Cryptanalysis of an Outsourced Modular Inversion Protocol

Charles Bouillaguet

Abstract—Public-key cryptographic primitives involve mathematical operations that are computationally intensive for devices with limited resources. A typical approach is to offload time-consuming operations from a (computationally weak) client device to an untrusted yet computationally powerful server. Such a delegation protocol needs to achieve the privacy of the server's inputs. Recently, Tian, Yu, Zhang, Xue, Wang and Ren [*IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 241–253, 2022] proposed a unimodular matrix transformation technique to realize secure outsourcing of modular inversion. We present an elementary cryptanalysis of their protocol and prove that it does not achieve the claimed security guarantees.

Index Terms—Cryptanalysis, outsourcing protocol, modular inversion, Euclidean lattices

1 Introduction

Computing modular inverse is a very common operation in public-key cryptography. It is required to generate RSA key pairs, to create DSA signatures, to perform the group operation on elliptic curves in some coordinate systems, etc. Since the computational resources can be very limited on certain devices, it is natural, as most of the devices are online or directly connected to a powerful device, to consider securely delegating some sensitive and costly operations to another, untrusted but more powerful device.

The problem of outsourcing cryptographic operations has already received a lot of attention but there has been a recent regain of interest with the development of cloud computing. In 2005, Hohenberger and Lysyanskaya [1] proposed formal security definitions for securely outsourcing computations from a computationally limited device, called the *client*, to untrusted helpers, called the *servers*. Delegating a cryptographic operation presents many risks since they usually involve sensitive information which should not be revealed to potential adversaries. Moreover, since the servers are not fully trusted, a delegation protocol should enable clients to verify the correctness of the result returned by the server with high probability.

A delegation protocol must therefore usually meet two security objectives: *privacy* —the actual computation that the client is delegating should not be revealed to a passive adversary— and *verifiability* —a malicious server should not be able to make the client accept an invalid value as the result of the delegated computation. Obviously, to be of practical interest, these delegation protocols must have a computational cost for the client lower than that of the delegated computation.

In a modular inversion delegation protocol, the client has two (potentially large) integers a and n and uses the help of a server to compute $a^{-1} \bmod n$. For instance, in order to generate an RSA key pair, it is common to compute the secret exponent as $d \leftarrow e^{-1} \bmod (p-1)(q-1)$. The public exponent e usually has a fixed value (very often $e = 65537$). The result d of the computation is the secret key. If either d or $(p-1)(q-1)$ leaks, then the resulting key pair offers no security at all. It is common to manipulate numbers of 2048 or even 4096 bits when generating RSA key pairs.

Protocols to delegate the computation of a modular inverse have been proposed by Su, Yu, Tian, Zhang and Hao [2] and more recently by Tian, Yu, Zhang, Xue, Wang and Ren [3]. The former requires the modulus to be the product of only two prime numbers (*i.e.* $n = pq$); this factorization must be known to the client; the delegation protocol requires two non-colluding servers to be secure. The latter protocol lift these restrictions and is claimed to be secure for arbitrary values of n using a single untrusted server.

The protocol proposed in [3] is simple and the authors claim that it is secure (privacy and verifiability). We show that a passive adversary can easily reconstruct the content of the delegated computation. We provide a complete python program implementing the attack. It runs almost as fast as actually performing the modular inversion. The protocol is thus broken beyond repair. As a final nail in its coffin, we show how the client could actually perform the modular inversion by itself and *faster* than by running the proposed delegation protocol, in some relevant use cases.

2 Description of the Protocol

We briefly describe the protocol proposed in [3]. Let k be a security parameter. It is assumed that the security offered by the protocol increases with k (and its efficiency decreases with k). The client has two relatively prime ℓ-bit numbers a, n. If used in the context of generating RSA key pairs, then $\ell \geq 2000$. In order to compute $a^{-1} \bmod n$, the client:

- Generates two random coprime k-bit integers x and y, then runs the extended Euclidean algorithm to compute (z, t) satisfying the Bezout relation $xz - yt = 1$. This preprocessing stage could be done in advance, before the values of a and n are known, but it has to be repeated for each new modular inversion.

- C. Bouillaguet was with Sorbonne Université, CNRS, LIP6, F-75005 Paris, France.

- Computes
$$\begin{pmatrix} a' \\ n' \end{pmatrix} = \begin{pmatrix} x & z \\ y & t \end{pmatrix} \begin{pmatrix} a \\ n \end{pmatrix}.$$
Sends (a', n') to the server.

The server:

- Runs the extended Euclidean algorithm to computes the Bezout relation $ua' + vn' = 1$. Sends back (u, v) to the client.

Finally the client:

- Checks that $ua' + vn' = 1$. If it is the case, it emits $(ux + vy) \bmod n$. Otherwise, it emits $\perp$.

The protocol is clearly correct: the client always emits either $a^{-1} \bmod n$ or $\perp$, and it always return the modular inverse when the server runs the protocol correctly.

Theorem 5 in [3] asserts that using the outsourcing protocol enables the client to compute modular inverses with $\mathcal{O}(\ell)$ less operations than directly using the fastest know algorithm.

Because the client and the server communicate over a presumably insecure channel, the messages they exchange are assumed to be public. Yet it is intended that the values manipulated by the client (here a and n) remain private. The authors of [3] claim that the protocol satisfies the notion of *input/output privacy*. More precisely, theorem 4 in [3] states that any probabilistic algorithm with polynomial running time (in k) that receives the server input (a' and n') cannot emit the client values (a and n), except with negligible probability (in k). In other terms, when the "security parameter" k (the bit size of the blinding factors x, y, z, t) increase linearly, then the performance of any adversary against the privacy of the protocol should collapse. The authors suggest $k \approx 80$, because exhaustively searching 2^{80} values seems practically infeasible.

3 Complexity Analysis

The authors of [3] justify the proposed delegation protocol by the fact that it would allow the client to do asymptotically less operations than directly computing modular inverses. In this section, we show that this is incorrect, and that using the protocol can only yield a constant acceleration in the most favorable case. This contradicts theorem 5 in [3] and shows that its proof is flawed.

In [3], the authors use only classical "schoolbook" algorithms for arithmetic operations. For the sake of comparison, we will do the same. Let x (resp. y) denote an n-bit integer (resp. m-bit). There exist two constants λ, μ such that:

- Computing $x + y$ or $x - y$ requires less than $\lambda(n + m + \mu)$ operations.
- Computing xy requires less than $\lambda(n + \mu)(m + \mu)$ operations.
- If y is the quotient of the Euclidean division of some number by x, then performing the division (*i.e.* computing the quotient and the remainder) requires less than $\lambda(n + \mu)(m + \mu)$ operations.

