \ i \ (xs!i))))$
 $\text{True} = (\prod_{i < n}. A \ i)$
 $\langle \text{proof} \rangle$

lemma *replicate-spmf-same-event*:

fixes $p :: 'a \ \text{spmf}$
fixes $n :: \text{nat}$
fixes $E :: 'a \ \text{option} \Rightarrow \text{bool}$
assumes $\text{spmf } (p \gg (\lambda x. \text{return-spmf } (E \ x))) \ \text{True} = A$
shows $\text{spmf } (\text{replicate-spmf } \ n \ p \gg (\lambda xs. \text{return-spmf } (\forall i < n. E \ (xs!i)))) \ \text{True}$
 $= A^{\wedge n}$
 $\langle \text{proof} \rangle$

lemma *replicate-pmf-indep'*:

fixes $p :: 'a \ \text{pmf}$
fixes $n :: \text{nat}$
fixes $E :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$
defines $rp \equiv \text{replicate-pmf } \ n \ p$
assumes $\forall i < n. \text{pmf } (p \gg (\lambda x. \text{return-pmf } (E \ i \ x))) \ \text{True} = A \ i$
assumes $I \subseteq \{..<n\}$
assumes $\forall i < n. i \notin I \longrightarrow A \ i = 1$
shows $\text{pmf } (\text{replicate-pmf } \ n \ p \gg (\lambda xs. \text{return-pmf } (\forall i < n. E \ i \ (xs!i)))) \ \text{True}$
 $= (\prod_{i \in I}. A \ i)$
 $\langle \text{proof} \rangle$

lemma *replicate-pmf-indep''*:

fixes $p :: 'a \ \text{pmf}$
fixes $n :: \text{nat}$
fixes $E :: 'a \Rightarrow \text{bool}$
fixes $i :: \text{nat}$
defines $rp \equiv \text{replicate-pmf } \ n \ p$
defines $A \equiv \text{pmf } (p \gg (\lambda x. \text{return-pmf } (E \ x))) \ \text{True}$
assumes $i < n$
shows $\text{pmf } (\text{replicate-pmf } \ n \ p \gg (\lambda xs. \text{return-pmf } (E \ (xs!i)))) \ \text{True} = A$
 $\langle \text{proof} \rangle$

lemma *replicate-pmf-indep*:

fixes $p :: 'a \ \text{pmf}$
fixes $n :: \text{nat}$
fixes $E :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$
defines $rp \equiv \text{replicate-pmf } \ n \ p$
shows $\text{prob-space.indep-events } rp \ (\lambda i. \{xs \in rp. E \ i \ (xs!i)\}) \ \{..<n\}$
 $\langle \text{proof} \rangle$

lemma *replicate-spmf-same-event-leq*:

fixes $p :: 'a \ \text{spmf}$

```

fixes  $n :: \text{nat}$ 
fixes  $E :: 'a \text{ option} \Rightarrow \text{bool}$ 
assumes  $\text{spmf } (p \gg (\lambda x. \text{return-spmf } (E x))) \text{ True} \leq A$ 
shows  $\text{spmf } (\text{replicate-spmf } n p \gg (\lambda xs. \text{return-spmf } (\forall i < n. E (xs!i)))) \text{ True}$ 
 $\leq A^{\wedge n}$ 
<proof>

```

lemma *pmf-of-finite-set-event*:

```

fixes  $S :: 'a \text{ set}$ 
fixes  $p :: 'a \text{ pmf}$ 
fixes  $P :: 'a \Rightarrow \text{bool}$ 
defines  $p \equiv \text{pmf-of-set } S$ 
assumes  $S \neq \{\}$ 
assumes finite  $S$ 
shows  $\text{pmf } (p \gg (\lambda t. \text{return-pmf } (P t))) \text{ True} = (\text{card } (\{t \in S. P t\})) / \text{card } S$ 
<proof>

```

lemma *spmf-of-finite-set-event*:

```

fixes  $S :: 'a \text{ set}$ 
fixes  $p :: 'a \text{ spmf}$ 
fixes  $P :: 'a \Rightarrow \text{bool}$ 
defines  $p \equiv \text{spmf-of-set } S$ 
assumes finite  $S$ 
shows  $\text{spmf } (p \gg (\lambda t. \text{return-spmf } (P t))) \text{ True} = (\text{card } (\{t \in S. P t\})) / \text{card } S$ 
<proof>

```

end

theory *Hidden-Number-Problem*

imports

HOL-Number-Theory.Number-Theory

Babai-Correctness-Updated

Misc-PMF

begin

hide-fact *Finite-Cartesian-Product.mat-def*

hide-const *Finite-Cartesian-Product.mat*

hide-const *Finite-Cartesian-Product.row*

hide-fact *Finite-Cartesian-Product.row-def*

hide-const *Determinants.det*

hide-fact *Determinants.det-def*

hide-type *Finite-Cartesian-Product.vec*

hide-const *Finite-Cartesian-Product.vec*

hide-fact *Finite-Cartesian-Product.vec-def*

hide-const (**open**) *Finite-Cartesian-Product.transpose*

hide-fact (**open**) *Finite-Cartesian-Product.transpose-def*

unbundle *no inner-syntax*

unbundle *no vec-syntax*

hide-const (**open**) *Missing-List.span*
hide-const (**open**)
dependent
independent
real-vector.representation
real-vector.subspace
span
real-vector.extend-basis
real-vector.dim
hide-const (**open**) *orthogonal*
no-notation *fps-nth* (**infixl** \$ 75)

9 General helper lemmas

lemma *uminus-sq-norm*:
fixes $u\ v :: \text{rat vec}$
assumes $\text{dim-vec } u = \text{dim-vec } v$
shows $\text{sq-norm } (u - v) = \text{sq-norm } (v - u)$
<proof>

lemma *smult-sub-distrib-vec*:
assumes $v \in \text{carrier-vec } q\ w \in \text{carrier-vec } q$
shows $(a :: 'a :: \text{ring}) \cdot_v (v - w) = a \cdot_v v - a \cdot_v w$
<proof>

lemma *set-list-subset*:
fixes $l :: 'a \text{ list}$
fixes $S :: 'a \text{ set}$
assumes $\forall i \in \{0..<\text{length } l\}. l ! i \in S$
shows $\text{set } l \subseteq S$
<proof>

lemma *nat-subset-inf*:
fixes $A :: \text{int set}$
assumes $A \subseteq \mathbf{N}$
assumes $A \neq \{\}$
shows $\text{Inf } A \in A$
<proof>

lemma *cong-set-subseteq*:
fixes $a\ b\ m :: 'a :: \{\text{unique-euclidean-ring, abs}\}$
defines $S \equiv \{|a + z * m| \mid z. \text{True}\}$
defines $S' \equiv \{|b + z * m| \mid z. \text{True}\}$
assumes $[a = b] \pmod{m}$
shows $S \subseteq S'$
<proof>

lemma *cong-set-eq*:

```

fixes  $a\ b\ m :: 'a :: \{unique\text{-euclidean-ring}, abs\}$ 
defines  $S \equiv \{a + z * m \mid z. True\}$ 
defines  $S' \equiv \{b + z * m \mid z. True\}$ 
assumes  $[a = b] \text{ (mod } m)$ 
shows  $S = S'$ 
 $\langle proof \rangle$ 

