

The Hidden Number Problem

Sage Binder, Eric Ren, and Katherine Kosaian

February 6, 2026

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	SPMF and PMF helper lemmas	2
2	General helper lemmas	7
2.1	Casting lemmas	9
3	HNP adversary locale	10
4	HNP locales	11
4.0.1	Arithmetic locale	11
4.1	Main HNP locale	12
4.2	Uniqueness lemma	12
4.2.1	Lattice construction and lemmas	13
4.2.2	dist-p definition and lemmas	17

4.2.3	Uniqueness lemma argument	18
4.2.4	Main uniqueness lemma statement	21
4.3	Main theorem	21
4.3.1	Oracle definition	21
4.3.2	Apply Babai lemmas to adversary	21
4.3.3	Main theorem statement	22
5	Some MSB instantiations and lemmas	22
5.1	Bit-shift MSB	22
5.2	MSB-p	23
6	This theory demonstrates an example of the executable adversary.	24

```

theory Misc-PMF
  imports
    HOL-Probability.Probability
    LLL-Basis-Reduction.LLL-Impl

```

```

begin

```

```

definition replicate-spmf :: nat => 'b pmf => '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.

1 SPMF and PMF helper lemmas

```

lemma spmf-eq-element: spmf (p ≫ (λx. return-spmf (x = t))) True = spmf p t
  <proof>

```

```

lemma pmf-true-false:
  fixes p :: 'a pmf
  fixes P Q :: 'a => bool
  defines a ≡ p ≫ (λx. return-pmf (P x))
  defines b ≡ p ≫ (λx. return-pmf (¬ P x))
  shows pmf a True = pmf b False
  <proof>

```

```

lemma spmf-true-false:
  fixes p :: 'a spmf
  fixes P Q :: 'a => bool
  defines a ≡ p ≫ (λx. return-spmf (P x))

```

defines $b \equiv p \gg (\lambda x. \text{return-spmf } (\neg P x))$
shows $\text{spm}f\ a\ \text{True} = \text{spm}f\ b\ \text{False}$
 $\langle \text{proof} \rangle$

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

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

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

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

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 $\text{lhs} = \text{rhs}$
 $\langle \text{proof} \rangle$

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

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

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

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 spmf
  fixes n :: nat
  fixes E :: 'a option  $\Rightarrow$  bool
  assumes spmf (p  $\gg$  (λx. return-spmf (E x))) True  $\leq$  A
  shows spmf (replicate-spmf n p  $\gg$  (λxs. return-spmf (∀ i < n. E (xs!i)))) True
 $\leq$  An
  <proof>

```

```

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

```

```

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

```

```

end
theory Hidden-Number-Problem
  imports
    HOL-Number-Theory.Number-Theory
    Babai-Nearest-Plane.Babai-Correct
    Misc-PMF