Computing inverse modulo arbitrary numbers can be done with the extended Euclidean algorithm. The heart of the problem is that it is incorrectly claimed in [3] that the extended Euclidean algorithm requires $\mathcal{O}(\ell^3)$ operations when executed on two ℓ-bit numbers. It is well-known that its running time is in fact $\mathcal{O}(\ell^2)$, even using schoolbook algorithms for arithmetic operations. In fact, we have the following

Theorem 1. *The extended Euclidean algorithm, when applied to $0 \leq a < n$ terminates after $\mathcal{O}(\mathrm{len}(a) \cdot \mathrm{len}(n))$ bit operations, even using schoolbook algorithm to implement multi-precision arithmetic operations. It returns (g, u, v) where $g = ua + vn$, g is the greatest common divisor of a and n, $|u| \leq n$ and $|v| \leq a$.*

A proof appears for instance in [4, §4.2]. This theorem can be improved, as the asymptotic complexity of modular inversion is actually subquadratic. The first quasilinear algorithm for GCD computations (and hence modular inversion) was given by Knuth in 1971 [5]. See also [6], [7].

Theorem 1 result indicates that computing a $a^{-1} \bmod n$ is only a constant factor more expensive than computing an using schoolbook multiplication. Both a theoretical analysis of the number of operations and empirical experiments with the GNU Multi-Precision library (comparing the relative speed of the `mpz_mul` and the `mpz_invert` functions) suggest that modular inversion is about 20 times slower than multiplication.

Computing the inverse of a modulo an ℓ-bit modulus n thus requires $\mathcal{O}(\ell^2)$ operations in general. In the protocol, the integers u and v have length $\ell + k$. The protocol requires the client to compute ua' and vn', in other terms to multiply $(k+\ell)$-bit numbers, which costs $\mathcal{O}((k+\ell)^2)$. It follows that the protocol can only save a constant fraction of the work in the best case.

We wrote a simple-minded C implementation of the protocol using the GMP library (just like the authors of [3]), and we benchmarked it with $\ell = 2048$ and $k = 128$. We find that inverting a random 2048-bit a modulo another random 2048-bit number n requires 9.7μs on the author's laptop. The protocol takes total time 12.5μs, of which 11.9μs are done by the server and 1.6μs are done by the client. Using the protocol thus saves a factor of about 6 in computation time for the client, in this particular case. This is in line with the experimental results presented in [3].

The authors of [3] explicitly discuss the relevant setting of delegating the inversion that takes place when generating RSA key pairs. In this case, it is fairly common to compute the inverse of a small number (typically $a = 65537$). Theorem 1 shows that computing the inverse is even more efficient in this case, as it only needs $\mathcal{O}(\ell)$ operations. Running the protocol is then asymptotically more expensive for the client than directly computing the modular inverse. Our practical experiments indicate that the inversion requires 0.29μs, while the protocol takes 5.0μs, of which 1.5μs are done by the client. Using the protocol is then 5 times less efficient than doing the computation directly.

In all cases, the asymptotic benefit of using the protocol is at most $\mathcal{O}(1)$ and not $\mathcal{O}(\ell)$ as advertised in [3].

4 A Polynomial-Time Attack on Privacy

This section describes a deterministic polynomial-time algorithm (the "attack") that recovers the client secrets (a, n)

from the server input (a', n'). As such, it invalidates the security claims made in [3], and shows that the proof of theorem 4 in [3] is flawed. The attack is very practical, easy to implement and runs in no measurable time.

The problem in the security proof lies in the (erroneous) claim that (a', n') can be seen as the "*one-time pad encryption*" of (a, n) with key (x, y, z, t), from which the authors deduce that "*consequently, no information will be leaked to the adversary*". This reasoning is incorrect, if only because it is well-known that perfect secrecy requires the key to be at least as long as the plaintext, a condition which is clearly not satisfied here.

Indeed, our attack works under the condition that $k \leq \ell$, in other terms that the "blinding factors" x, y, z, t are smaller than the modulus n used when computing the inverse of a. If this condition did not hold, then the "preprocessing stage" where the client computes (z, t) using the extended Euclidean algorithm would be more costly than directly inverting a modulo n. As such, the protocol would be pointless.

The attack recovers the blinding factors x, y, z, t from the public input (a', n'). This is enough to recover a and n because of the relation:

$$\begin{pmatrix} a \\ n \end{pmatrix} = \begin{pmatrix} t & -z \\ -y & x \end{pmatrix} \begin{pmatrix} a' \\ n' \end{pmatrix}. \tag{1}$$

Equation (1) holds because $xt - yz = 1$ (the matrix is unimodular, and thus invertible over $\mathbb{Z}$). Consider the following integer linear program:

$$\begin{cases} |\gamma a' + \delta n'| & \leq 2^{\ell} \\ \quad |\gamma|, |\delta| & \leq 2^{k} \end{cases} \tag{2}$$

It follows from (1) that $(t, -z)$ and $(-y, x)$ are both solutions of (2). In section 5, we will prove the following

Theorem 2. *If $k \leq \ell$, the linear program* (2) *admits at most $24 \cdot 2^{\ell+k}/n' + 1$ solutions. Computing them requires $\mathcal{O}\left(\ell^2 \left(1 + 2^{\ell+k}/n'\right)\right)$ bit operations.*

Because n' is expected to be very close to $2^{\ell+k}$, we thus expect the number of solutions of (2) to be quite small. Given this result, an adversary could run the following procedure to recover the secrets:

- Determine the set Ω of all solutions of (2).
- For each pair $(\alpha, \beta), (\gamma, \delta)$ of solutions in Ω, set $a \leftarrow \alpha a' + \beta n'$ and $n \leftarrow \gamma a' + \delta n'$; if $|\alpha\delta - \beta\gamma| = 1$ and $0 \leq a < n$, then output (a, n).

Because $(t, -z)$ and $(-y, x)$ are both solutions of (2), this algorithms necessarily outputs (a, n). It may produce several other outputs; it is left to an external mechanism to determine which output is really the good one. This in turn depends on the context in which the inversion delegation takes place. Once Ω has been determined, the complexity of this procedure is that of performing less than $576 \cdot (2^{\ell+k}/n')^2$ arithmetic operations on $(\ell + k)$-bit numbers. This dominates the total cost of the attack. It is also an upper bound on the number of solutions that can be emitted.