```

```

context vec-module
begin

```

```

lemma sumlist-distrib:

```

```

fixes  $ys :: ('a::comm\text{-ring-1})\ \text{vec}\ \text{list}$ 
assumes  $\bigwedge w. List.member\ ys\ w \implies dim\text{-vec}\ w = n$ 
shows  $k \cdot_v\ \text{sumlist}\ ys = \text{sumlist}\ (\text{map}\ (\lambda i. k \cdot_v\ i)\ ys)$ 
 $\langle proof \rangle$ 

```

```

lemma smult-in-lattice-of:

```

```

fixes  $\text{lattice-basis} :: ('a::comm\text{-ring-1})\ \text{vec}\ \text{list}$ 
assumes  $\bigwedge w. List.member\ \text{lattice-basis}\ w \implies dim\text{-vec}\ w = n$ 
fixes  $\text{init-vec} :: ('a::comm\text{-ring-1})\ \text{vec}$ 
fixes  $k :: 'a::comm\text{-ring-1}$ 
assumes  $\text{init-vec} \in \text{lattice-of}\ \text{lattice-basis}$ 
shows  $k \cdot_v\ \text{init-vec} \in \text{lattice-of}\ ((\text{map}\ ((\cdot_v)\ k)\ \text{lattice-basis}))$ 
 $\langle proof \rangle$ 

```

```

end

```

```

lemma filter-distinct-sorted:

```

```

fixes  $l :: ('a::linorder)\ \text{list}$ 
fixes  $A :: 'a\ \text{set}$ 
defines  $P \equiv (\lambda x. x \in A)$ 
defines  $n \equiv \text{length}\ l$ 
assumes sorted  $l$ 
assumes distinct  $l$ 
assumes  $A \subseteq \text{set}\ l$ 
shows  $\text{filter}\ P\ l = \text{sorted-list-of-set}\ A$ 
 $\langle proof \rangle$ 

```

```

lemma filter-or:

```

```

fixes  $n\ a\ b$ 
fixes  $l :: \text{nat}\ \text{list}$ 
defines  $l \equiv [0..<n]$ 
defines  $P \equiv (\lambda x. x = a \vee x = b)$ 
assumes  $a < b$ 
assumes  $b < \text{length}\ l$ 

```

shows $\text{filter } P \ l = [a, b]$
 ⟨proof⟩

9.1 Casting lemmas

lemma *int-rat-real-casting-helper*:
assumes $a = \text{rat-of-int } b$
shows $\text{real-of-rat } a = \text{real-of-int } b$
 ⟨proof⟩

lemma *casting-expansion-aux*:
shows $\text{int-of-rat } (\text{rat-of-int } w1 + \text{rat-of-int } w2) = w1 + w2$
 ⟨proof⟩

lemma *casting-expansion*:
assumes $\text{dim-vec } w1 = \text{dim-vec } w2$
assumes $\exists w2\text{-vec. } w2 = \text{map-vec rat-of-int } w2\text{-vec}$
shows $\text{map-vec int-of-rat } ((\text{map-vec rat-of-int } w1) + w2) =$
 $\text{map-vec int-of-rat } (\text{map-vec rat-of-int } w1) + (\text{map-vec int-of-rat } w2)$
 ⟨proof⟩

lemma *casting-sum-aux*:
assumes $\text{dim-vec } k1 = \text{dim-vec } k2$
shows $(\text{map-vec rat-of-int } k1) + (\text{map-vec rat-of-int } k2) =$
 $\text{map-vec rat-of-int } (k1 + k2)$
 ⟨proof⟩

lemma *casting-sum-lemma*:
assumes $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies \text{dim-vec mem} = n$
assumes $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies$
 $(\exists k::\text{int vec. } \text{map-vec rat-of-int } k = \text{mem})$
shows $(\exists k::\text{int vec. } (\text{abelian-monoid.sumlist } (\text{module-vec TYPE}(\text{rat}) \ n) \ w) =$
 $\text{map-vec rat-of-int } k)$
 ⟨proof⟩

lemma *casting-lattice-aux*:
assumes $\exists k::\text{int vec. } \text{map-vec rat-of-int } k = v$
assumes $\text{map-vec int-of-rat } v =$
 $\text{map-vec int-of-rat } (\text{abelian-monoid.sumlist } (\text{module-vec TYPE}(\text{rat}) \ n) \ w)$
assumes $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies$
 $(\exists k::\text{int vec. } \text{map-vec rat-of-int } k = \text{mem})$
assumes $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies \text{dim-vec mem} = n$
shows $\text{map-vec int-of-rat } v =$
 $\text{abelian-monoid.sumlist } (\text{module-vec TYPE}(\text{int}) \ n)$
 $(\text{map } (\text{map-vec int-of-rat}) \ w)$
 ⟨proof⟩

lemma *in-lattice-casting*:

assumes $\bigwedge mem. List.member\ qs\ mem \implies (\exists k::int\ vec. map-vec\ rat-of-int\ k = mem)$

assumes $(\bigwedge mem. List.member\ qs\ mem \implies dim-vec\ mem = n)$

assumes $v \in vec-module.lattice-of\ n\ qs$

shows $\exists k. map-vec\ rat-of-int\ k = v$

<proof>

lemma *casting-lattice-lemma-aux2*:

assumes $\bigwedge mem. List.member\ qs\ mem \implies$

$(\exists k::int\ vec. map-vec\ rat-of-int\ k = mem)$

shows $(map\ (map-vec\ int-of-rat)\ (map\ (\lambda i. rat-of-int\ (c\ i)\ \cdot_v\ qs\ !\ i)\ [0..<length\ qs])) =$

