# Formalizing Coppersmith's Method in Isabelle/HOL

Katherine Kosaian[1]✉, Yong Kiam Tan[2], and Kristin Yvonne Rozier[1]

[1] Iowa State University, Ames IA 50011, USA
kkosaian,kyrozier@iastate.edu
[2] Institute for Infocomm Research (I$^2$R), A*STAR, Singapore
tan_yong_kiam@i2r.a-star.edu.sg

**Abstract.** We formalize *Coppersmith's method*, an algorithm for finding small (in magnitude) roots of univariate integer polynomials mod $M$, in the theorem prover Isabelle/HOL. Our work is motivated by the goal of moving cryptography into the realm of formal methods by formalizing not only the correctness and security arguments behind cryptographic algorithms but also the mathematics behind *attacks* on those algorithms. Coppersmith's method fits into this goal as it has important applications in cryptography and is used in various attacks on the RSA algorithm for public-key cryptography. We overview and give insights into our formalization, which includes new contributions to Isabelle/HOL's libraries, and builds on the existing formalization of the Lenstra–Lenstra–Lovász (LLL) algorithm for lattice basis reduction.

**Keywords:** Coppersmith's method · theorem proving · LLL basis reduction · cryptography

## 1 Introduction

Coppersmith's method [18] is an algorithm to find small roots of univariate integer polynomials mod $M$. There are good techniques for finding roots of polynomials over the integers [54], but working mod $M$ is more challenging—accordingly, Coppersmith's method takes a polynomial $f$ and finds a polynomial $g$ so that, for all integers $x$ with magnitude smaller than a computable bound $B$, $x$ is a root of $f$ mod $M$ *only if* $x$ is a root of $g$ over the integers. Coppersmith's method has applications in cryptography [22], particularly for breaking (improper) implementations of RSA [48], a widely used public-key encryption algorithm. It was used as recently as 2017 in an attack on RSA that became known as ROCA (short for "Return of Coppersmith's attack") [39]. As RSA is so widely used, successful attacks can have widespread and devastating impact; ROCA rendered hundreds of thousands of Estonian ID cards vulnerable [4].

In this work, we formalize Coppersmith's method in the theorem prover Isabelle/HOL. We chose to work with Isabelle/HOL [42, 43] because of its user-friendliness—including strong automation tactics, Sledgehammer [45], and useful

search tools [27,50]—and its well-developed mathematical libraries. The library support is particularly crucial: Isabelle/HOL's Archive of Formal Proofs (AFP) already contains a formalization [13,19,51] of the Lenstra–Lenstra–Lovász (LLL) algorithm [34], which is a key ingredient in Coppersmith's method. To our knowledge, the algorithm has not been formalized in any other theorem prover.

Our work is largely motivated by the growing recognition of the benefits of *computer-aided cryptography* [3], i.e., the use of formal methods in the design and implementation of cryptography. At the design level, cryptographic security proofs can be flawed [29], and while formal verification is by no means a panacea, it is excellent at catching flawed or imprecise arguments, thus providing an added level of confidence in formalized results. There is already considerable work on formalizing protocols and security results thereof in tools like Tamarin [7] and ProVerif [9]; frameworks for formalizing game-based cryptographic arguments have been built in a number of provers, including Isabelle/HOL [5,8,24,46] following a call by Halevi [23]. Formally verified implementations of cryptographic primitives are now included in real-world, widely-used libraries [11,21,47,58].

However, there is arguably a gap in *formalizing cryptographic attacks* and, particularly, the mathematical tools used to formulate these attacks, like Coppersmith's method. This is significant, because formalization of cryptographic attacks could help researchers to understand these attacks (and their limits) more deeply, e.g., the aerospace industry has specifically identified researching attack strategies as important [17,55]. Formalization of attacks could also enable a deeper understanding of the corresponding cryptographic algorithms; in particular, given formalizations of both a cryptographic algorithm and a corresponding attack, one could *formally* identify which assumptions are needed to secure the former against the latter.

The contributions of this paper are as follows.

- We present a formal proof of Coppersmith's method, first formalizing a "lightweight" version of Coppersmith's method, then the fully general version. This stepping-stone approach was presented by Galbraith [22], and we show that it also works well for structuring the formal proof.[3]
- Along the way, we formalize a theorem due to Howgrave-Graham which is instrumental in the (simplified) presentation of Coppersmith's method; we also contribute some formalized results about LLL and lattices.
- We give insight into handling some of the (low-level) challenges we faced, such as verifying arithmetic and navigating frequent type conversions. We also revisit the proofs to abstract the core correctness argument for Coppersmith's method using Isabelle/HOL's *locale* system [2]. This yields a reusable and slightly generalized result.

Our work is a novel application of the existing LLL formalization [13,19,51]. Although our primary motivation is to formalize cryptographic attacks,

[3] While developing the formalization, we also found a small error in the textbook's statement of the bound $B$ on the magnitude of roots that can be found by Coppersmith's method. This has been acknowledged and fixed by the author.

executability of the LLL formalization means our formalized algorithm can be readily executed. Accordingly, we lay the groundwork for a verified decision procedure[4] for the existence of (sufficiently) small polynomial roots mod $M$. Our final formalization is ≈3675 lines of code (the algorithm itself is less than 100 lines, excluding code for the existing verified LLL implementation). Our formalization is available on the Archive of Formal Proofs (AFP) [30].

In the remainder of this paper, we introduce related work (Sect. 2) and Coppersmith's method (Sect. 3), discuss details of our formalization (Sect. 4), and conclude with a discussion of future work (Sect. 5).

## 2 Related Work

Some common cryptographic attacks on well-known and widely-used algorithms (including RSA, DSA, and ECDSA) have shared mathematical underpinnings based on the LLL algorithm [34], which is an algorithm for performing *lattice reduction* [57]. A *lattice* is a grid of points (see Fig. 1) which can be represented in various ways, including by a set of linearly independent *basis vectors*, where any point in the lattice is an integer linear combination of the basis vectors. Lattice reduction aims to find representations of lattices that are mathematically equivalent (i.e., they describe the same lattice) but are also "better" or "simpler" according to some metric; see Fig. 1. The LLL algorithm in particular can find *short vectors in lattices*, which is useful in a wide variety of applications [40].

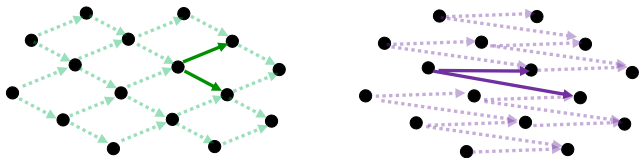

**Fig. 1.** Two different representations of a lattice (the grid of black points). Solid green arrows (LHS) visualize one basis; dotted green arrows visualize how these vectors span the lattice. Solid purple arrows (RHS) visualize another basis (the dotted purple arrows are included for visualization purposes). The green vectors are shorter than the purple.