Note that up to a constant factor, the cost of the attack is the same as the cost of performing the legitimate operations using classical "schoolbook" algorithms for arithmetic operations.

5 Euclidean Lattices and Linear Programs

This section proves theorem 2 with a reasoning based on Euclidean lattices. Our presentation is entirely self-contained, yet some familiarity with the basic concepts of the geometry of numbers cannot hurt. The interested reader will find some background in [8].

The approach we use is not very original. The connection between Euclidean lattice basis reduction and linear integer programming was observed as early as 1981 and used to obtain important complexity results [9]. See also [10].

5.1 Preliminaries

Consider the Euclidean lattice $\mathcal{L}$ spanned by the rows of

$$M = \begin{pmatrix} a' & 2^{\ell-k} & 0 \\ n' & 0 & 2^{\ell-k} \end{pmatrix}.$$

In other terms, $\mathcal{L}$ is the subset of $\mathbb{R}^3$ formed by all linear combinations with integer coefficients of the two rows of M (see fig. 1). M can be built from public information, and therefore is known to the attacker.

These two vectors are clearly linearly independent, therefore the linear subspace of $\mathbb{R}^3$ spanned by the rows of M has dimension two. We thus say that the lattice $\mathcal{L}$ has dimension two as well. Indeed, the lattice is embedded into the plane that goes through the origin and is orthogonal to $(2^{\ell-k}, -a', -n')$.

The main reason for introducing Euclidean Lattices is the following. Each solution (γ, δ) of the integer linear program (2) corresponds to a vector

$$(\gamma, \delta)M = (\gamma a' + \delta n', 2^{\ell-k}\gamma, 2^{\ell-k}\delta) \tag{3}$$

that belongs to the lattice $\mathcal{L}$. These vectors are all distinct because the rows of M are linearly independent. Moreover, these vectors have norm bounded by $\sqrt{3} \cdot 2^{\ell}$ because each coordinate is less than 2^{ℓ} in absolute value.

This observation is useful, because the main idea that enables us to control the number of solutions of the linear program (2) is the following: vectors that correspond to solutions of (2) are among the shortest ones of the lattice $\mathcal{L}$, and this lattice can only contain a limited number of short vectors (see fig. 1). Furthermore, they are easy to find.

Let us now get into the details. If U is a unimodular matrix, then the rows of UM span the same Euclidean lattice as that of M (the converse is true). This implies that the lattice $\mathcal{L}$ is also spanned by the rows of

$$M' = \begin{pmatrix} t & -z \\ -y & x \end{pmatrix} M = \begin{pmatrix} a & t2^{\ell-k} & -z2^{\ell-k} \\ n & -y2^{\ell-k} & x2^{\ell-k} \end{pmatrix}.$$

Note that M' contains the secret information and is therefore unknown to the attacker. The rows of M' are linearly independent (because those of M are). As argued above, the norm of each of these two new basis vectors is less than $\sqrt{3} \cdot 2^{\ell}$.

A Euclidean Lattice is a discrete structure. As such, it has a non-zero vector of minimal norm, that we call the "shortest vector" of the lattice (see fig. 1). There may be several non-zero vectors of minimum norm, but we just pick one arbitrarily. If $\mathbf{r}$ is the shortest vector of $\mathcal{L}$, then let $\mathbf{s}$ denote a shortest non-zero vector that is not colinear with

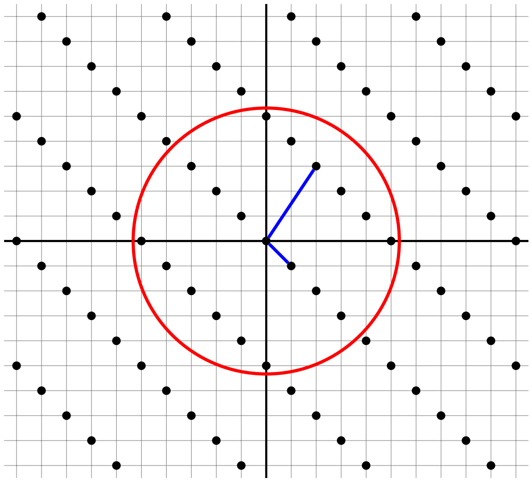

Fig. 1. A Euclidean lattice with the two shortest vectors. There is a limited supply of vectors of bounded norm (inside the red circle).

```
function LagrangeReduce($\mathbf{u}, \mathbf{v}$)
    repeat
        $q \leftarrow \lfloor \mathbf{u} \cdot \mathbf{v} \,/\, \|\mathbf{u}\|^2 \rceil$
        $\mathbf{w} \leftarrow \mathbf{v} - q\mathbf{u}$
        $\mathbf{v} \leftarrow \mathbf{u}$
        $\mathbf{u} \leftarrow \mathbf{w}$
    until $\|\mathbf{u}\| \geq \|\mathbf{v}\|$
    return $(\mathbf{v}, \mathbf{u})$
```

Fig. 2. Lagrange's lattice reduction algorithm (cubic version).

$\mathbf{r}$ (again, there may be several). This $\mathbf{s}$ is called the "second shortest vector" of $\mathcal{L}$.

A two-dimensional lattice always admits a basis composed of the two shortest vectors. This result is well-known but non-trivial (it is not true in dimension 5 or greater). More precisely, we have the well-known

Theorem 3. *Let* $(\mathbf{u}, \mathbf{v})$ *denote two vectors with integer coordinates, and let* $\mathcal{M}$ *denote the Euclidean lattice spanned by* $(\mathbf{u}, \mathbf{v})$.

Lagrange's reduction algorithm (fig. 2) computes another basis $(\mathbf{r}, \mathbf{s})$ *of* $\mathcal{M}$ *such that* $\mathbf{r}$ *(resp.* $\mathbf{s}$*) is the shortest (resp. second shortest) vector of the lattice.*

The running time of the algorithm is quadratic in the bit length of the input vectors.

The interested reader will find a proof in appendix A. Lagrange's algorithm first appeared in [11].

It follows from theorem 3 and the bounds on the norms of the rows of M' that the lattice $\mathcal{L}$ admits a minimal basis $(\mathbf{r}, \mathbf{s})$ where

$$\|\mathbf{r}\| \leq \|\mathbf{s}\| \leq \sqrt{3} \cdot 2^{\ell}. \tag{4}$$

In addition, obtaining $\mathbf{r}$ and $\mathbf{s}$ from the input is computationally easy for the server: it just has to run Lagrange's algorithm, which takes time $\mathcal{O}\left(\ell^2\right)$.