$(map\ (\lambda i. of-int\ (c\ i)\ \cdot_v\ map\ (map-vec\ int-of-rat)\ qs\ !\ i)\ [0..<length\ qs])$

<proof>

lemma *casting-lattice-lemma*:

fixes $v :: rat\ vec$

fixes $qs :: rat\ vec\ list$

assumes $\exists k::int\ vec. map-vec\ rat-of-int\ k = v$

assumes $\bigwedge mem. List.member\ qs\ mem \implies$

$(\exists k::int\ vec. map-vec\ rat-of-int\ k = mem)$

assumes $dim-vecs: \bigwedge mem. List.member\ qs\ mem \implies dim-vec\ mem = n$

assumes $v \in vec-module.lattice-of\ n\ qs$

shows $map-vec\ int-of-rat\ v$

$\in vec-module.lattice-of\ n\ (map\ (map-vec\ int-of-rat)\ qs)$

<proof>

10 HNP adversary locale

locale *hnp-adversary* =

fixes $d\ p :: nat$

begin

type-synonym *adversary* = $(nat \times nat)\ list \Rightarrow nat$

definition *int-gen-basis* :: $nat\ list \Rightarrow int\ vec\ list$ **where**

$int-gen-basis\ ts = map\ (\lambda i. (p^{\wedge}2\ \cdot_v\ (unit-vec\ (d+1)\ i)))\ [0..<d]$
 $\quad @\ [vec-of-list\ ((map\ (\lambda x. (of-nat\ p)\ * x)\ ts)\ @\ [1])]$

fun *int-to-nat-residue* :: $int \Rightarrow nat \Rightarrow nat$

where *int-to-nat-residue* $I\ modulus = nat\ (I\ mod\ modulus)$

definition *ts-from-pairs* :: $(nat \times nat)\ list \Rightarrow nat\ list$ **where**

ts-from-pairs $pairs = map\ fst\ pairs$

definition *scaled-uvec-from-pairs* :: $(nat \times nat)\ list \Rightarrow int\ vec$ **where**

*scaled-uvec-from-pairs pairs = vec-of-list ((map ($\lambda(a,b). p*b$) pairs) @ [0])*

fun *A-vec* :: (*nat* × *nat*) *list* ⇒ *int vec* **where**
A-vec pairs = (*full-babai*
(int-gen-basis (ts-from-pairs pairs))
(map-vec rat-of-int (scaled-uvec-from-pairs pairs))
(4/3))

fun *A* :: *adversary* **where**
A pairs = *int-to-nat-residue ((A-vec pairs)\$d) p*

end

11 HNP locales

11.0.1 Arithmetic locale

locale *hnp-arith* =
fixes *n* α *d p k* :: *nat*
assumes *p*: *prime p*
assumes α : $\alpha \in \{1..<p\}$
assumes *d*: $d = 2 * \text{ceiling} (\text{sqrt } n)$
assumes *n*: $n = \text{ceiling} (\log 2 p)$
assumes *k*: $k = \text{ceiling} (\text{sqrt } n) + \text{ceiling} (\log 2 n)$
assumes *n-big*: $961 < n$
begin

definition μ :: *nat* **where**
 $\mu \equiv \text{nat} (\text{ceiling} (1/2 * (\text{sqrt } n)) + 3)$

lemma μ : $\mu = (\text{ceiling} (1/2 * (\text{sqrt } n)) + 3)$
 $\langle \text{proof} \rangle$

lemma *int-p-prime*: *prime (int p)* $\langle \text{proof} \rangle$

lemma *p-geq-2*: $p \geq 2$ $\langle \text{proof} \rangle$

lemma *n-geq-1*: $n \geq 1$
 $\langle \text{proof} \rangle$

lemma μ -le-k:
shows $\mu + 1 \leq k$
 $\langle \text{proof} \rangle$

lemma *k-plus-1-lt*:
shows $k + 1 < \log 2 p$
 $\langle \text{proof} \rangle$

lemma *p-k-le-p-mu*:

shows $p/(2^k) \leq p/(2^{\mu+1})$
 ⟨proof⟩

lemma *k-geq-1*: $k \geq 1$ ⟨proof⟩

lemma *final-ineq*:

$((4::\text{real})/3) \text{ powr } ((d+1)/2) * (11/10) * (d+1) * (p / (2 \text{ powr } (\text{real } k - 1)))$
 $< p / (2 \text{ powr } \mu)$
 ⟨proof⟩

end

11.1 Main HNP locale

locale *hnp* = *hnp-arith* $n \ \alpha \ d \ p \ k$ + *hnp-adversary* $d \ p$ **for** $n \ \alpha \ d \ p \ k$ +
fixes *msb-p* :: $\text{nat} \Rightarrow \text{nat}$
assumes *msb-p-dist*: $\bigwedge x. |\text{int } x - \text{int } (\text{msb-p } x)| < p / (2^k)$
begin

11.2 Uniqueness lemma

sublocale *vec-space* *TYPE*(*rat*) $d + 1$ ⟨proof⟩

lemma *sumlist-index-commute*:

fixes *Lst* :: $\text{rat } \text{vec } \text{list}$

fixes *i* :: nat

assumes *set Lst* \subseteq *carrier-vec* $(d + 1)$

assumes $i < (d + 1)$

shows $(\text{sumlist } Lst)\$i = \text{sum-list } (\text{map } (\lambda j. (Lst!\$j)\$i) [0..<(\text{length } Lst)])$

⟨proof⟩

definition *ts-pmf* **where** *ts-pmf* = *replicate-pmf* d (*pmf-of-set* $\{1..<p\}$)

lemma *set-pmf-ts*: *set-pmf* *ts-pmf* = $\{l. \text{length } l = d \wedge (\forall i < d. !i \in \{1..<p\})\}$
 ⟨proof⟩

definition *ts-to-as* :: $\text{nat } \text{list} \Rightarrow \text{nat } \text{list}$ **where**

ts-to-as *ts* = $(\text{map } (\text{msb-p} \circ (\lambda t. (\alpha * t) \text{ mod } p)) \text{ ts})$

definition *ts-to-u* :: $\text{nat } \text{list} \Rightarrow \text{rat } \text{vec}$ **where**

ts-to-u *ts* = *vec-of-list* $(\text{map of-nat } (\text{ts-to-as } \text{ts}) @ [0])$

lemma *ts-to-u-alt*:

ts-to-u *ts* = *vec-of-list* $((\text{map } (\text{of-nat} \circ \text{msb-p} \circ (\lambda t. (\alpha * t) \text{ mod } p)) \text{ ts}) @ [0])$