```

```

begin

```

```

hide-fact (open) Finite-Cartesian-Product.mat-def
hide-const (open) Finite-Cartesian-Product.mat
hide-const (open) Finite-Cartesian-Product.row
hide-fact (open) Finite-Cartesian-Product.row-def
hide-const (open) Determinants.det
hide-fact (open) Determinants.det-def
hide-type (open) Finite-Cartesian-Product.vec
hide-const (open) Finite-Cartesian-Product.vec
hide-fact (open) 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)

2 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 $A \subseteq \mathbb{N}$
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 $a\ b\ m :: 'a :: \{\text{unique-euclidean-ring}, \text{abs}\}$

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 $S = S'$

<proof>

context *vec-module*

begin

lemma *sumlist-distrib*:

fixes $ys :: ('a::\text{comm-ring-1})\ \text{vec}\ \text{list}$

assumes $\bigwedge w. \text{List.member } ys\ w \implies \text{dim-vec } w = n$

shows $k \cdot_v \text{sumlist } ys = \text{sumlist } (\text{map } (\lambda i. k \cdot_v i)\ ys)$

<proof>

lemma *smult-in-lattice-of*:

fixes $\text{lattice-basis} :: ('a::\text{comm-ring-1})\ \text{vec}\ \text{list}$

assumes $\bigwedge w. \text{List.member } \text{lattice-basis } w \implies \text{dim-vec } w = n$

fixes $\text{init-vec} :: ('a::\text{comm-ring-1})\ \text{vec}$

fixes $k :: 'a::\text{comm-ring-1}$

assumes $\text{init-vec} \in \text{lattice-of } \text{lattice-basis}$

shows $k \cdot_v \text{init-vec} \in \text{lattice-of } ((\text{map } (\cdot_v k)\ \text{lattice-basis}))$

<proof>

end

lemma *filter-distinct-sorted*:

fixes $l :: ('a::\text{linorder})\ \text{list}$

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 $\text{filter } P\ l = \text{sorted-list-of-set } A$

<proof>

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>

2.1 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 $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies \text{dim-vec } \text{mem} = n$
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 $\bigwedge \text{mem. } \text{List.member } w \ \text{mem} \implies \text{dim-vec } \text{mem} = n$
shows $\text{map-vec int-of-rat } v =$

$abelian-monoid.sumlist$ ($module-vec$ $TYPE(int)$ n)
 $(map$ ($map-vec$ $int-of-rat$) w)
 $\langle proof \rangle$

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

$\langle proof \rangle$

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 ($\lambda i. rat-of-int$ ($c\ i$) $\cdot_v\ qs\ !\ i$)

$[0..<length\ qs]$)) =

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

$\langle proof \rangle$

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)

$\langle proof \rangle$

3 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 × nat) list ⇒ int vec **where**
*scaled-uvec-from-pairs pairs = vec-of-list ((map (λ(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

4 HNP locales

4.0.1 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**
*μ ≡ nat (ceiling (1/2 * (sqrt n)) + 3)*

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

lemma *int-p-prime*: prime (int *p*) ⟨proof⟩

lemma *p-geq-2*: $p \geq 2$ ⟨proof⟩

lemma *n-geq-1*: $n \geq 1$
 ⟨proof⟩

lemma *μ-le-k*:
shows $\mu + 1 \leq k$
 ⟨proof⟩

lemma *k-plus-1-lt*:
shows $k + 1 < \log 2 p$

<proof>

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

4.1 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

4.2 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* $(\text{map of-nat } (\text{ts-to-as } ts) \text{ } @ [0])$

lemma *ts-to-u-alt*:

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

<proof>

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

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

4.2.1 Lattice construction and lemmas

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

lemma *length-p-vecs*: $\text{length } p\text{-vecs} = d$ ⟨proof⟩

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

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

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*: $\text{length } (\text{gen-basis } ts) = d + 1$ ⟨proof⟩

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 = -)
 ⟨proof⟩

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**
 $\text{close-vec } ts \text{ } v \iff (\text{sq-norm } ((ts\text{-to-}u \text{ } ts) - v) < ((\text{of-nat } p) / 2^{\wedge} \mu)^{\wedge} 2)$

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**
 $\text{good-lattice } ts \iff (\forall v \in \text{gen-lattice } ts. \text{close-vec } ts \text{ } v \implies \text{good-vec } v)$

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

$bad\text{-}lattice\ ts \longleftrightarrow \neg\ good\text{-}lattice\ ts$

definition *sampled-lattice-good* :: *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 $c :: \text{nat} \Rightarrow \text{int}$
fixes $i :: \text{nat}$
assumes $i \leq \text{length } ts$
assumes $\text{length } ts = d$
shows $(\text{sumlist } (\text{map } (\lambda i. \text{of-int } (c \ i) \cdot_v ((\text{gen-basis } ts) ! \ i)) [0 ..< \text{length } (\text{gen-basis } ts)])) \$ i$
 $= (\text{if } (i = d) \text{ then } (\text{rat-of-int } (c \ d) / \text{rat-of-nat } p) \text{ else } \text{rat-of-int } ((c \ d) * ts ! i + (c \ i) * p))$
 $\langle \text{proof} \rangle$

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

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

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

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

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

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

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

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

lemma *vec-class-carrier*:

assumes $\text{length } ts = d$

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

$\langle \text{proof} \rangle$

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

assumes $\text{length } ts = d$

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

$\langle \text{proof} \rangle$

lemma *vec-class-last*:

assumes $\text{length } ts = d$

assumes $v \in \text{vec-class } ts \ \beta$
shows $v \cdot d = \text{rat-of-int } \beta / \text{rat-of-int } p$
 $\langle \text{proof} \rangle$

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 $\text{real}(p^2) * \text{Inf}\{\text{real-of-rat } (\text{sq-norm } (x - u)) \mid x. x \in \text{gen-lattice } ts\}$
 $= \text{babai.closest-distance-sq } (\text{int-gen-basis } ts) ((\text{rat-of-nat } p) \cdot_v u)$
 $\langle \text{proof} \rangle$