The formalization of the LLL algorithm [13, 19, 51] in Isabelle/HOL makes the prover a natural choice for our work. The LLL algorithm has a wide variety of applications (not only in cryptography!) [40], a few of which have already been formalized in Isabelle [12, 20]. Other results related to lattices have also been formalized in Isabelle; notably, the *closest vector* problem and the *shortest vector* problem have been formalized and proved to be NP-hard [33].

Beyond the LLL algorithm, Isabelle/HOL has been used in many formalizations in the area of security, including Paulson's pioneering work on formalizing

[4] To achieve a verified decision procedure, we would also need to link our work to a (verified) implementation for integer polynomial root finding.

cryptographic protocols [44]. Other security formalizations range from specifying cryptographic algorithms such as RSA [35, 56] and ECDSA [56], to model checking attack trees in Isabelle/HOL [28]. The CryptHOL framework [8, 36] was developed to assist cryptographers in developing foundational proofs of *security properties* for cryptographic constructions in Isabelle/HOL. In particular, CryptHOL allows one to model and reason about randomized cryptographic games that interact with (adaptive) adversaries; it has been used in a number of works, e.g., [14–16,32]. Isabelle/HOL is also used as a backend for the `qrhl-tool` proof assistant for (post-)quantum cryptographic security proofs [53].

Coppersmith's algorithm itself has applications beyond attacking RSA. In cryptography, it is used, for example, in the proof of security for Rabin-SAEP via reduction to factoring [10]; Barthe et al. [6] used EasyCrypt to mechanize the security proof for a closely related encryption scheme but the part of their work utilizing Coppersmith's method is carried out on pen-and-paper. Work in Coq used—but did not verify—Coppersmith's method and also its (heuristic) bivariate extension to construct a proof-producing procedure to find all small roots of univariate and bivariate polynomials over the integers (without modulus) [37, 38].

## 3 Coppersmith's Method and Formalization Overview

We now turn to a discussion of Coppersmith's method, interspersing an explanation of the mathematical workings of Coppersmith's method with a high-level view of our formalization. We largely defer in-depth discussion of low-level formalization details to Sect. 4. Our formalization primarily follows the presentation in Galbraith [22, Chapter 19], although we did not hesitate to consult other references when necessary, e.g., [52, Section 17.3]; we adopt Galbraith's notation in this section. There are also some high-level similarities in our presentation: like Galbraith, we begin with a discussion of Howgrave-Graham's theorem (a key helper lemma in the proof of Coppersmith's method), then discuss a "lightweight" version of Coppersmith's method that we use in our formalization as a stepping-stone towards the full version, and finally discuss the full Coppersmith's method.

*Reading Isabelle/HOL code.* Throughout this section, we include snippets of some of the key portions of our formalization. Isabelle/HOL supports both function definitions (analogous to code) and proofs of properties of those definitions (formalized versions of pen-and-paper proofs). Isabelle definitions begin with the keywords **definition**, **fun**, or **function**; proofs begin with the keywords **lemma** and **theorem**. In definitions, the types of arguments and the return value are explicitly given. For example, a definition named *square_integer* that takes an integer (type *int* in Isabelle/HOL) and returns its square could be written as:

```
definition square_integer :: "int ⇒ int" where
  "square_integer x = x * x"
```

In lemma and theorem statements (example below), the keyword **fixes** introduces a variable (e.g., **fixes** *var :: "type"* introduces variable *var* of type *type*), the keyword **assumes** introduces an assumption (e.g., **assumes** *assm_1: "x > 0"* introduces an assumption labeled *assm_1* which states that variable *x* is greater than 0), the keyword *defines* introduces an abbreviation (e.g., **defines** *"y ≡ square_integer x"* allows us to use *y* and *square_integer x* interchangeably), and the keyword **shows** specifies the conclusion of the lemma (i.e., the statement that is being proved; e.g., **shows** *"y > 0"* claims that squaring integer *x* returns a positive result).

```
lemma pos_square:
  fixes x :: "int"
  assumes assm_1: "x > 0"
  defines "y ≡ square_integer x"
  shows "y > 0"
```

For brevity in this paper, we only include *statements* of lemmas and theorems. The proofs are in the associated formalization; Isabelle/HOL's structured proof language [41] makes these formal proofs stylistically similar to pen-and-paper proofs. We explain each Isabelle/HOL code snippet that we include in this paper; for additional reference, see Koutsoukou-Argyraki's excellent introduction of Isabelle/HOL for mathematicians [31].

### 3.1 Howgrave-Graham's Theorem

The original proof of Coppersmith's method [18] was simplified by Howgrave-Graham [26], and we follow the latter simplified presentation. Howgrave-Graham proved the following key result:

**Theorem 1 (Howgrave-Graham).** *(Cross-reference [22, Theorem 19.1.2]). Let $X$ be an integer greater than 0. Let $F(x) = a_0 + a_1x + \cdots + a_dx^d$ be a polynomial, and let $x_0$ be an integer with $|x_0| \leq X$ so that $F(x_0) \equiv 0 \mod M$. Taking the row vector $(a_0, a_1X, ..., a_dX^d)$, and letting $||v||$ denote the Euclidean norm of vector $v$, if $||(a_0, a_1X, ..., a_dX^d)|| < M/\sqrt{d+1}$, then $F(x_0) = 0$.*

Intuitively, this says that "small enough" roots of a polynomial $F \mod M$ are also roots of $F$ over the integers, where the precise notion of "small enough" is given in terms of an associated vector. We can think of associating row vectors to polynomials where, given a fixed integer $X$ with $X > 0$, the polynomial $a_0 + a_1x + \cdots + a_dx^d$ is associated to the vector $(a_0, a_1X, a_2X^2, \ldots, a_dX^d)$, as in the theorem statement. We can also construct polynomials associated to row vectors in an inverse way, so that a row vector of shape $(a_0, a_1X, a_2X^2, \ldots, a_dX^d)$ is associated to the polynomial $a_0+a_1x+\cdots+a_dx^d$. This bidirectional association will be used throughout the development of Coppersmith's method. The proof of Howgrave-Graham's theorem mainly involves arithmetic. A full presentation is available in Galbraith [22]. We formalize Theorem 1 as follows.

```
lemma Howgrave_Graham_int_poly:
  fixes F :: "int poly"
  fixes M X :: "nat"
  fixes x0 :: "int"
  assumes M_gt: "M > 0"
  assumes root_mod_M: "poly F x0 mod M = 0"
  assumes root_bound: "abs x0 ≤ X"
  assumes norm_bound:
    "sqrt(sq_norm_vec(vec_associated_to_int_poly F X))
      < M / sqrt (degree F + 1)"
  shows "poly F x0 = 0"
```

Here, we have `F`, a univariate polynomial with integer coefficients (type `int poly`),[5] natural numbers `M` and `X` (type `nat`), and an integer `x0` (type `int`). We assume that `M > 0` (assumption `M_gt`), that the absolute value of `x0` is less than `X` (assumption `root_bound`), that `x0` is a root of `F` mod `M` (in assumption `root_mod_M`, where `poly F x0` is Isabelle/HOL's function to evaluate the univariate polynomial `F` at `x0`), and that the Euclidean norm of the vector associated to `F` (given `X`) satisfies the desired inequality (in assumption `norm_bound`, where `vec_associated_to_int_poly F X` is our function for obtaining the vector associated to `F`, and `sq_norm_vec` is a function that returns the square of the Euclidean norm of a vector). The conclusion says that `poly F x0 = 0`, meaning that the polynomial `F` evaluated at integer `x0` is zero.