⟨proof⟩

lemma *u-carrier*: $\text{length } \text{ts} = d \implies \text{ts-to-u } \text{ts} \in \text{carrier-vec } (d + 1)$
 ⟨proof⟩

lemma *ts-to-u-carrier*:
fixes $ts :: \text{nat list}$
shows $(ts\text{-to-}u\ ts) \in \text{carrier-vec } ((\text{length } ts) + 1)$
 $\langle \text{proof} \rangle$

11.2.1 Lattice construction and lemmas

definition *p-vecs* $:: \text{rat vec list}$ **where**
 $p\text{-vecs} = \text{map } (\lambda i. \text{of-int-hom.vec-hom } ((\text{of-nat } p) \cdot_v (\text{unit-vec } (d+1) i))) [0..<d]$

lemma *length-p-vecs*: $\text{length } p\text{-vecs} = d$ $\langle \text{proof} \rangle$

lemma *p-vecs-carrier*: $\forall v \in \text{set } p\text{-vecs}. \text{dim-vec } v = d + 1$ $\langle \text{proof} \rangle$

lemma *lincomb-of-p-vecs-last*: $(\text{lincomb-list } (\text{of-int} \circ \text{cs}) p\text{-vecs})\$d = 0$ (**is** $?lhs = 0$)
 $\langle \text{proof} \rangle$

definition *gen-basis* $:: \text{nat list} \Rightarrow \text{rat vec list}$ **where**
 $\text{gen-basis } ts = p\text{-vecs} @ [\text{vec-of-list } ((\text{map } \text{of-nat } ts) @ [1 / (\text{of-nat } p)])]$

lemma *gen-basis-length*: $\text{length } (\text{gen-basis } ts) = d + 1$ $\langle \text{proof} \rangle$

lemma *gen-basis-units*:
assumes $i < d$
assumes $j < d + 1$
shows $((\text{gen-basis } ts)!i)\$j = \text{of-nat } (\text{if } i = j \text{ then } p \text{ else } 0)$ (**is** $(?x)\$j = -$)
 $\langle \text{proof} \rangle$

definition *int-gen-lattice* $:: \text{nat list} \Rightarrow \text{int vec set}$ **where**
 $\text{int-gen-lattice } ts = \text{vec-module.lattice-of } (d + 1) (\text{int-gen-basis } ts)$

definition *gen-lattice* $:: \text{nat list} \Rightarrow \text{rat vec set}$ **where**
 $\text{gen-lattice } ts = \text{vec-module.lattice-of } (d + 1) (\text{gen-basis } ts)$

definition *close-vec* $:: \text{nat list} \Rightarrow \text{rat vec} \Rightarrow \text{bool}$ **where**
 $\text{close-vec } ts\ v \iff (\text{sq-norm } ((\text{ts-to-}u\ ts) - v) < ((\text{of-nat } p) / 2^{\wedge}\mu)^{\wedge}2)$

definition *good-vec* $:: \text{rat vec} \Rightarrow \text{bool}$ **where**
 $\text{good-vec } v \iff \text{dim-vec } v = d + 1 \wedge (\exists \beta :: \text{int}. [\alpha = \beta] (\text{mod } p) \wedge \text{of-rat } (v\$d) = \beta/p)$

definition *good-lattice* $:: \text{nat list} \Rightarrow \text{bool}$ **where**
 $\text{good-lattice } ts \iff (\forall v \in \text{gen-lattice } ts. \text{close-vec } ts\ v \longrightarrow \text{good-vec } v)$

definition *bad-lattice* $:: \text{nat list} \Rightarrow \text{bool}$ **where**
 $\text{bad-lattice } ts \iff \neg \text{good-lattice } ts$

definition *sampled-lattice-good* $:: \text{bool pmf}$ **where**

```

sampled-lattice-good = do {
  ts ← ts-pmf;
  return-pmf (good-lattice ts)
}

```

interpretation *vec-int*: *vec-module TYPE(int) d + 1* *<proof>*

lemma *int-gen-basis-carrier*:
fixes *ts* :: *nat list*
assumes *length ts = d*
shows *set (int-gen-basis ts) ⊆ carrier-vec (d + 1)*
<proof>

lemma *int-gen-lattice-carrier*:
fixes *ts* :: *nat list*
assumes *length ts = d*
shows *int-gen-lattice ts ⊆ carrier-vec (d + 1)*
<proof>

lemma *gen-basis-vecs-carrier*:
fixes *ts* :: *nat list*
fixes *i* :: *nat*
assumes *length ts = d*
assumes *i ∈ {0..< d + 1}*
shows (*gen-basis ts*) ! *i* ∈ *carrier-vec (d + 1)*
<proof>

lemma *gen-basis-carrier*:
fixes *ts* :: *nat list*
assumes *length ts = d*
shows *set (gen-basis ts) ⊆ carrier-vec (d + 1)*
<proof>

lemma *gen-lattice-carrier*:
fixes *ts* :: *nat list*
assumes *length ts = d*
shows *gen-lattice ts ⊆ carrier-vec (d + 1)*
<proof>

lemma *sampled-lattice-good-map-pmf*: *sampled-lattice-good = map-pmf good-lattice ts-pmf*
<proof>

lemma *coordinates-of-gen-lattice*:
fixes *ts* :: *nat list*
fixes *c* :: *nat ⇒ int*
fixes *i* :: *nat*
assumes *i ≤ length ts*

assumes $length\ ts = d$
shows $(sumlist\ (map\ (\lambda i. of-int\ (c\ i) \cdot_v\ ((gen-basis\ ts)\ !\ i))\ [0\ ..<\ length\ (gen-basis\ ts)]))\i
 $= (if\ (i = d)\ then\ (rat-of-int\ (c\ d)\ /\ rat-of-nat\ p)\ else\ rat-of-int\ ((c\ d)\ *\ ts!\ i + (c\ i)*p))$
 $\langle proof \rangle$

lemma *gen-lattice-int-gen-lattice-vec*:

fixes $scaled-v :: int\ vec$
fixes $ts :: nat\ list$
assumes $length\ ts = d$
assumes $(scaled-v \in int-gen-lattice\ ts)$
shows $((1/(of-nat\ p)) \cdot_v\ (map-vec\ rat-of-int\ scaled-v) \in (gen-lattice\ ts))$
 $\langle proof \rangle$

definition *t-vec* $:: nat\ list \Rightarrow rat\ vec$ **where**

$t-vec\ ts = vec-of-list\ ((map\ of-nat\ ts)\ @\ [1\ /\ (of-nat\ p)])$

lemma *t-vec-dim*: $dim-vec\ (t-vec\ ts) = length\ ts + 1$

$\langle proof \rangle$

lemma *t-vec-last*: $length\ ts = d \Longrightarrow (t-vec\ ts)\$d = 1\ /\ (of-nat\ p)$