Let us next introduce another property of Euclidean lattices. The Gram matrix of the two vectors $\mathbf{u}$ and $\mathbf{v}$ is

$$MM^t = \begin{pmatrix} \|\mathbf{u}\|^2 & \mathbf{u} \cdot \mathbf{v} \\ \mathbf{u} \cdot \mathbf{v} & \|\mathbf{v}\|^2 \end{pmatrix}$$

The *volume* of a lattice spanned by the rows of a matrix M is the quantity given by

$$V = \sqrt{\det MM^t} = \sqrt{\|\mathbf{u}\|^2 \cdot \|\mathbf{v}\|^2 - (\mathbf{u} \cdot \mathbf{v})^2} \tag{5}$$

It does not depend on the choice of a particular basis matrix: if $G = UM$, then

$$\det GG^t = \det UMM^tU^t = (\det U)(\det MM^t)(\det U^t),$$

and since $\det U = \pm 1$, we find that $\det GG^t = \det MM^t$. The volume is thus an intrinsic property of the lattice, and it is easy to compute given any basis. In the case of our lattice $\mathcal{L}$, we find that

$$V = 2^{\ell-k}\sqrt{2^{2\ell-2k} + a'^2 + n'^2}. \tag{6}$$

It follows from (5) that $V \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\|$ (equality only holds if the two vectors are orthogonal). Combined with (4), this provides the following lower-bound on the length of the shortest vector

$$\|\mathbf{r}\| \geq \frac{V}{\sqrt{3} \cdot 2^{\ell}}. \tag{7}$$

The vectors of a lattice basis need not be orthogonal. Therefore, it makes sense to compute the Gram-Schmidt orthogonalization of the input basis matrix (see fig. 3). Let $\mathbf{r}^\star \leftarrow \mathbf{r}$ and $\mathbf{s}^\star \leftarrow \mathbf{s} - \mu\mathbf{r}$ with

$$\mu = \frac{\mathbf{r} \cdot \mathbf{s}}{\|\mathbf{r}\|^2}.$$

Note that we have

$$\begin{pmatrix} \mathbf{r} \\ \mathbf{s} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \mu & 1 \end{pmatrix} \begin{pmatrix} \mathbf{r}^\star \\ \mathbf{s}^\star \end{pmatrix} \tag{8}$$

The two vectors $\mathbf{r}^\star$ and $\mathbf{s}^\star$ are orthogonal. This, combined with (5) implies that

$$V = \|\mathbf{r}^\star\| \cdot \|\mathbf{s}^\star\|. \tag{9}$$

However, $\mathbf{s}^\star$ has rational entries, as opposed to $\mathbf{s}$ that has integer entries.

5.2 Enumerating Short Vectors

We now describe a procedure that enumerates all the lattice points of norm less than some given bound B. If $\mathbf{r}$ is the shortest vector of the lattice, then it is clear that there are exactly $2B/\|\mathbf{r}\|+1$ multiples of $\mathbf{r}$ inside the disk of radius B. Establishing upper bounds on the number of enumerated points will therefore necessarily require a lower-bound on the length of the shortest vector of the lattice, which is why we derived (7) in the first place.

Let $\mathbf{x} \in \mathcal{L}$ be such that $\|\mathbf{x}\| \leq B$. Because $(\mathbf{r}, \mathbf{s})$ is a lattice basis, we can write $\mathbf{x} = x_1\mathbf{r} + x_2\mathbf{s}$. Using (8), this becomes $\mathbf{x} = (x_1 + \mu x_2)\mathbf{r}^\star + x_2\mathbf{s}^\star$. The upper-bound on the norm of $\mathbf{x}$ then translates into bounds on the coefficients x_1 and x_2. Indeed, because $\mathbf{r}^\star$ and $\mathbf{s}^\star$ are orthogonal, we find that

$$\|x\|^2 = \mathbf{x} \cdot \mathbf{x} = (x_1 + \mu x_2)^2 \left\|\mathbf{r}^\star\right\|^2 + {x_2}^2 \left\|\mathbf{s}^\star\right\|^2 \leq B^2 \tag{10}$$

All the terms in the left-hand side of (10) are positive. Then using (9) we find that

$$|x_2| \leq \frac{B}{\|\mathbf{s}^\star\|} = \frac{B\,\|\mathbf{r}^\star\|}{V} = \frac{B\,\|\mathbf{r}\|}{V} \tag{11}$$