### 3.2 A Stepping Stone

Intuitively, Coppersmith's method, and variants thereof, start by constructing a lattice using basis vectors associated to some well-chosen polynomials (using the vector-polynomial association introduced in Sect. 3.1). The polynomials are chosen to satisfy a number of important properties; most notably, they all have $x_0$ as a root modulo $M^k$, for some $k \geq 1$, and so the polynomial associated to *any* vector in this lattice also satisfies this property. Then, LLL basis reduction is used to find a *short* vector in this lattice; because LLL preserves various properties of the input lattice, the polynomial associated to the short vector returned by LLL will also have $x_0$ as a root modulo $M^k$. The respective correctness theorems give specific conditions under which this vector returned by the LLL algorithm is short enough for the associated polynomial to satisfy the conditions of Howgrave-Graham (which means that we can factor this polynomial over the integers to find $x_0$, as long as $x_0$ is itself small enough). We visualize this intuition and the high-level workings of Coppersmith's method (and variants) in Fig. 2.

The way in which the initial lattice is constructed (i.e., the set of chosen polynomials) determines an upper bound on the length of the short vector obtained from LLL; the shorter the vector, the larger the $X$ can be, i.e., the larger magnitude of roots $x_0$ that are guaranteed to be found by the method.

[5] In fact, Howgrave-Graham's result holds for polynomials with `real` coefficients and we also formalize this more general fact in Isabelle/HOL.

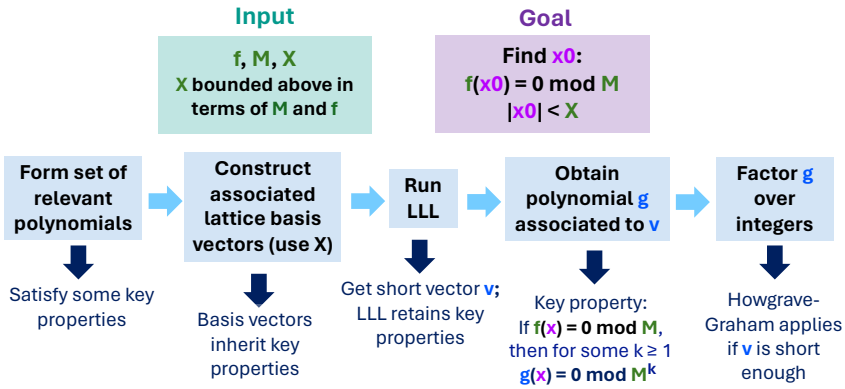


**Fig. 2.** A visualization of the high-level workings of Coppersmith's method.

The full version of Coppersmith's method uses a cleverly chosen set of polynomials to obtain a particularly short vector. There are also "lightweight" versions of Coppersmith's method that construct slightly simpler lattices and obtain slightly worse bounds on $X$. Galbraith's presentation of Coppersmith's method [22] details one of these lightweight methods before presenting the full generality; Trappe, Wade, and Washington also present this lightweight method in their textbook [52, Section 17.3]. We found it effective in our formalization to establish the lightweight method first, as it shares significant mathematical content with the full method, but is somewhat conceptually and mathematically simpler. Our top-level result for the lightweight method is the following.

```
lemma towards_coppersmith:
  fixes p f:: "int poly"
  fixes M X:: "nat"
  fixes x0:: "int"
  defines "d ≡ degree p"
  defines "f ≡ towards_coppersmith p M X"
  assumes monic_poly: "monic p"
  assumes "d > 0" and "M > 0" and "X > 0"
  assumes zero_mod_M: "poly p x0 mod M = 0"
  assumes X_lt:
    "X < 1/sqrt 2 * 1/root d (d+1) * (root (d*(d+1)) M)^2"
  assumes x0_le: "abs x0 ≤ X"
  shows "poly f x0 = 0"
```

In this lemma, `towards_coppersmith` is our function for the lightweight version of Coppersmith. We assume that we are given a monic polynomial `p` with integer coefficients and degree `d > 0`. We also assume that integer `x0` is a root of `p` mod `M` (assumption `zero_mod_M`), where `M > 0`, and absolute value `abs x0` is bounded by a user-chosen bound `X` (assumption `x0_le`), where `X > 0`. Importantly, `X` must be at most $X < 1/(\sqrt{2}(d+1)^{1/d}) \cdot M^{2/(d(d+1))}$ (assumption `X_lt`, where `root a b`

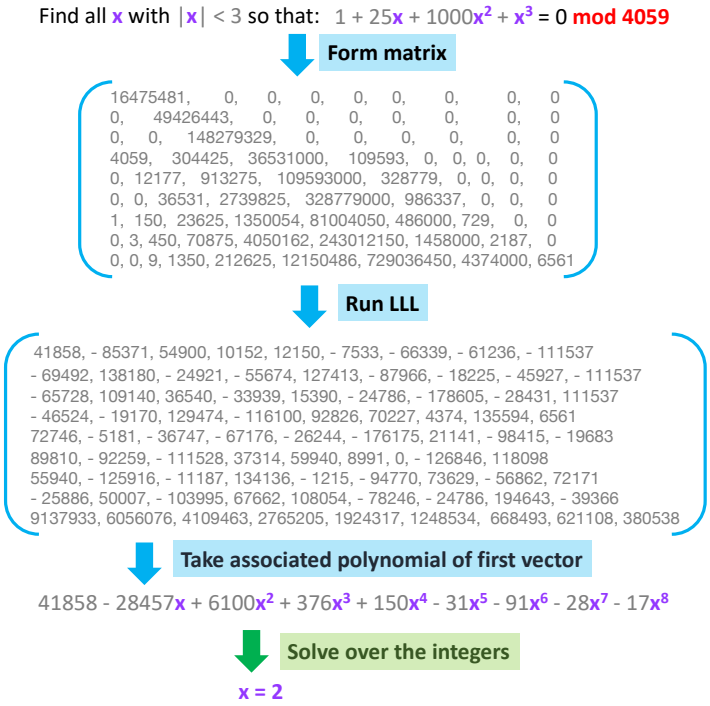


**Fig. 3.** An example lattice in Coppersmith's method, before and after running LLL.

is Isabelle's syntax for $a^{1/b}$). The conclusion says that, given these assumptions, `x0` is also a root of the polynomial `f` over the integers, where `f` is the polynomial obtained from `towards_coppersmith p M X`.

### 3.3 Coppersmith's Method

To get to the full Coppersmith's method, we essentially use the same idea as in Sect. 3.2, but with a different choice of lattice. We input a square lattice of basis vectors corresponding to the polynomials $G_{i,j}(x) = M^{h-1-j}p(x)^j x^i$, where $d$ is the degree of our input polynomial $p$, $\epsilon > 0$ is a parameter to the algorithm that satisfies certain bounds, and $0 \leq i < d$ and $0 \leq j < h$ where $h = \lceil(((d-1)/(d \cdot \epsilon) + 1)/d)\rceil$ [22].