$\langle proof \rangle$

definition *z-vecs* $:: int\ vec\ set$ **where**

$z-vecs = \{v. dim-vec\ v = d + 1 \wedge v\$d = 0\}$

definition *vec-class* $:: nat\ list \Rightarrow int \Rightarrow rat\ vec\ set$ **where**

$vec-class\ ts\ \beta = \{(of-int\ \beta) \cdot_v\ (t-vec\ ts) + lincomb-list\ (of-int\ \circ\ cs)\ p-vecs\ |\ cs :: nat \Rightarrow int. True\}$

definition *vec-class-mod-p* $:: nat\ list \Rightarrow int \Rightarrow rat\ vec\ set$ **where**

$vec-class-mod-p\ ts\ \beta = \bigcup \{vec-class\ ts\ \beta' \mid \beta'. [\beta = \beta']\ (mod\ p)\}$

lemma *vec-class-carrier*:

assumes $length\ ts = d$

shows $vec-class\ ts\ \beta \subseteq carrier-vec\ (d + 1)$

$\langle proof \rangle$

lemma *vec-class-mod-p-carrier*:

assumes $length\ ts = d$

shows $vec-class-mod-p\ ts\ \beta \subseteq carrier-vec\ (d + 1)$

$\langle proof \rangle$

lemma *vec-class-last*:

assumes $length\ ts = d$

assumes $v \in vec-class\ ts\ \beta$

shows $v\$d = rat-of-int\ \beta\ /\ rat-of-int\ p$

$\langle proof \rangle$

lemma *gen-lattice-int-gen-lattice-vec'*:
fixes $v :: \text{rat vec}$
fixes $ts :: \text{nat list}$
assumes $\text{length } ts = d$
assumes $v \in \text{gen-lattice } ts$
shows $\text{map-vec int-of-rat } ((\text{of-nat } p) \cdot_v v) \in \text{int-gen-lattice } ts$
 $\text{map-vec rat-of-int } (\text{map-vec int-of-rat } ((\text{of-nat } p) \cdot_v v)) = (\text{rat-of-nat } p) \cdot_v v$
 $\langle \text{proof} \rangle$

lemma *gen-lattice-int-gen-lattice-closest*:
fixes $\text{scaled-}v :: \text{int vec}$
fixes $u :: \text{rat vec}$
fixes $ts :: \text{nat list}$
assumes $\text{length } ts = d$
assumes $\text{dim-vec } u = d + 1$
shows $\text{real}(p^2) * \text{Inf}\{\text{real-of-rat } (\text{sq-norm } (x - u)) \mid x \in \text{gen-lattice } ts\}$
 $= \text{babai.closest-distance-sq } (\text{int-gen-basis } ts) ((\text{rat-of-nat } p) \cdot_v u)$
 $\langle \text{proof} \rangle$

lemma *close-vector-exists*:
fixes $ts :: \text{nat list}$
assumes $\text{length } ts = d$
shows $\exists w \in (\text{gen-lattice } ts). \text{sq-norm } ((\text{ts-to-u } ts) - w) \leq (\text{of-nat } ((d+1) * p^2)) / 2^{(2*k)}$
 $\langle \text{proof} \rangle$

lemma *gen-lattice-dim*:
assumes $ts \in \text{set-pmf } ts\text{-pmf}$
assumes $v \in \text{gen-lattice } ts$
shows $\text{dim-vec } v = d + 1$
 $\langle \text{proof} \rangle$

lemma *vec-class-union*:
fixes $ts :: \text{nat list}$
assumes $ts \in \text{set-pmf } ts\text{-pmf}$
defines $L \equiv \text{gen-lattice } ts$
shows $L = \bigcup \{\text{vec-class } ts \beta \mid \beta. \text{True}\}$
 $\langle \text{proof} \rangle$

lemma *vec-class-mod-p-union*:
fixes $ts :: \text{nat list}$
assumes $ts \in \text{set-pmf } ts\text{-pmf}$
defines $L \equiv \text{gen-lattice } ts$
shows $L = \bigcup \{\text{vec-class-mod-}p \text{ } ts \beta \mid \beta. \beta \in \{0..<p::\text{int}\}\} (\text{is } - = ?\text{rhs})$
 $\langle \text{proof} \rangle$

11.2.2 dist-p definition and lemmas

definition $dist-p :: int \Rightarrow int \Rightarrow int$ **where**
 $dist-p\ i\ j = Inf\ \{abs\ (i - j + z * (of-nat\ p)) \mid z.\ True\}$

lemma $dist-p$ -well-defined:
fixes $i :: int$
fixes $j :: int$
shows $dist-p\ i\ j \in \{abs\ (i - j + z * (of-nat\ p)) \mid z.\ True\}$ (**is** $- \in ?S$)
(*proof*)

lemma $dist-p$ -set-bdd-below:
fixes $i\ j :: int$
shows $\forall x \in \{abs\ (i - j + z * (of-nat\ p)) \mid z.\ True\}.\ 0 \leq x$
(*proof*)

lemma $dist-p$ -le:
fixes $i\ j :: int$
shows $\forall x \in \{abs\ (i - j + z * (of-nat\ p)) \mid z.\ True\}.\ dist-p\ i\ j \leq x$ (**is** $\forall x \in ?S.$
 $-$)
(*proof*)

lemma $dist-p$ -equiv:
fixes $i :: int$
fixes $j :: int$
shows $[dist-p\ i\ j = i - j] (mod\ p) \vee [dist-p\ i\ j = j - i] (mod\ p)$
(*proof*)

lemma $dist-p$ -equiv':
fixes $i :: int$
fixes $j :: int$
shows $dist-p\ i\ j = min\ ((i - j)\ mod\ p)\ ((j - i)\ mod\ p)$
(*proof*)

lemma $dist-p$ -equiv'':
fixes $i :: int$
fixes $j :: int$
shows $dist-p\ i\ j = ((i - j)\ mod\ p) \vee dist-p\ i\ j = ((j - i)\ mod\ p)$
(*proof*)