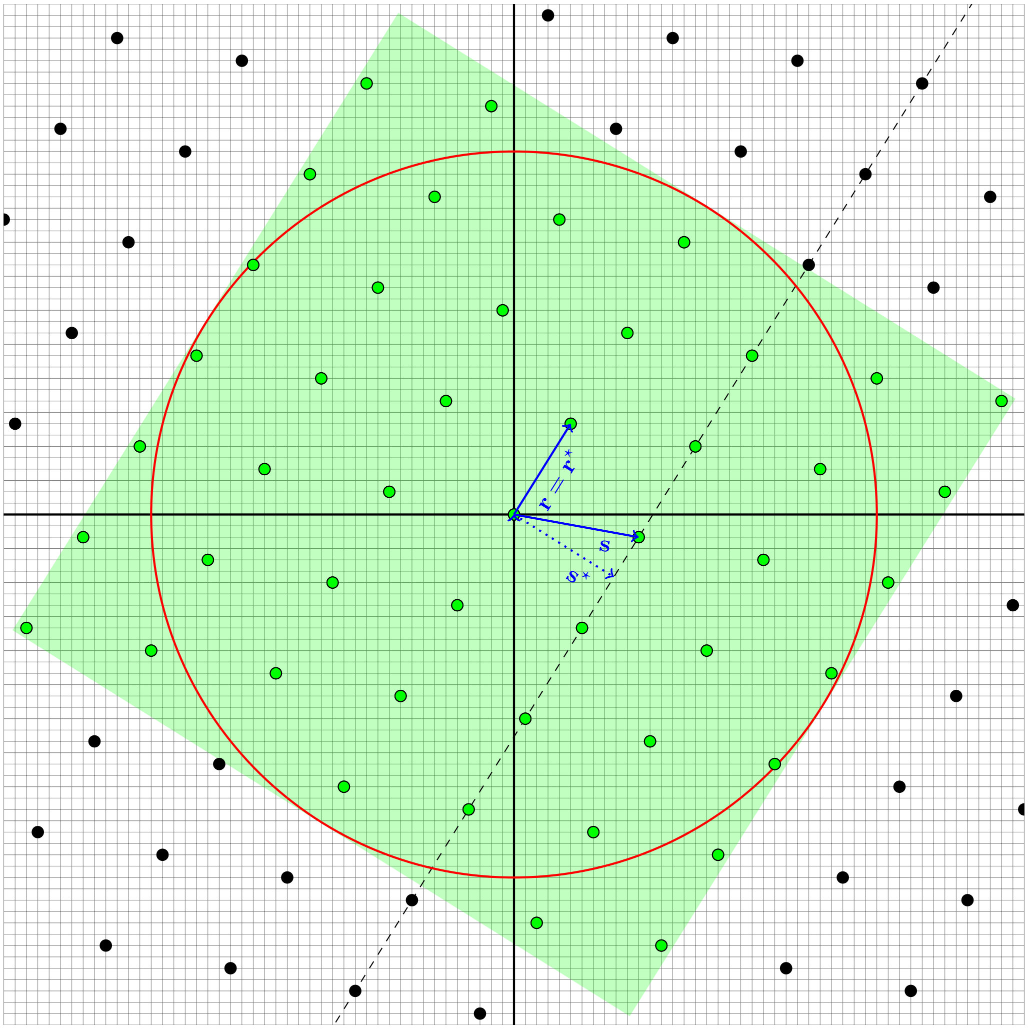

Fig. 3. $(\mathbf{r}^\star, \mathbf{s}^\star)$ is the Gram-Schmidt orthogonalization of the lattice basis $(\mathbf{r}, \mathbf{s})$. The enumeration procedure is illustrated: in order to enumerate all lattice points inside the red circle, it enumerates all the green points that are inside the square. The sides of the square are parallel to the orthogonalized basis. The circle is tangent to all sides.

This shows that x_2 is confined to live in a (hopefully) small interval that can be searched exhaustively. We also find by the same reasoning that

$$|x_1 + \mu x_2| \leq \frac{B}{\|\mathbf{r}^\star\|} = \frac{B}{\|\mathbf{r}\|} \tag{12}$$

This limits the possible values of x_1.

The culmination of all this development about Euclidean lattices is the algorithm shown in figure 4 that enumerate short vectors in a two-dimensional lattice. A geometric interpretation is in order (see fig. 3): in order to find all the lattice points inside the disk of radius B, the algorithm in fact enumerates all points inside a square of size $2B$ that contains the disk.

Combining (12) and (11) shows that the number of iterations of the innermost loop is upper-bounded by

$$\left(2\frac{B\,\|\mathbf{r}\|}{V}+1\right)\left(2\frac{B}{\|\mathbf{r}\|}+1\right) = \frac{4B^2}{V} + 2\frac{B\,\|\mathbf{r}\|}{V} + 2\frac{B}{\|\mathbf{r}\|} + 1$$

This count again admits a simple geometric interpretation. Each lattice point "occupies" an area which is equal to the volume of the lattice, so that the number of lattice points in the square is essentially the surface of the box divided by the volume of the lattice $(4B^2/V)$. In addition, it is clear that the disk necessarily contains $2B/\|\mathbf{r}\| + 1$ multiples of $\mathbf{r}$ (including the origin). Lastly, the box also contains $2B\|\mathbf{r}\|/V + 1$ multiples of $\mathbf{s}$, although this is less obvious. In addition, they may not all lie inside the disk.

```
procedure ENUMERATE($\mathbf{u}, \mathbf{v}, B$)
    $(\mathbf{r}, \mathbf{s}) \leftarrow$ LAGRANGEREDUCE($\mathbf{u}, \mathbf{v}$)
    $\mu \leftarrow \mathbf{r} \cdot \mathbf{s} \;/\; \|\mathbf{r}\|^2$
    $a \leftarrow B \cdot \|\mathbf{r}\| \;/\; V$
    $b \leftarrow B \;/\; \|\mathbf{r}\|$
    for $-\lfloor a \rfloor \leq x_2 \leq \lfloor a \rfloor$ do
        for $-\lfloor b - \mu x_2 \rfloor \leq x_1 \leq \lfloor b - \mu x_2 \rfloor$ do
            $\mathbf{x} \leftarrow x_1\mathbf{r} + x_2\mathbf{s}$
            if $\|\mathbf{x}\| \leq B$ then
                Emit $\mathbf{x}$
```

Fig. 4. Enumeration procedure that emits all vectors of norm less than B in the lattice spanned by the two vectors $(\mathbf{u}, \mathbf{v})$.

The naive enumeration algorithm given in figure 4 has long been known. It appears (under a different presentation) in the 1969 edition of *The Art of Computer Programming* [12, §3.3.4].

5.3 Enumerating Solution of (2)

To find all the solutions of the linear program (2), we set $B = \sqrt{3} \cdot 2^{\ell}$ and we use the algorithm of figure 4 to enumerate all vectors of norm less than B in the lattice $\mathcal{L}$. Note that all lattice points are written $(\gamma a' + \delta n', \gamma \cdot 2^{\ell-k}, \delta 2^{\ell-k})$. From any lattice point, we can extract (γ, δ) and check if these values satisfy (2). Indeed, all solutions of (2) yield lattice points of norm less than B, but they may also be other "spurious" lattice points inside the disk of radius B.

Thanks to (6), we lower-bound the volume by $V \geq 2^{\ell-k} n'$. In addition, using (4) and (7) we get that the number of enumerated lattice points is less than

$$24 \frac{2^{\ell+k}}{n'} + 1.$$

This is also an upper bound on number of solutions of (2).

Processing each lattice point requires a constant number of operations on vectors whose norm is less than $\sqrt{2} \cdot B$. As a consequence, using only classical ("schoolbook") algorithms to implement the arithmetic operations, the bit complexity of the procedure is

$$\mathcal{O}\left(\ell^2 + \frac{2^{\ell+k}}{n'} \ell^2\right).$$

This proves theorem 2.

6 Implementation of the Attack

This section describes a pure Python implementation of the attack described in the previous section. It is entirely self-contained and does not use any external library. It works very well in practice and recovers the secrets in no measurable time.

One notably useful feature of Python is that it supports arbitrarily large integers out of the box. This makes it easy to implement functions that process cryptographic-size values with, say, 2048 bits.