*Example 1.* We visualize an example of the above lattice construction in Fig. 3, where we start with the polynomial $x^3 + 1000x^2 + 25x + 1$, and our goal is to find roots with magnitude less than $X$ mod $M$ where $X = 3$ and $M = 4059$. Here, $d = 3$, and we choose $\epsilon = 0.1$,[6] and so $h = \lceil(((3-1)/(3 \cdot 0.1) + 1)/3)\rceil = 3$.

[6] We have considerable leeway when we choose this value. To satisfy the conditions in our upcoming theorem, `coppersmith_method_finds_small_roots`, we only need $\epsilon > 0$, $\epsilon < 1/d$, $\epsilon \leq 0.18 \cdot (d-1)/d$, and $X < 1/2 \cdot M^{1/d-\epsilon}$, or $\epsilon < 1/d - \log_M(2X)$.

So, we have $G_{i,j}(x)$ for $i, j \in \{0, 1, 2\}$, which gives us nine basis vectors. We want to have a square input lattice of dimension $9 \times 9$, so we will zero-pad our vectors when necessary. As an example of this, the second vector in this lattice corresponds to $G_{1,0}(x) = 4059^2 x$ and so is $(0, 4059^2 \cdot 3, 0, 0, 0, 0, 0, 0, 0)$. As a second example, the seventh vector in this lattice corresponds to $G_{0,2}(x) = M^0 \cdot p(x)^2 = x^6 + 2000x^5 + 1000050x^4 + 50002x^3 + 2625x^2 + 50x + 1$, and so our vector is $(1, 50 \cdot 3, 2625 \cdot 3^2, 50002 \cdot 3^3, 1000050 \cdot 3^4, 2000 \cdot 3^5, 3^6, 0, 0)$.

In Fig. 3, we visualize both the matrix with the initial lattice basis vectors and the matrix after running the LLL algorithm. The short vector that we obtain is the first row of this second matrix, which is $(41858, -85371, 54900, 10152, 12150, -7533, -66339, -61236, -111537)$. The polynomial associated to this vector is then $41858 - \frac{85371}{3}x + \frac{54900}{3^2}x^2 + \frac{10152}{3^3}x^3 + \frac{12150}{3^4}x^4 - \frac{7533}{3^5}x^5 - \frac{66339}{3^6}x^6 - \frac{61236}{3^7}x^7 - \frac{111537}{3^8}x^8 = 41858 - 28457x + 6100x^2 + 376x^3 + 150x^4 - 31x^5 - 91x^6 - 28x^7 - 17x^8$, and factoring over the integers yields that $x_0 = 2$ is a root of this polynomial.

Our top-level result for Coppersmith's method is stated using the following definition of the root bound term which forces $X < \frac{1}{2} \cdot M^{1/d-\epsilon}$. The formalized theorem is shown further below.

```
definition root_bound :: "nat ⇒ nat ⇒ real ⇒ real" where
  "root_bound M d eps = 1/2 * (M powr (1/d - eps))"
```

```
theorem coppersmith_method_finds_small_roots:
  fixes p f:: "int poly"
  fixes M X:: "nat"
  fixes x0:: "int"
  fixes eps:: "real"
  defines "d ≡ degree p"
  defines "f ≡ coppersmith p M X eps"
  assumes monic_poly: "monic p"
  assumes "d > 1" and "M > 0" and "X > 0"
  assumes zero_mod_M: "poly p x0 mod M = 0"
  assumes X_lt: "X < root_bound M d eps"
  assumes x0_le: "abs x0 ≤ X"
  assumes eps_le: "eps ≤ 0.18*(d-1)/d"
  assumes eps_lt: "eps < 1 / d"
  assumes eps_gt: "eps > 0"
  shows "poly f x0 = 0"
```

Similar to the lightweight method, we assume that `p` is a monic polynomial with integer coefficients and that `x0` is a root of `p` mod `M` where the absolute value `abs x0` is bounded by a user-chosen bound `X` (we also require that `p` has degree `d` $> 1$). The upper bound on $X$ is stated in terms of `root_bound M d eps` as described above; the additional parameter `eps` $> 0$ must be within a specific range (more specifically, `eps` $< 1/$`d` and `eps` $\leq 0.18($`d` $- 1)/$`d`). Given the choice of `X` and `eps`, Coppersmith's method returns a polynomial `f = coppersmith p M X eps` with integer coefficients such that `x0` is a root of `f` over the integers.

We can see that the bound $X < \frac{1}{2} \cdot M^{1/d-\epsilon}$ is better than that obtained in Sect. 3.2, i.e., the parameter `X` can be chosen to cover a wider range of roots `x0` compared to the lightweight version—the more intricate lattice construction has paid off. This bound can actually be further improved to $X = M^{1/d}$ [18], (by, for example, applying the method we have formalized multiple times to cover the missing portion of the interval [22, Remark 19.1.13]), and so Coppersmith's method is often stated as a method to find roots smaller than $M^{1/d}$, but what we have formalized is the core of the result.

The proof of our formalized result for Coppersmith's method is relatively similar to the proof of the lightweight method; one of the primary differences is the arithmetic required. The main advantage of proving the lightweight method first is that it helped identify the different components needed to formalize full Coppersmith; this was also useful when generalizing to the generic Coppersmith-type result, which will be discussed further in Sect. 4.4.

*Example 2.* While Fig. 3 shows one example of Coppersmith's method, the polynomial in that example has relatively small coefficients and the bound $X$ on the root we are searching for is very small. The beauty of Coppersmith's method is that it is quite quick, thanks to the speed of LLL. In Fig. 4, we visualize a more substantial example, where we use our verified implementation of Coppersmith's method to factor $f(x) = x^3 + (2^{25} - 2883584)x^2 + 46976195x + 227 \bmod M$, where $M = (2^{20}+7)(2^{21}+17)$; this is an exercise taken from Galbraith [22]. Running our implementation of Coppersmith's method in Isabelle/HOL on this example takes about 20 seconds (this does not count the factoring step, which we perform separately, in a computer algebra tool).

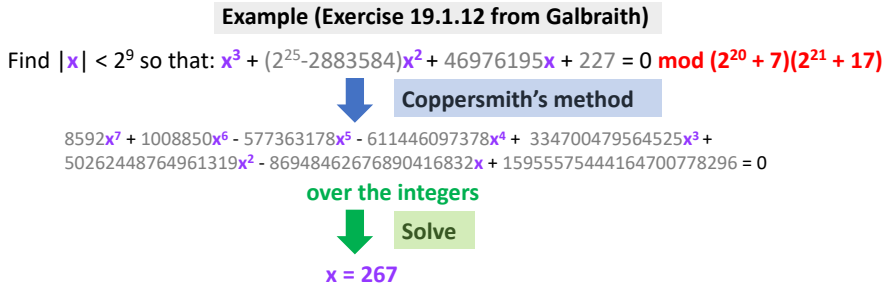


**Fig. 4.** A second example of Coppersmith's method in action, with larger values of $M$ and $X$ and for a polynomial with larger coefficients.