lemma $dist-p$ -instances-help:
fixes $i :: int$
fixes $j :: int$
fixes $dist :: int$
assumes $coprime\ (i - j)\ p$
shows $card\ \{t \in \{1..<p\}.\ (dist-p\ (t*i)\ (t*j)) = dist\} \leq 2$
(*proof*)

lemma $dist-p$ -nat:
fixes $i :: int$

fixes $j :: int$
shows $dist-p\ i\ j \in \mathbb{N}$
 $\langle proof \rangle$

lemma $dist-p-nat'$:
shows $nat\ (dist-p\ i\ j) = dist-p\ i\ j$
 $\langle proof \rangle$

lemma $dist-p-instances$:
fixes $i :: int$
fixes $j :: int$
fixes $bound :: rat$
assumes $i \neq j$
assumes $coprime\ (i - j)\ p$
assumes $0 < bound$
shows $rat-of-nat\ (card\ \{t \in \{1..<p\}. rat-of-int\ (dist-p\ (t*i)\ (t*j)) \leq bound\}) \leq 2*bound$
 $\langle proof \rangle$

lemma $dist-p-smallest$:
fixes $\beta\ ts\ i$
assumes $ts \in set-pmf\ ts-pmf$
assumes $v \in vec-class-mod-p\ ts\ \beta$
assumes $i < d$
defines $u \equiv ts-to-u\ ts$
shows $(of-int\ (dist-p\ (int\ (ts\ !\ i) * \beta)\ (int\ (ts-to-as\ ts\ !\ i))))^2 \leq ((u - v)\$i)^2$
 $(is\ (of-int\ ?d)^2 \leq -)$
 $\langle proof \rangle$

lemma $dist-p-diff-helper$:
fixes $a\ b\ b'$
assumes $b \geq b'$
shows $|dist-p\ a\ b - dist-p\ a\ b'| \leq b - b'$
 $\langle proof \rangle$

lemma $dist-p-diff$:
fixes $a\ b\ b'$
shows $|dist-p\ a\ b - dist-p\ a\ b'| \leq |b - b'|$
 $\langle proof \rangle$

11.2.3 Uniqueness lemma argument

definition $good-beta\ where$

$good-beta\ ts\ \beta \longleftrightarrow (\forall v \in vec-class-mod-p\ ts\ \beta. close-vec\ ts\ v \longrightarrow good-vec\ v)$

definition $bad-beta\ where$

$bad-beta\ ts\ \beta \longleftrightarrow \neg good-beta\ ts\ \beta$

definition $some-bad-beta\ where$

$\text{some-bad-beta } ts \longleftrightarrow (\exists \beta \in \{0..<p::int\}. \text{bad-beta } ts \ \beta)$

definition *bad-beta-union* **where**

$\text{bad-beta-union} = (\bigcup \beta \in \{0..<p::int\}. \{ts. \text{bad-beta } ts \ \beta\})$

lemma *reduction-1*:

assumes $ts \in \text{set-pmf } ts\text{-pmf}$

assumes $\neg \text{good-lattice } ts$

shows $\text{some-bad-beta } ts$

$\langle \text{proof} \rangle$

lemma *reduction-1-pmf*:

$\text{pmf sampled-lattice-good False} \leq \text{pmf } (ts\text{-pmf} \gg (\lambda ts. \text{return-pmf } (\text{some-bad-beta } ts))) \text{ True}$

$\langle \text{proof} \rangle$

lemma *reduction-2*: $\{ts. \text{some-bad-beta } ts\} \subseteq \text{bad-beta-union}$

$\langle \text{proof} \rangle$

lemma *reduction-2-pmf*:

$\text{pmf } (ts\text{-pmf} \gg (\lambda ts. \text{return-pmf } (ts \in \{ts. \text{some-bad-beta } ts\}))) \text{ True}$

$\leq \text{pmf } (ts\text{-pmf} \gg (\lambda ts. \text{return-pmf } (ts \in \text{bad-beta-union}))) \text{ True}$

$\langle \text{proof} \rangle$

lemma *bad-beta-union-bound*:

$\text{pmf } (ts\text{-pmf} \gg (\lambda ts. \text{return-pmf } (ts \in \text{bad-beta-union}))) \text{ True}$

$\leq (\sum \beta \in \{0..<p::int\}. \text{pmf } (ts\text{-pmf} \gg (\lambda ts. \text{return-pmf } (ts \in \{ts. \text{bad-beta } ts \ \beta\})))) \text{ True}$

(is ?lhs \leq ?rhs)

$\langle \text{proof} \rangle$

lemma *fixed-beta-close-vec-dist-p*:

fixes $\beta \ ts$

assumes $ts \in \text{set-pmf } ts\text{-pmf}$

assumes $v \in \text{vec-class-mod-p } ts \ \beta$

assumes $\text{close-vec } ts \ v$

defines $u \equiv ts\text{-to-}u \ ts$

shows $\forall i < d. \text{real-of-int } (dist\text{-}p \ (int \ (ts \ ! \ i) \ * \ \beta) \ (int \ (ts\text{-to-}u \ ts \ ! \ i))) < \text{real } p / 2^{\wedge} \mu$

$\langle \text{proof} \rangle$

lemma *fixed-beta-bad-prob-arith-helper*:

defines $T\text{-lwr-bnd} \equiv (\text{rat-of-nat } (p - 1) - 2 * (2 * (\text{rat-of-nat } p)) / (2^{\wedge} \mu))$

shows $1 - 5 / (2^{\wedge} \mu) \leq \text{real-of-rat } T\text{-lwr-bnd} / (p - 1)$