We first need some very usual routines to deal with vectors. When given integer vectors and integer scalars, these functions only perform integer arithmetic and therefore can deal with arbitrarily large numbers.

```
def dot(u, v):
    """
    Return the dot product of the vectors u and v
    """
    r = 0
    for (x, y) in zip(u, v):
        r += x * y
    return r

def sqnorm(u):
    """
    Return the square of the norm of the vector u
    """
    return dot(u, u)

def aupbv(a, u, b, v):
    """
    Given two vectors (u, v) and two scalars (a, b),
    return the vector a*u + b*v
    """
    z = []
    for (x, y) in zip(u, v):
        z.append(a * x + b * y)
    return z
```

Computing the norm of a vector requires a bit more care, as the `sqrt` (square root) function in Python only operates on double-precision floating point values. The largest value that can be represented in this way is $2^{1023} \times (1 + (1 - 2^{-52}))$, and this is not enough for our purpose. Therefore, we cannot compute the norm of a vector $\mathbf{u}$ by just computing $\sqrt{\mathbf{u} \cdot \mathbf{u}}$, as both the argument and the result of the square root would be too large. To circumvent this problem, we properly *scale* our vectors to reduce the magnitude of their entries. Most vectors we use have norm about 2^{ℓ}, so we just divide each coefficient by 2^{ℓ} (this divides the norm by 2^{ℓ}). To keep a good precision, we use rational arithmetic, as provided by the `fractions` module of Python's standard library. When computing norms, fractions are automatically converted to the floating-point number that approximate them best.

```
def norm(u):
    return sqrt(dot(u, u))

def scale(s, v):
    """
    Return 1/s * v, where s is a (potentially large)
    scalar and v is a vector.  This yields a vector
    of rationals
    """
    f = fractions.Fraction(1, s)
    return [f * x for x in v]
```

Computing the shortest vectors of the lattice is done by Lagrange's algorithm. Given integer vectors, it only perform integer arithmetic. In order to compare norms, it is enough to compare their square (this avoids computing square roots). In other terms, We check whether $\|\mathbf{u}\|^2 \leq \|\mathbf{v}\|^2$ instead of $\|\mathbf{u}\| \leq \|\mathbf{v}\|$. This function implements the cubic algorithm of figure 2.

```
def lagrange_reduction(u, v):
    """
    Given a basis (u, v) of a 2-dimensional lattice,
    return the 2 shortest vectors.
    """
    if sqnorm(u) < sqnorm(v):
        tmp = u
        u = v
        v = tmp
    while True:
        f = fractions.Fraction(dot(u, v), dot(v, v))
        q = round(f)
        r = aupbv(1, u, -q, v)
        u = v
        v = r
        if sqnorm(u) <= sqnorm(v):
            return (u, v)
```

We now come to the meat of the attack, namely the procedure that enumerate all shorts vectors in the lattice. This requires several norm computations, and vectors are therefore scaled appropriately. The function therefore takes as input a integer "scaling factor" N (a large integer), with the assumption that the norms of $\mathbf{r}$ and $\mathbf{s}$ are close to N.

For instance, the volume of the lattice is close to N^2, and obtaining its value requires a square root computation. We thus compute the scaled volume `sVol` which is equal to the actual volume divided by N^2. Later on, we need the norms of the Gram-Schmidt orthogonalized vectors $\|\mathbf{r}^\star\|$ and $\|\mathbf{s}^\star\|$. Their norms are close to N, so we compute scaled norms `snorm_rstar` $= \|\mathbf{r}^\star\|/N$ and `snorm_sstar` $= \|\mathbf{s}^\star\|/N$.

```
def enumerate(r, s, B, N):
    """Given a lattice L spanned by the two vectors
    (r, s), assumed to be the shortest, return the
    list of all vectors of L with norm less than B.
    It is assumed that r and s have norm about N,
    and that B is close to N.
    """
    short_vectors = []
    # Compute volume
    Vol2 = sqnorm(r) * sqnorm(s) - dot(r, s)**2
    sVol = sqrt(fractions.Fraction(Vol2, N**4))
    # Gram-Schmidt orthogonalization
    mu = fractions.Fraction(dot(r, s), dot(r, r))
    snorm_rstar = norm(scale(N, r))
    snorm_sstar = sVol / snorm_rstar
    # enumeration
    sB = fractions.Fraction(B, N)
    x2_max = floor(sB / snorm_sstar)
    for x2 in range(-x2_max, x2_max + 1):
        x1_max = floor(sB / snorm_rstar - mu*x2)
        x1_min = ceil(-sB / snorm_rstar - mu*x2)
        for x1 in range(x1_min, x1_max + 1):
            w = aupbv(x1, r, x2, s)
            if sqnorm(w) <= B * B:
                short_vectors.append(w)
    return short_vectors
```

Armed with this procedure, we can efficiently enumerate the solutions of the linear program (2). For this, we enumerate all vectors of norm less than $B = \sqrt{3} \cdot 2^\ell$ in the lattice. However, computing B is annoying because the square root returns a floating-point value and 2^ℓ is too large. So we just overshoot by using $B = 2^{\ell+1}$, which is a little larger. Note that we provide 2^ℓ as the scaling factor to the enumeration function. We recover vector of the form given by (3), so we have to remove the $2^{\ell-k}$ multiplicative factor in the last two components.

```
def linear_program(aprime, nprime, l, k):
    """
    Return the set of all (gamma, delta) such that
    |gamma * aprime + delta * nprime| <= 2**l
    |gamma| <= 2**k
    |delta| <= 2**k
    """
    u = [aprime, 2**(l - k), 0]
    v = [nprime, 0, 2**(l - k)]
    r, s = lagrange_reduction(u, v)
    B = 2**(l+1)
    solutions = []
    short_vectors = enumerate(r, s, B, 2**l)
    for (a, g, d) in short_vectors:
        g = g // 2**(l - k)
        d = d // 2**(l - k)
        if abs(a) <= 2**l and abs(g) <= 2**k and \
           abs(d) <= 2**k:
            solutions.append([g, d])
    return solutions
```

And we can finally reconstruct the secrets of the client. In order to avoid returning (n, a) or $(-a, -n)$, we filter out the extraneous outputs. As was shown above, this function provably outputs the secrets.

```
def break_protocol(aprime, nprime, l, k):
    """
    Given the server input (a', n'), produce the
    secrets of the client (a, n).
    """
    Omega = linear_program(aprime, nprime, l, k)
    secrets = []
    for (t, mz) in Omega:
        for (my, x) in Omega:
            if abs(t * x - my * mz) != 1:
                continue
            a = t * aprime + mz * nprime
            n = my * aprime + x * nprime
            if 0 < a < n:
                secrets.append([a, n])
    return secrets
```

With RSA-sized parameters ($\ell = 2048, k = 128$), this functions runs in 25ms on a recent laptop.

7 Conclusion

The modular inversion outsourcing protocol described in [3] is completely insecure. In addition, it does not offer the expected performance gains.