## 4 Formalization Details

We now discuss further details of our formalization: challenges we encountered, some useful results we contribute to the Isabelle/HOL libraries, and a slight generalization of Coppersmith's method that we formalized.

### 4.1 Type Conversion Woes

One surprisingly thorny challenge that we encounter throughout our formalization is converting between the Isabelle/HOL types for natural numbers (*nat*), integers (*int*), rationals (*rat*) and reals (*real*). This is a low-level difficulty that arises in formalization when mathematical source material effortlessly changes between terms with different (formal) types, e.g., the various kinds of numbers listed above are ordered by inclusion.

Implicit type coercions can be seen in the assumed bounds in our theorems such as *coppersmith_method_finds_small_roots* from Sect. 3.3. For example, in the bound *eps* $\leq$ *0.18*(d-1)/d*, the degree *d* is a natural number which is implicitly converted to type *real* by Isabelle/HOL for the theorem statement. Although these implicit coercions are helpful for reducing syntactic clutter, the challenge is that Isabelle/HOL sometimes places the coercions in non-ideal positions, which can be awkward in proofs.

A more involved problem is that Coppersmith's method works with integer polynomials of type *int poly*. This means that our lattice has basis vectors with integer entries, of type *int vec*. However, *int* is not a field, so many results for vector spaces do not hold for *int vec* (which is a module). Thus, we frequently needed to go back and forth between objects of type *int vec* and objects of type *rat vec*, which is the type of vectors with rational entries. The theorems about LLL from the libraries, which we use to establish useful properties of our vectors, are proven for vectors with rational entries.

Unfortunately, some properties that transfer obviously when converting from integer vectors to rational vectors do not transfer automatically in Isabelle/HOL. For example, it was useful to establish that if a set of integer vectors is linearly independent (with integer scalars), then the corresponding lifted set of rational vectors is also linearly independent (with rational scalars), and vice-versa. The formalization of these results (in contrapositive form) is shown below; here, linear dependence for a set of vectors *vs* means that there is a linear combination of *vs* (with appropriate scalars) summing to the 0 vector.

```
lemma lin_dep_int_to_rat:
  assumes vs: "vs ⊆ carrier_vec n"
  assumes ldi: "module.lin_dep class_ring (module_vec TYPE(int) n) vs"
  shows "module.lin_dep class_ring (module_vec TYPE(rat) n)
     (of_int_vec ‘ vs)"
```

```
lemma lin_dep_rat_to_int:
  assumes vs: "vs ⊆ carrier_vec n"
  assumes ldi: "module.lin_dep class_ring (module_vec TYPE(rat) n)
     (of_int_vec ‘ vs)"
  shows "module.lin_dep class_ring (module_vec TYPE(int) n) vs"
```

These theorem statements are rather involved, e.g., linear dependence is spelled out with explicit arguments *module.lin_dep class_ring ... vs*. The issue here is that the notion of linear dependence *lin_dep* is formalized in a so-called *locale* in Isabelle/HOL with respect to a specific ring. Intuitively, locales

in Isabelle are similar to modules; they are often used when formalizing algebraic structures as they provide a convenient way to specify and manage local variables and assumptions [2]. While this system facilitates managing results *within* the same locale, as the conversions shown above involve the concept of linear dependence over two different rings, we need explicit arguments for two different locale instances. Note, also, that the `rat_to_int` direction is slightly subtle mathematically; one needs to turn a rational linear combination of vectors into a corresponding integer linear combination by multiplying out all denominators.

### 4.2 Arithmetic

The proof of Coppersmith's method involves considerable arithmetic; the proof of Howgrave-Graham's theorem relies on arithmetic, as does establishing that the bound required to use Howgrave-Graham's theorem holds for the lattice used in Coppersmith's method. Although straightforward on paper, formalizing arithmetic can pose some challenges. As one example, proving that the bound required to use Howgrave-Graham's theorem applies to the Coppersmith lattice requires the following lemma, which says that for real $x$ with $0 < x \leq 0.18$, $(1 + \frac{1}{x})^x \leq \sqrt{2}$.

```
lemma coppersmith_arithmetic_convergence:
  fixes x:: "real"
  assumes x: "0 < x" "x ≤ 0.18"
  shows "(1 + (1/x)) powr (x) ≤ sqrt 2"
```

Intuitively, as the function $x \mapsto (1 + \frac{1}{x})^x$ is increasing for $0 < x \leq 0.18$, the result holds because $(1 + \frac{1}{0.18})^{0.18} \leq \sqrt{2}$. The main proof strategy we adopted here (and also in other arithmetic lemmas) was to get rid of irrational terms and to remove non-natural number exponents by taking appropriate powers on both sides of the (in)equalities to be proved. Our proof of this is in three parts:

- We prove a slightly tighter result (without the square root on the RHS), namely: for $0 < x \leq 0.18$, $(1 + 1/x)^{1/(1/x)} \leq 1.414$.
- By a change of variable, we show that the function $y \mapsto (1+y)^{1/y}$ is increasing on the interval $[1/0.18, \infty)$ by showing that its first derivative is non-negative on that interval.
- Then, we prove the bound at $x = 0.18$ by restating the goal as $(1414/10^3) \geq (1 + 1/(18/(10^2)))^{9/50}$ and proving $(1414/10^3)^{50} \geq ((1 + 1/(18/10^2)))^9$; Isabelle can discharge this latter subgoal with Sledgehammer [45].[7]

Formalizing arithmetic like this may seem tedious and unnecessary, but elaborating the low-level details required by a proof assistant is useful for ensuring correctness. Indeed, in the process of formalizing the arithmetic, we encountered a small error in the textbook we were following. In Galbraith [22], the

[7] An alternative method is to use Isabelle/HOL's approximation tactic [25], but this requires trusting Isabelle/HOL's (unverified) code generator.

first line of Theorem 19.1.9 (the theorem associated to Coppersmith's method) bounds $\epsilon < \min\{\frac{1}{d}, 0.18\}$, but for the arithmetic to work out correctly, we need $0 < \epsilon < \min\{0.18(\frac{d-1}{d}), \frac{1}{d}\}$ (this change affects some lines in the proof as well). This is now included in the errata for the book.[8]

### 4.3 Library Contributions

The existing LLL formalization [13,19,51] proves a number of key invariants of the algorithm. We contribute the following upper bound on the length of the shortest vector produced by LLL reduction in terms of the Gram determinant (a generalized version of the determinant [51]). This bound follows from the existing weakly-reduced property of the LLL-reduced basis matrix [19], which bounds the sizes of vectors in the output matrix sequentially.

```
lemma short_vector_det_bound:
  assumes m: "m ≠ 0"
  assumes k: "k ≤ m"
  shows "(rat_of_int ‖short_vector‖²) ^ k ≤
     α ^ (k * (k-1) div 2) *
     rat_of_int (gs.Gramian_determinant reduce_basis k)"
```

Here, `reduce_basis` is the reduced basis matrix produced by the LLL algorithm, $\alpha$ is a parameter of the algorithm, and `short_vector` is the shortest vector produced by LLL (which is the first row in `reduce_basis`). The Gram determinant for a matrix $A$ is $\det(A^T A)$, and this quantity was originally used by Divasón et al. [19] for termination analysis of the LLL algorithm; note that the Gram determinant of $A$ is equal to the square of the determinant if $A$ is a square matrix.