$\langle \text{proof} \rangle$

lemma *dist-p-helper*:

fixes $ts \ i$

assumes $ts: ts \in \text{set-pmf } ts\text{-pmf}$

assumes $i: i < d$
assumes $dist$: $dist-p (int (ts!i) * \beta) ((ts-to-as ts)!i) < p/(2^{\widehat{\mu}})$
shows $dist-p ((ts!i) * \beta) ((ts!i) * \alpha) \leq (2*p)/(2^{\widehat{\mu}})$
 $\langle proof \rangle$

lemma *prob-A-helper*:

fixes $\beta :: int$
defines $[simp]$: $t-pmf \equiv pmf-of-set \{1..<p\}$
defines $[simp]$: $A-cond \equiv \lambda t. (2*p)/(2^{\widehat{\mu}}) < dist-p (\beta * t) (\alpha * t)$
defines $[simp]$: $A-pmf \equiv t-pmf \gg (\lambda t. return-pmf (A-cond t))$
defines $[simp]$: $A \equiv pmf A-pmf True$
assumes $\beta: \beta \in \{0..<p::int\}$
assumes *False*: $\neg (\beta = \alpha \bmod p)$
shows $(1 - A) \leq (5 / (2^{\widehat{\mu}}))$
 $\langle proof \rangle$

lemma *prob-A-helper'*:

fixes $\beta :: int$
defines $[simp]$: $t-pmf \equiv pmf-of-set \{1..<p\}$
defines $[simp]$: $A-cond \equiv \lambda t. (2*p)/(2^{\widehat{\mu}}) < dist-p (\beta * t) (\alpha * t)$
defines $[simp]$: $A-pmf \equiv t-pmf \gg (\lambda t. return-pmf (A-cond t))$
defines $[simp]$: $A \equiv pmf A-pmf True$
assumes $\beta: \beta \in \{0..<p::int\}$
assumes *False*: $\neg (\beta = \alpha \bmod p)$
shows $(1 - A)^d \leq (5 / (2^{\widehat{\mu}}))^d$
 $\langle proof \rangle$

lemma *fixed-beta-bad-prob*:

fixes β
assumes $\beta \in \{0..<p::int\}$
defines $M \equiv ts-pmf \gg (\lambda ts. return-pmf (ts \in \{ts. bad-beta ts \beta\}))$
shows $\beta = \alpha \bmod p \implies pmf M True = 0$
 $\neg (\beta = \alpha \bmod p) \implies pmf M True \leq (5/(2^{\widehat{\mu}}))^d$
 $\langle proof \rangle$

lemma *bad-beta-union-bound-pmf*:

$pmf (ts-pmf \gg (\lambda ts. return-pmf (ts \in bad-beta-union))) True \leq (p - 1) * (5/(2^{\widehat{\mu}}))^d$
 $\langle proof \rangle$

lemma *sampled-lattice-unlikely-bad*:

shows $pmf sampled-lattice-good False \leq (p - 1) * ((5/(2^{\widehat{\mu}}))^d)$
 $\langle proof \rangle$

11.2.4 Main uniqueness lemma statement

lemma *sampled-lattice-likely-good*: $pmf sampled-lattice-good True \geq 1/2$

$\langle proof \rangle$

11.3 Main theorem

11.3.1 Oracle definition

definition $\mathcal{O} :: \text{nat} \Rightarrow \text{nat}$ **where**

$$\mathcal{O} t = \text{msb-}p ((\alpha * t) \bmod p)$$

11.3.2 Apply Babai lemmas to adversary

We use Babai lemmas to show that the adversary wins when the ts generate a good lattice.

lemma *gen-basis-assms*:

fixes $ts :: \text{nat list}$

assumes $\text{length } ts = d$

shows $LLL.LLL\text{-with-assms } (d+1) (d+1) (\text{int-gen-basis } ts) (4/3)$

<proof>

definition $u\text{-vec-is-msbs} :: (\text{nat} \times \text{nat}) \text{ list} \Rightarrow \text{bool}$ **where**

$$u\text{-vec-is-msbs } pairs = ((\text{of-nat } p) \cdot_v ts\text{-to-}u (ts\text{-from-pairs } pairs) = \text{of-int-hom.vec-hom} (\text{scaled-uvec-from-pairs } pairs))$$

lemma *updated-full-Babai-correct-int-target*:

assumes $\text{full-babai-with-assms } (\text{map-vec rat-of-int target}) (d + 1) \text{fs-init } (4/3)$

shows $\text{real-of-int } (\text{sq-norm } ((\text{full-babai fs-init } (\text{map-vec rat-of-int target}) (4/3)) - \text{target}))$

$$\leq ((4/3)^\wedge(d + 1) * (d + 1)) * 11/10 * \text{babai.closest-distance-sq fs-init} (\text{map-vec rat-of-int target}) \wedge$$

$(\text{full-babai fs-init } (\text{map-vec rat-of-int target}) (4/3)) \in \text{vec-module.lattice-of } (d + 1) \text{fs-init}$

<proof>

lemma *ad-output-vec-close*:

fixes $pairs :: (\text{nat} \times \text{nat}) \text{ list}$

assumes $\text{length } pairs = d$

assumes $u\text{-vec-is-msbs } pairs$

shows $\text{real-of-int}(\text{sq-norm}$

$$((\mathcal{A}\text{-vec } pairs)$$

$$- (\text{scaled-uvec-from-pairs } pairs))) \leq$$

$$(4 / 3)^\wedge(d + 1) * \text{real } (d + 1) * 11 / 10 * p^\wedge 2 * (\text{of-nat } ((d+1) * p^\wedge 2)) / 2^\wedge(2*k)$$

$$\wedge (\mathcal{A}\text{-vec } pairs) \in \text{int-gen-lattice } (ts\text{-from-pairs } pairs)$$

<proof>

lemma *ad-output-vec-class*:

fixes $pairs :: (\text{nat} \times \text{nat}) \text{ list}$

assumes $\text{length } pairs = d$

assumes $u\text{-vec-is-msbs } pairs$

shows $\text{sq-norm } ((1 / (\text{rat-of-nat } p)) \cdot_v (\text{map-vec rat-of-int } (\mathcal{A}\text{-vec } pairs)) - (ts\text{-to-}u (ts\text{-from-pairs } pairs))) < ((\text{of-nat } p) / 2^\wedge(\mu))^\wedge 2$

$\wedge ((1/(\text{rat-of-nat } p)) \cdot_v (\text{map-vec rat-of-int } (\mathcal{A}\text{-vec pairs})) \in \text{vec-class-mod-}p$
 $(\text{ts-from-pairs pairs}) (\mathcal{A}\text{ pairs}))$
 $\langle \text{proof} \rangle$

If we get a good lattice (which is likely), the adversary finds the hidden number

lemma *hnp-adversary-exists-helper*:
assumes $ts \in \text{set-pmf ts-pmf}$
defines $ts\text{-Ots} \equiv \text{map } (\lambda t. (t, \mathcal{O} t)) ts$
assumes *good-lattice ts*
shows $\alpha = \mathcal{A} ts\text{-Ots}$
 $\langle \text{proof} \rangle$

11.3.3 Main theorem statement

The cryptographic game which the adversary should win with high probability.