Acknowledgments

The author thanks Damien Vergnaud for suggesting to investigate the security of modular inversion delegation protocols, as well as Léo Ducas and Damien Sthelé for useful discussions about Euclidean lattices.

The author is supported in part by the French *Agence Nationale de la Recherche* under project "GORILLA" (ANR-20-CE39-0002).

References

[1] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 3378. Springer, 2005, pp. 264–282. [Online]. Available: https://doi.org/10.1007/978-3-540-30576-7_15

[2] Q. Su, J. Yu, C. Tian, H. Zhang, and R. Hao, "How to securely outsource the inversion modulo a large composite number," *J. Syst. Softw.*, vol. 129, pp. 26–34, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2017.04.015

[3] C. Tian, J. Yu, H. Zhang, H. Xue, C. Wang, and K. Ren, "Novel secure outsourcing of modular inversion for arbitrary and variable modulus," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 241–253, 2022.

[4] V. Shoup, *A computational introduction to number theory and algebra.* Cambridge University Press, 2006.

[5] D. E. Knuth, "Mathematical analysis of algorithms," in *Information Processing, Proceedings of IFIP Congress 1971, Volume 1 - Foundations and Systems, Ljubljana, Yugoslavia, August 23-28, 1971*, C. V. Freiman, J. E. Griffith, and J. L. Rosenfeld, Eds. North-Holland, 1971, pp. 19–27.

[6] D. Stehlé and P. Zimmermann, "A binary recursive gcd algorithm," in *Algorithmic Number Theory, 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 13-18, 2004, Proceedings*, ser. Lecture Notes in Computer Science, D. A. Buell, Ed., vol. 3076. Springer, 2004, pp. 411–425. [Online]. Available: https://doi.org/10.1007/978-3-540-24847-7_31

[7] R. P. Brent and P. Zimmermann, "Modern computer arithmetic (version 0.5.1)," *CoRR*, vol. abs/1004.4710, 2010. [Online]. Available: http://arxiv.org/abs/1004.4710

[8] P. Q. Nguyen, "Hermite's constant and lattice algorithms," in *The LLL Algorithm - Survey and Applications*, ser. Information Security and Cryptography, P. Q. Nguyen and B. Vallée, Eds. Springer, 2010, pp. 19–69. [Online]. Available: https://doi.org/10.1007/978-3-642-02295-1_2

[9] H. W. L. Jr., "Integer programming with a fixed number of variables," *Math. Oper. Res.*, vol. 8, no. 4, pp. 538–548, 1983. [Online]. Available: https://doi.org/10.1287/moor.8.4.538

[10] K. Aardal and F. Eisenbrand, "The LLL algorithm and integer programming," in *The LLL Algorithm - Survey and Applications*, ser. Information Security and Cryptography, P. Q. Nguyen and B. Vallée, Eds. Springer, 2010, pp. 293–314. [Online]. Available: https://doi.org/10.1007/978-3-642-02295-1_9

[11] J.-L. Lagrange, *Recherches d'arithmétique*. Nouveaux Mémoires de l'Académie royale des Sciences et Belles-Lettres de Berlin, 1775.

[12] D. E. Knuth, *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969. [Online]. Available: https://www.worldcat.org/oclc/310551264

[13] S. D. Galbraith, *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. [Online]. Available: https://www.math.auckland.ac.nz/%7Esgal018/crypto-book/crypto-book.html

[14] J. Hoffstein, J. Pipher, and J. Silverman, *An Introduction to Mathematical Cryptography*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[15] P. Q. Nguyen and B. Vallée, Eds., *The LLL Algorithm - Survey and Applications*, ser. Information Security and Cryptography. Springer, 2010. [Online]. Available: https://doi.org/10.1007/978-3-642-02295-1

[16] P. Q. Nguyen and D. Stehlé, "Low-dimensional lattice basis reduction revisited," in *Algorithmic Number Theory, 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 13-18, 2004, Proceedings*, ser. Lecture Notes in Computer Science, D. A. Buell, Ed., vol. 3076. Springer, 2004, pp. 338–357. [Online]. Available: https://doi.org/10.1007/978-3-540-24847-7_26

Charles Bouillaguet received the Ph.D degree in Cryptology from the Ecole Normale Supérieure, Paris, France, in 2011. He is currently an assistant professor in Sorbonne University in Paris. His research interests include cryptanalysis and high-performance computing.

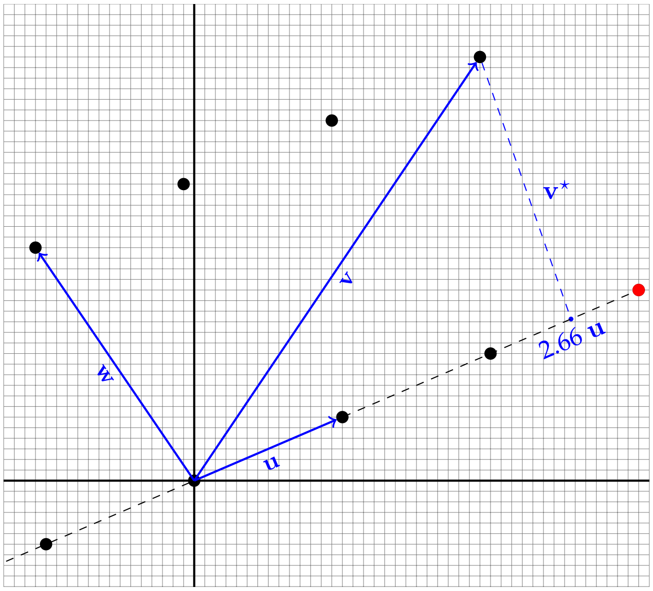

Fig. 5. One step of the Lagrange algorithm.

Appendix A
Lattice Basis Reduction in Dimension Two

In this section, we prove the correctness and complexity of Lagrange's reduction algorithm. This proves theorem 3 above. This material is well-known but is rarely presented in its entirety, and with all the details. It can be found scattered across several textbooks [13], [14], [15] and publications [16]. It is assembled here to make the article as self-contained as possible.

Figure 5 illustrates the intuition behind Lagrange's reduction algorithm: at the beginning of each iteration, except maybe the first one, $\mathbf{u}$ is shorter than $\mathbf{v}$. At each step, the algorithm tries to reduce the norm of $\mathbf{v}$ as much as possible, by adding to it a well-chosen multiple of $\mathbf{u}$. The norm of $\mathbf{v} - q\mathbf{u}$ is minimal when q is chosen such that $q\mathbf{u}$ is as close as possible to $\mathbf{v}$. The optimal value of q is therefore $q = \mathbf{u} \cdot \mathbf{v}\ /\ \|\mathbf{u}\|^2$, which corresponds to the orthogonal projection of $\mathbf{v}$ along $\mathbb{R}\mathbf{u}$. However, q is not necessarily an integer. Therefore, the best we can do is using the closest possible *integer* multiple of $\mathbf{u}$ (shown in red in the figure) by rounding q to the nearest integer.