We also formalize the well-known result that two different sets of basis vectors for the same lattice have the same Gram determinant.

```
lemma lattice_of_eq_gram_det_eq:
  fixes a b::"'a vec list"
  assumes a: "distinct a" "lin_indpt (set a)" "set a ⊆ carrier_vec n"
  assumes b: "set b ⊆ carrier_vec n" "length a = length b"
  assumes lat_eq: "lattice_of a = lattice_of b"
  defines A: "A ≡ mat_of_cols n a"
  defines B: "B ≡ mat_of_cols n b"
  shows "det (A^T * A) = det (B^T * B)"
```

Since LLL reduction preserves the input lattice, the two results above imply that the determinant appearing in the RHS bound of `short_vector_det_bound` can be equivalently replaced by the determinant of the input matrix to LLL. In the case of Coppersmith's method, this means that we can bound the length of the short vector using only the determinant of the input matrix. Moreover, since the method uses a lower triangular matrix, a closed-form expression for the input determinant is easily derived as the product of entries on the diagonal.

[8] https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html

### 4.4 Retrospective Generalization

Our initial formalization of the two versions of Coppersmith's method closely followed Galbraith [22], as we presented in Sect. 3. We found it beneficial to revisit our code and factor out the common argument underlying both the lightweight and general versions of Coppersmith's method. This helped simplify our proofs and also allowed us to lift the key mathematical argument behind Coppersmith's method into a reusable locale so that our work is easier to build upon for formalizing other Coppersmith-like arguments. As a rough point of comparison, the initial formalization consisted of two separate proof files with $4411 (= 2827 + 1584)$ lines of code. The refactored formalization consists of a file formalizing the shared argument and two specialized files that specialize the argument to specific choices of lattices, totaling $2912 (= 967 + 780 + 1165)$ lines.[9]

We set up the following locale, `coppersmith_assms`, which captures key assumptions underlying Coppersmith-like results. It extends the previously defined `LLL_with_assms` which is the locale for formalized results about LLL [13,19,51].

```
locale coppersmith_assms = LLL_with_assms +
  fixes x0 :: int
  fixes M X :: nat
  fixes F :: "int poly"
  assumes M: "M > 0" and X: "X > 0" and n: "n > 0"
  assumes x0: "poly F x0 mod M = 0" "|x0| ≤ int X"
  assumes lfs: "length fs_init ≠ 0"
  assumes Xpoly: "⋀v j. v ∈ set fs_init ⟹ j < n ⟹
    v $ j mod int X ^ j = 0"
  assumes rt: "⋀v. v ∈ set fs_init ⟹
    poly (int_poly_associated_to_vec v X) x0 mod M = 0"
```

This locale fixes an integer `x0`, a polynomial `F` with integer coefficients, and natural numbers `M` and `X`. For LLL, it also fixes natural numbers `m` and `n` and a set of `m` linearly independent input vectors, `fs_init`, each of length `n`; it also fixes an approximation factor $\alpha$ with $\alpha \geq$ `4/3` ($\alpha$ is used as an input to the LLL algorithm). The locale assumes that `M > 0`, `X > 0`, `n > 0`, `x0` is a root of `F` mod `M`, `|x0| < X`, and the length of `fs_init` (i.e., `m`) is nonzero. Finally, it assumes two well-formedness assumptions about the basis vectors: in `Xpoly`, that the `j`th entry of each vector in `fs_init` is divisible by `X^j`; in `rt`, it assumes that, for any vector in `fs_init`, `x0` is a root of the associated polynomial mod `M`.

Under the locale's assumptions, we prove the following two generic results. The first lemma says that, if the Euclidean norm of `short_vector` is bounded above by $\sqrt{M\hat{}2/n}$, then `x0` is a root of the polynomial associated to `short_vector` over the integers. The second lemma gives a condition on the Gram determinant of the input lattice under which this bound on `short_vector` holds.

[9] We only contribute the refactored files to the AFP. These counts do not include shared setup between the initial/refactored formalizations, like Howgrave-Graham's theorem and the bound from Sect. 4.3.

```
lemma root_poly_short_vector:
  assumes bnd: "real_of_int ‖short_vector‖^2 < M^2 / n"
  shows "poly (int_poly_associated_to_vec short_vector X) x0 = 0"
```

```
lemma det_bound_imp_short_vec_bound:
  assumes det_bound:"(real_of_rat α) ^ (m * (m - 1) div 2) *
    real_of_int (gs.Gramian_determinant fs_init m) * (real n) ^ m
    < M ^ (2 * m)"
  shows "real_of_int ‖short_vector‖^2 < M^2 / n"
```

Now, the refactored proofs for variants of Coppersmith's method proceed by *instantiating* the `coppersmith_assms` locale, i.e., by showing that the chosen input lattices satisfy all of the locale assumptions. In particular, these proofs do not need to handle lower-level reasoning about the LLL invariants which have been done once-and-for-all in our proofs. It is also notable that all of the locale assumptions (except the ones about `x0`) and the `bnd` assumption are computable. Overall, the upshot of the generalization is that other implementations of Coppersmith-like techniques to find small roots `x0` (e.g., ones that use different input lattices than in our work) inherit much existing proof structure by simply instantiating the locale.

We remark that the retrospective generalization here may be somewhat atypical in formalized mathematics—we could have started from the refactored locale directly instead of duplicating our proof effort. However, formalizing the initial proofs was crucial for gaining the intuition that allowed us to correctly identify minimal assumptions needed in the locale. In particular, we found that first formalizing the lightweight method and subsequently Coppersmith allowed us to easily identify which parts of the proofs were shared, so that we could then isolate out the common mathematical structure between the two methods.

## 5 Conclusion and Future Work

Our formalization of the lightweight, full, and more generalized versions of Coppersmith's method in Isabelle/HOL provide formally verified methods for finding small roots of polynomials mod $M$. This has broader impacts in cryptography as it provides a path towards formally specifying lattice-based *attack* strategies on implementations of cryptographic algorithms, particularly RSA. This would in turn enable certification of the robustness of implementations of RSA against attacks. Specification and verification in a formal methods tool like Isabelle/HOL are expensive bottlenecks to using formal proofs for the external certification of safety-critical software [49]. Our work widens the bottleneck for producing correct cryptographic implementations by providing reusable library results for formalizing lattice-reduction attacks and insights into fruitful strategies for similar formalizations.