definition *game* :: *adversary* \Rightarrow *bool pmf* **where**
 $game \mathcal{A}' = do \{$
 $ts \leftarrow \text{replicate-pmf } d (\text{pmf-of-set } \{1..<p\});$
 $return\text{-pmf } (\alpha = \mathcal{A}' (\text{map } (\lambda t. (t, \mathcal{O} t)) ts))$
 $\}$

The adversary finds the hidden number with probability at least 1/2.

theorem *hnp-adversary-exists*: $\text{pmf } (game \mathcal{A}) \text{ True} \geq 1/2$
 $\langle \text{proof} \rangle$

Alternative definition of *game*, written as a *map-pmf*

definition *game'* :: *adversary* \Rightarrow *bool pmf* **where**
 $game' \mathcal{A}' = \text{map-pmf } ((\lambda ts. (\alpha = \mathcal{A}' (\text{map } (\lambda t. (t, \mathcal{O} t)) ts)))) (\text{replicate-pmf } d$
 $(\text{pmf-of-set } \{1..<p\}))$

lemma $game' = game$
 $\langle \text{proof} \rangle$

end

12 Some MSB instantiations and lemmas

12.1 Bit-shift MSB

definition *msb* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* **where**
 $msb k n x \equiv (x \text{ div } 2^{(n-k)}) * 2^{(n-k)}$

lemma *msb-dist*:
fixes $k n :: nat$
assumes $k < n$
assumes $1 \leq k$

shows $|int(x) - int(msb\ k\ n\ x)| < 2^{\wedge}n / (2^{\wedge}k)$
 $\langle proof \rangle$

lemma (in *hnp-arith*) *msb-kp1-dist-hnp*:
defines $msb-k \equiv msb\ (k + 1)\ n$
shows $|int(x) - int(msb-k\ x)| < p / (2^{\wedge}k)$
 $\langle proof \rangle$

lemma *msb-kp1-valid-hnp*:
assumes *hnp-arith* $n\ \alpha\ d\ p\ k$
shows *hnp* $n\ \alpha\ d\ p\ k\ (msb\ (k + 1)\ n)$
 $\langle proof \rangle$

12.2 MSB-p

definition *msb-p* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$ **where**
 $msb-p\ p\ k\ x =$
 $(let\ t = (THE\ t.\ (t - 1) * (p / 2^{\wedge}k) \leq x \wedge x < t * p / 2^{\wedge}k)$
 $in\ nat\ (floor\ (t * p / 2^{\wedge}k) - 1))$

lemma *msb-p-defined*:
fixes $p\ k :: nat$
fixes $x :: nat$
assumes $p > 0$
assumes $k > 0$
shows $(\exists!t.\ (t - 1) * (p / 2^{\wedge}k) \leq x \wedge x < t * p / 2^{\wedge}k)$
 $\langle proof \rangle$

lemma (in *hnp-arith*) *msb-p-dist-hnp*:
defines $msb-k \equiv msb-p\ p\ k$
shows $|int\ x - int\ (msb-k\ x)| < p / 2^{\wedge}k$
 $\langle proof \rangle$

lemma *msb-p-valid-hnp*:
assumes *hnp-arith* $n\ \alpha\ d\ p\ k$
defines $msb-k \equiv msb-p\ p\ k$
shows *hnp* $n\ \alpha\ d\ p\ k\ msb-k$
 $\langle proof \rangle$

end

theory *Ad-Codegen-Example*
imports *Hidden-Number-Problem*

begin

13 This theory demonstrates an example of the executable adversary.

full-babai does not need a global interpretation to be executed.

```
value full-babai [vec-of-list [0, 1], vec-of-list [2, 3]] (vec-of-list [2.3, 6.4]) (4/3)
```

Let's define our d , n , p , α , and k .

```
abbreviation d  $\equiv$  72
abbreviation n  $\equiv$  1279
abbreviation p  $\equiv$  (2::nat)1279 - 1
abbreviation  $\alpha$   $\equiv$  p div 3
abbreviation k  $\equiv$  47
```

```
value p
value  $\alpha$ 
```

Since our adversary definition is inside a locale, we need a global interpretation.

```
global-interpretation ad-interp: hnp-adversary d p
defines  $\mathcal{A} = ad\text{-interp}.\mathcal{A}$ 
and int-gen-basis = ad-interp.int-gen-basis
and int-to-nat-residue = ad-interp.int-to-nat-residue
and ts-from-pairs = ad-interp.ts-from-pairs
and scaled-uvec-from-pairs = ad-interp.scaled-uvec-from-pairs
and  $\mathcal{A}\text{-vec} = ad\text{-interp}.\mathcal{A}\text{-vec}$ 
<proof>
```

For this example, we use our executable msb function.

```
abbreviation  $\mathcal{O} \equiv \lambda t. msb\ k\ n\ ((\alpha * t) \bmod p)$ 
```

```
definition inc-amt :: nat where inc-amt = p div 5
```

```
fun gen-ts :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat list where
  gen-ts 0 t = []
| gen-ts (Suc i) t = t # (gen-ts i ((t + inc-amt) mod p))
```

```
definition gen-pairs :: (nat  $\times$  nat) list where
  gen-pairs = map ( $\lambda t. (t, \mathcal{O}\ t)$ ) (gen-ts d 1)
```

The *gen-pairs* function generates the data that the adversary receives. We prove that the adversary is likely to be successful when the *ts* which define this data are uniformly distributed. Here, we use the *gen-ts* function to generate an explicit list of *ts*.

```
value length gen-pairs
```

```
value gen-pairs
```

```
value ad-interp. $\mathcal{A}$  gen-pairs
```

value α

end

References

- [1] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1):1–13, 1986.
- [2] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *LNCS*, pages 129–142. Springer, 1996.
- [3] R. Bottesch, M. W. Haslbeck, and R. Thiemann. A verified efficient implementation of the LLL basis reduction algorithm. In G. Barthe, G. Sutcliffe, and M. Veanes, editors, *LPAR*, volume 57 of *EPiC Series in Computing*, pages 164–180. EasyChair, 2018.
- [4] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A Verified Factorization Algorithm for Integer Polynomials with Polynomial Complexity. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Factorization.html, Formal proof development.
- [5] A. Lenstra, H. Lenstra, and L. László. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261, 12 1982.
- [6] R. Thiemann, R. Bottesch, J. Divasón, M. W. Haslbeck, S. J. C. Joosten, and A. Yamada. Formalizing the LLL basis reduction algorithm and the LLL factorization algorithm in Isabelle/HOL. *J. Autom. Reason.*, 64(5):827–856, 2020.