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 $L = \bigcup \{\text{vec-class-mod-}p \ ts \ \beta \mid \beta. \beta \in \{0..<p::\text{int}\}\} \text{ (is - = ?rhs)}$

<proof>

4.2.2 dist-p definition and lemmas

definition *dist-p* :: *int* ⇒ *int* ⇒ *int* **where**
dist-p *i j* = *Inf* {*abs* (*i - j + z * (of-nat p)*) | *z. True*}

lemma *dist-p-well-defined*:
fixes *i* :: *int*
fixes *j* :: *int*
shows *dist-p* *i j* ∈ {*abs* (*i - j + z * (of-nat p)*) | *z. True*} (**is** - ∈ ?*S*)
<proof>

lemma *dist-p-set-bdd-below*:
fixes *i j* :: *int*
shows ∀ *x* ∈ {*abs* (*i - j + z * (of-nat p)*) | *z. True*}. *0* ≤ *x*
<proof>

lemma *dist-p-le*:
fixes *i j* :: *int*
shows ∀ *x* ∈ {*abs* (*i - j + z * (of-nat p)*) | *z. True*}. *dist-p* *i j* ≤ *x* (**is** ∀ *x* ∈ ?*S*.
-)
<proof>

lemma *dist-p-equiv*:
fixes *i* :: *int*
fixes *j* :: *int*
shows [*dist-p* *i j* = *i - j*] (*mod p*) ∨ [*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*) ∨ *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* ∈ {*1..<p*}. (*dist-p* (*t*i*) (*t*j*)) = *dist*} ≤ 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 $|dist-p\ a\ b - dist-p\ a\ b'| \leq |b - b'|$
 $\langle proof \rangle$

4.2.3 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

$bad\text{-}beta\ ts\ \beta \longleftrightarrow \neg\ good\text{-}beta\ ts\ \beta$

definition *some-bad-beta where*

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

definition *bad-beta-union where*

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

lemma *reduction-1:*

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

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

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

$\langle proof \rangle$

lemma *reduction-1-pmf:*

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *reduction-2-pmf:*

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

$\langle proof \rangle$

lemma *bad-beta-union-bound:*

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

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

(is ?lhs \leq ?rhs)

$\langle proof \rangle$

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

fixes $\beta\ ts$

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

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

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

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

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

$\langle proof \rangle$

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

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

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

$\langle proof \rangle$

lemma *dist-p-helper*:
fixes $ts\ i$
assumes $ts: ts \in \text{set-pmf } ts\text{-pmf}$
assumes $i: i < d$
assumes $dist: dist\text{-}p\ (int\ (ts!i) * \beta)\ ((ts\text{-}to\text{-}as\ ts)!i) < p / (2^{\wedge}\mu)$
shows $dist\text{-}p\ ((ts!i) * \beta)\ ((ts!i) * \alpha) \leq (2 * p) / (2^{\wedge}\mu)$
 $\langle proof \rangle$

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

lemma *prob-A-helper'*:
fixes $\beta :: int$
defines $[simp]: t\text{-pmf} \equiv \text{pmf-of-set } \{1..<p\}$
defines $[simp]: A\text{-cond} \equiv \lambda t. (2 * p) / (2^{\wedge}\mu) < dist\text{-}p\ (\beta * t)\ (\alpha * t)$
defines $[simp]: A\text{-pmf} \equiv t\text{-pmf} \gg (\lambda t. \text{return-pmf } (A\text{-cond } t))$
defines $[simp]: A \equiv \text{pmf } A\text{-pmf } True$
assumes $\beta: \beta \in \{0..<p::int\}$
assumes *False*: $\neg (\beta = \alpha \text{ mod } p)$
shows $(1 - A)^{\wedge}d \leq (5 / (2^{\wedge}\mu))^{\wedge}d$
 $\langle proof \rangle$

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

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

lemma *sampled-lattice-unlikely-bad*:
shows $\text{pmf } \text{sampled-lattice-good } False \leq (p - 1) * ((5 / (2^{\wedge}\mu))^{\wedge}d)$
 $\langle proof \rangle$

4.2.4 Main uniqueness lemma statement

lemma *sampled-lattice-likely-good*: $\text{pmf sampled-lattice-good True} \geq 1/2$
 ⟨proof⟩

4.3 Main theorem

4.3.1 Oracle definition

definition $\mathcal{O} :: \text{nat} \Rightarrow \text{nat}$ **where**
 $\mathcal{O} t = \text{msb-p } ((\alpha * t) \bmod p)$

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

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

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

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

assumes $length\ pairs = d$
assumes $u\text{-vec-is-msbs}\ pairs$
shows $sq\text{-norm} ((1/(rat\text{-of-nat}\ p)) \cdot_v (map\text{-vec}\ rat\text{-of-int}\ (\mathcal{A}\text{-vec}\ pairs)) - (ts\text{-to-u}\ (ts\text{-from-pairs}\ pairs))) < ((of\text{-nat}\ p)/2^{\wedge}(\mu))^{\wedge}2$
 $\wedge ((1/(rat\text{-of-nat}\ p)) \cdot_v (map\text{-vec}\ rat\text{-of-int}\ (\mathcal{A}\text{-vec}\ pairs)) \in vec\text{-class-mod-p}\ (ts\text{-from-pairs}\ pairs)\ (\mathcal{A}\ pairs))$
 $\langle proof \rangle$

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

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

4.3.3 Main theorem statement

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

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

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

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

Alternative definition of $game$, written as a $map\text{-pmf}$

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

lemma $game' = game$
 $\langle proof \rangle$

end

5 Some MSB instantiations and lemmas

5.1 Bit-shift MSB

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

lemma *msb-dist*:
fixes $k\ n :: \text{nat}$
assumes $k < n$
assumes $1 \leq k$
shows $|\text{int } (x) - \text{int } (\text{msb } k\ n\ x)| < 2^{\wedge}n / (2^{\wedge}k)$
 $\langle \text{proof} \rangle$

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

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

5.2 MSB-p

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

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

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

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

end

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

begin

6 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-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  $\mathcal{A}$ -vec = ad-interp. $\mathcal{A}$ -vec
```

```
  <proof>
```

For this example, we use our executable msb function.

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