*Future Work.* We are interested in using our formalization of Coppersmith's method to formalize several concrete attacks on RSA. Additionally, our formalization could be used in formalizations of factoring algorithms more broadly; recent theoretical work considered the use of Coppersmith's method in factoring integers [1]. It could also serve as the foundation for exploring open theoretical questions in a formal-methods setting. Extending Coppersmith's method to multivariate polynomials is largely an open problem; it would be highly compelling to have a theoretical result that is also *verified*, as even bivariate extensions of Coppersmith's method are considered somewhat intricate [22]. As another interesting future goal, we could formalize a complexity analysis of Coppersmith's method. The mathematical statement of Coppersmith's theorem provides complexity bounds on the runtime of the algorithm. The difficulty, then, is on the formalization side. However, we will benefit from prior work—although the original formalization of the LLL algorithm [19] did not involve polynomial runtime analysis, subsequent work [13] formalized this complexity result.

**Acknowledgments.** Thanks to NSF CAREER Award CNS-1552934 for supporting this work. Yong Kiam Tan was supported by A*STAR, Singapore. We also thank the anonymous CICM reviewers for useful feedback on the paper.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Ajani, Y., Bright, C.: A hybrid SAT and lattice reduction approach for integer factorization. In: Ábrahám, E., Sturm, T. (eds.) SC-Square@ISSAC. CEUR Workshop Proceedings, vol. 3455, pp. 39–43. CEUR-WS.org (2023), https://ceur-ws.org/Vol-3455/short1.pdf
2. Ballarin, C.: Locales: A module system for mathematical theories. J. Autom. Reason. **52**(2), 123–153 (2014). https://doi.org/10.1007/s10817-013-9284-7
3. Barbosa, M., Barthe, G., Bhargavan, K., Blanchet, B., Cremers, C., Liao, K., Parno, B.: SoK: Computer-aided cryptography. In: SP. pp. 777–795. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00008
4. Barth, B.: Estonia suspends national 760,000 ID cards found prone to encryption vulnerability. SC Magazine (2017), https://www.scmagazine.com/news/estonia-suspends-national-760000-id-cards-found-prone-to-encryption-vulnerability
5. Barthe, G., Grégoire, B., Heraud, S., Béguelin, S.Z.: Computer-aided security proofs for the working cryptographer. In: Rogaway, P. (ed.) CRYPTO. LNCS, vol. 6841, pp. 71–90. Springer (2011). https://doi.org/10.1007/978-3-642-22792-9_5
6. Barthe, G., Pointcheval, D., Béguelin, S.Z.: Verified security of redundancy-free encryption from Rabin and RSA. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) CCS. pp. 724–735. ACM (2012). https://doi.org/10.1145/2382196.2382272, https://doi.org/10.1145/2382196.2382272

7. Basin, D.A., Cremers, C., Dreier, J., Sasse, R.: Tamarin: Verification of large-scale, real-world, cryptographic protocols. IEEE Secur. Priv. **20**(3), 24–32 (2022). https://doi.org/10.1109/MSEC.2022.3154689
8. Basin, D.A., Lochbihler, A., Sefidgar, S.R.: CryptHOL: Game-based proofs in higher-order logic. J. Cryptol. **33**(2), 494–566 (2020). https://doi.org/10.1007/S00145-019-09341-Z
9. Blanchet, B., Cheval, V., Cortier, V.: ProVerif with lemmas, induction, fast subsumption, and much more. In: SP. pp. 69–86. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833653
10. Boneh, D.: Simplified OAEP for the RSA and Rabin functions. In: Kilian, J. (ed.) CRYPTO. LNCS, vol. 2139, pp. 275–291. Springer (2001). https://doi.org/10.1007/3-540-44647-8_17, https://doi.org/10.1007/3-540-44647-8_17
11. Boston, B., Breese, S., Dodds, J., Dodds, M., Huffman, B., Petcher, A., Stefanescu, A.: Verified cryptographic code for everybody. In: Silva, A., Leino, K.R.M. (eds.) CAV. LNCS, vol. 12759, pp. 645–668. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_31
12. Bottesch, R., Divasón, J., Thiemann, R.: Two algorithms based on modular arithmetic: lattice basis reduction and Hermite normal form computation. Archive of Formal Proofs (March 2021), https://isa-afp.org/entries/Modular_arithmetic_LLL_and_HNF_algorithms.html, Formal proof development
13. Bottesch, R., Haslbeck, M.W., Thiemann, R.: A verified efficient implementation of the LLL basis reduction algorithm. In: Barthe, G., Sutcliffe, G., Veanes, M. (eds.) LPAR. EPiC Series in Computing, vol. 57, pp. 164–180. EasyChair (2018). https://doi.org/10.29007/XWWH
14. Butler, D.: Formalising cryptography using CryptHOL. Ph.D. thesis, University of Edinburgh, UK (2020). https://doi.org/10.7488/ERA/510
15. Butler, D., Aspinall, D., Gascón, A.: Formalising oblivious transfer in the semi-honest and malicious model in CryptHOL. In: Blanchette, J., Hritcu, C. (eds.) CPP. pp. 229–243. ACM (2020). https://doi.org/10.1145/3372885.3373815
16. Butler, D., Lochbihler, A., Aspinall, D., Gascón, A.: Formalising $\sum$-protocols and commitment schemes using CryptHOL. J. Autom. Reason. **65**(4), 521–567 (2021). https://doi.org/10.1007/S10817-020-09581-W
17. Canan, J.W.: Defending against cyber threats. Aerospace America pp. 22–27, 41 (October 2011), https://www.aiaa.org/docs/default-source/uploadedfiles/publications/aerospace-america-october-2011.pdf?sfvrsn=3062b789_2
18. Coppersmith, D.: Finding a small root of a univariate modular equation. In: Maurer, U.M. (ed.) EUROCRYPT. LNCS, vol. 1070, pp. 155–165. Springer (1996). https://doi.org/10.1007/3-540-68339-9_14
19. Divasón, J., Joosten, S.J.C., Thiemann, R., Yamada, A.: A formalization of the LLL basis reduction algorithm. In: Avigad, J., Mahboubi, A. (eds.) ITP. LNCS, vol. 10895, pp. 160–177. Springer (2018). https://doi.org/10.1007/978-3-319-94821-8_10
20. Divasón, J., Joosten, S.J.C., Thiemann, R., Yamada, A.: 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
21. Erbsen, A., Philipoom, J., Gross, J., Sloan, R., Chlipala, A.: Simple high-level code for cryptographic arithmetic - with proofs, without compromises. In: SP. pp. 1202–1219. IEEE (2019). https://doi.org/10.1109/SP.2019.00005

22. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press (2012), https://www.math.auckland.ac.nz/%7Esgal018/crypto-book/crypto-book.html
23. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. IACR Cryptol. ePrint Arch. p. 181 (2005)
24. Haselwarter, P.G., Rivas, E., Muylder, A.V., Winterhalter, T., Abate, C., Sidorenco, N., Hritcu, C., Maillard, K., Spitters, B.: SSProve: A foundational framework for modular cryptographic proofs in Coq. ACM Trans. Program. Lang. Syst. **45**(3), 15:1–15:61 (2023). https://doi.org/10.1145/3594735
25. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: ACM SIGSAM PLMMS. pp. 38–45 (2009)
26. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) Cryptography and Coding, 6th IMA International Conference, Cirencester, UK, December 17-19, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1355, pp. 131–142. Springer (1997). https://doi.org/10.1007/BFB0024458
27. Huch, F., Krauss, A.: Findfacts: A scalable theorem search. CoRR **abs/2204.14191** (2022). https://doi.org/10.48550/ARXIV.2204.14191
28. Kammüller, F.: Higher order model checking in isabelle for human centric infrastructure security (2023), https://arxiv.org/abs/2312.17555
29. Koblitz, N., Menezes, A.: Critical perspectives on provable security: Fifteen years of "another look" papers. Adv. Math. Commun. **13**(4), 517–558 (2019). https://doi.org/10.3934/amc.2019034
30. Kosaian, K., Tan, Y.K.: Formalizing Coppersmith's method. Archive of Formal Proofs (June 2024), https://www.isa-afp.org/entries/Coppersmith_Method.html, Formal proof development
31. Koutsoukou-Argyraki, A.: Formalising mathematics–in praxis; a mathematician's first experiences with Isabelle/HOL and the why and how of getting started. Jahresbericht der Deutschen Mathematiker-Vereinigung **123**, 3–26 (2021). https://doi.org/10.1365/s13291-020-00221-1
32. Kreuzer, K.: Verification of correctness and security properties for CRYSTALS-KYBER. IACR Cryptol. ePrint Arch. p. 87 (2023), https://eprint.iacr.org/2023/087
33. Kreuzer, K., Nipkow, T.: Verification of NP-hardness reduction functions for exact lattice problems. In: Pientka, B., Tinelli, C. (eds.) CADE. Lecture Notes in Computer Science, vol. 14132, pp. 365–381. Springer (2023). https://doi.org/10.1007/978-3-031-38499-8_21
34. Lenstra, A., Lenstra, H., László, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261** (12 1982). https://doi.org/10.1007/BF01457454
35. Lindenberg, C., Wirt, K., Buchmann, J.: Formal proof for the correctness of RSA-PSS. IACR Cryptol. ePrint Arch. p. 11 (2006), http://eprint.iacr.org/2006/011
36. Lochbihler, A., Sefidgar, S.R.: A tutorial introduction to CryptHOL. IACR Cryptol. ePrint Arch. p. 941 (2018), https://eprint.iacr.org/2018/941
37. Martin-Dorel, É.: Contributions to the Formal Verification of Arithmetic Algorithms. (Contributions à la vérification formelle d'algorithmes arithmétiques). Ph.D. thesis, École normale supérieure de Lyon, France (2012), https://tel.archives-ouvertes.fr/tel-00745553

38. Martin-Dorel, É., Hanrot, G., Mayero, M., Théry, L.: Formally verified certificate checkers for hardest-to-round computation. J. Autom. Reason. **54**(1), 1–29 (2015). https://doi.org/10.1007/S10817-014-9312-2, https://doi.org/10.1007/s10817-014-9312-2
39. Nemec, M., Sýs, M., Svenda, P., Klinec, D., Matyas, V.: The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS. pp. 1631–1648. ACM (2017). https://doi.org/10.1145/3133956.3133969
40. Nguyen, P.Q., Vallée, B. (eds.): The LLL Algorithm - Survey and Applications. Information Security and Cryptography, Springer (2010). https://doi.org/10.1007/978-3-642-02295-1
41. Nipkow, T., Klein, G.: Isar: a Language for Structured Proofs, pp. 3–69. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10542-0_5
42. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002). https://doi.org/10.1007/3-540-45949-9
43. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reason. **5**(3), 363–397 (1989). https://doi.org/10.1007/BF00248324
44. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Comput. Secur. **6**(1-2), 85–128 (1998). https://doi.org/10.3233/JCS-1998-61-205
45. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL. EPiC Series in Computing, vol. 2, pp. 1–11. EasyChair (2010). https://doi.org/10.29007/36DT
46. Petcher, A., Morrisett, G.: The foundational cryptography framework. In: Focardi, R., Myers, A.C. (eds.) POST. LNCS, vol. 9036, pp. 53–72. Springer (2015). https://doi.org/10.1007/978-3-662-46666-7_4
47. Protzenko, J., Parno, B., Fromherz, A., Hawblitzel, C., Polubelova, M., Bhargavan, K., Beurdouche, B., Choi, J., Delignat-Lavaud, A., Fournet, C., Kulatova, N., Ramananandro, T., Rastogi, A., Swamy, N., Wintersteiger, C.M., Béguelin, S.Z.: EverCrypt: A fast, verified, cross-platform cryptographic provider. In: SP. pp. 983–1002. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00114
48. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342
49. Rozier, K.Y.: Specification: The biggest bottleneck in formal methods and autonomy. In: VSTTE. LNCS, vol. 9971, pp. 1–19. Springer-Verlag, Toronto, ON, Canada (July 2016). https://doi.org/10.1007/978-3-319-48869-1_2
50. Stathopoulos, Y., Koutsoukou-Argyraki, A., Paulson, L.C.: SErAPIS: A concept-oriented search engine for the Isabelle libraries based on natural language. In: Online proceedings of the Isabelle Workshop affiliated to IJCAR 2020 (virtual) (2020), https://www.cl.cam.ac.uk/~lp15/papers/Alexandria/Serapis.pdf
51. Thiemann, R., Bottesch, R., Divasón, J., Haslbeck, M.W., Joosten, S.J.C., Yamada, A.: Formalizing the LLL basis reduction algorithm and the LLL factorization algorithm in Isabelle/HOL. J. Autom. Reason. **64**(5), 827–856 (2020). https://doi.org/10.1007/S10817-020-09552-1
52. Trappe, W., Washington, L.C.: Introduction to Cryptography with Coding Theory (2nd Edition). Prentice-Hall, Inc., USA (2005)

53. Unruh, D.: Quantum relational hoare logic. Proc. ACM Program. Lang. **3**(POPL), 33:1–33:31 (2019). https://doi.org/10.1145/3290346
54. Van Hoeij, M.: Factoring polynomials and the knapsack problem. Journal of Number theory **95**(2), 167–189 (2002)
55. Werner, D.: Hackers as allies. Aerospace America pp. 24–30 (June 2020), https://aerospaceamerica.aiaa.org/features/hackers-as-allies/
56. Whitley, A.: Cryptographic standards. Archive of Formal Proofs (June 2023), https://isa-afp.org/entries/Crypto_Standards.html, Formal proof development
57. Wübben, D., Seethaler, D., Jaldén, J., Matz, G.: Lattice reduction. IEEE Signal Process. Mag. **28**(3), 70–91 (2011). https://doi.org/10.1109/MSP.2010.938758
58. Zinzindohoué, J.K., Bhargavan, K., Protzenko, J., Beurdouche, B.: HACL*: A verified modern cryptographic library. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) CCS. pp. 1789–1806. ACM (2017). https://doi.org/10.1145/3133956.3134043