Lagrange's algorithm returns another basis of the lattice spanned by its input vectors $(\mathbf{u}, \mathbf{v})$. This is easily seen by induction: at each iteration, we have

$$\begin{pmatrix} \mathbf{u}' \\ \mathbf{v}' \end{pmatrix} = \begin{pmatrix} -q & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix},$$

and the transformation matrix is unimodular.

The algorithm shown in fig. 2 does not have quadratic complexity. However, the slightly more involved version shown in fig. 6 does, and turns out to be easier to study. It explicitly maintains the Gram matrix G of the current basis. In other terms, at the beginning of each iteration of the loop we have

$$G_{11} = \|\mathbf{u}\|^2$$
$$G_{12} = \mathbf{u} \cdot \mathbf{v}$$
$$G_{22} = \|\mathbf{u}\|^2$$

```
1:  function LagrangeReduce($\mathbf{u}, \mathbf{v}$)
2:      $G_{11} \leftarrow \|\mathbf{u}\|^2$                    ▷ Setup Gram matrix
3:      $G_{12} \leftarrow \mathbf{u} \cdot \mathbf{v}$
4:      $G_{22} \leftarrow \|\mathbf{v}\|^2$
5:      repeat
6:          $q \leftarrow \lfloor G_{12}/G_{11} \rceil$     ▷ equivalent to $\mathbf{u} \cdot \mathbf{v}\ /\ \|\mathbf{u}\|^2$
7:          $y \leftarrow G_{12} - qG_{11}$
8:          $\mathbf{w} \leftarrow \mathbf{v} - q\mathbf{u}$
9:          $\mathbf{v} \leftarrow \mathbf{u}$
10:         $\mathbf{u} \leftarrow \mathbf{w}$
11:         Swap $G_{11}$ and $G_{22}$             ▷ Update Gram matrix
12:         $G_{11} \leftarrow G_{11} - q(y + G_{12})$
13:         $G_{12} \leftarrow y$
14:     until $G_{11} \geq G_{22}$               ▷ equivalent to $\|\mathbf{u}\| \geq \|\mathbf{v}\|$
15:     return($\mathbf{v}, \mathbf{u}$)
```

Fig. 6. Lagrange's lattice reduction algorithm, quadratic version.

This is true on the first iteration because of the setup on lines 2–4, and it remains true thanks to the maintenance in lines 11– 13 (checking this is left to the reader).

Lemma 1. *The algorithm returns a "Lagrange-reduced" basis* $(\mathbf{r}, \mathbf{s})$, *which means that the following properties are satisfied:*

i) $\|\mathbf{r}\| \leq \|\mathbf{s}\|$,
ii) $|\mathbf{r} \cdot \mathbf{s}| \leq \|\mathbf{r}\|^2/2$.

Proof. That the first condition holds on termination follows from the fact that the loop stops, and the fact that comparing G_{11} and G_{22} is equivalent to comparing $\|\mathbf{u}\|$ and $\|\mathbf{v}\|$. To establish that *ii*) holds as well, we first observe that it is equivalent to saying that $|G_{12}| \leq G_{22}/2$ at the end of the last iteration. We claim that this in fact holds at the end of all iterations. Indeed, the new value installed in G_{12} (by line 13) is $G_{12} - \lfloor G_{12}/G_{22} \rceil G_{22}$. Without the rounding, this would be equal to zero, and because of the rounding this value is less than $G_{22}/2$ in absolute value. □

The following lemma shows that the algorithm optimally shortens $\mathbf{v}$ in each iteration.

Lemma 2. *For any two vectors* $\mathbf{u}$ *and* $\mathbf{v}$, *we have*

$$|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\|^2/2 \iff \forall k \in \mathbb{Z}.\ \|\mathbf{v}\| \leq \|\mathbf{v} + k\mathbf{u}\|.$$

Proof. For the forward direction, we have:

$$\begin{aligned} \|\mathbf{v}\| \leq \|v + k\mathbf{u}\|^2 &= \|\mathbf{v}\|^2 + 2k\mathbf{u} \cdot \mathbf{v} + k^2\|\mathbf{u}\|^2 \\ &\geq \|\mathbf{v}\|^2 - |2k\mathbf{u} \cdot \mathbf{v}| + k^2\|\mathbf{u}\|^2 \\ &\geq \|\mathbf{v}\|^2 + (k^2 - |k|)\|\mathbf{u}\|^2 \end{aligned}$$

The quantity $(k^2 - |k|)$ is always positive for integral values of k; this proves forward implication. For the reverse implication, first set $k = 1$:

$$\begin{aligned} \|\mathbf{v}\| &\leq \|\mathbf{v} + \mathbf{u}\| \\ \|\mathbf{v}\|^2 &\leq \|\mathbf{v}\|^2 + 2\mathbf{u} \cdot \mathbf{v} + \|\mathbf{u}\|^2 \\ -1/2 &\leq \mathbf{u} \cdot \mathbf{v}/\|\mathbf{u}\|^2. \end{aligned}$$

Then set $k = -1$:

$$\begin{aligned} \|\mathbf{v}\|^2 &\leq \|\mathbf{v}\|^2 - 2\mathbf{u} \cdot \mathbf{v} + \|\mathbf{u}\|^2 \\ \mathbf{u} \cdot \mathbf{v}/\|\mathbf{u}\|^2 &\leq 1/2 \end{aligned}$$

Combining these inequalities shows that $|\mathbf{u}\cdot\mathbf{v}|/\|\mathbf{u}\|^2 \leq 1/2$. □

Lemma 3. *If $(\mathbf{r}, \mathbf{s})$ is the basis returned by the algorithm, then $\mathbf{r}$ (resp. $\mathbf{s}$) is the shortest (resp. second shortest) vector of the lattice.*

Proof. This follows from the fact that $(\mathbf{r}, \mathbf{s})$ is a Lagrange-reduced basis. Because $(\mathbf{r}, \mathbf{s})$ is a lattice basis, any non-zero lattice vector $\mathbf{x}$ can be written $\mathbf{x} = a\mathbf{r} + b\mathbf{s}$ (with $a, b \in \mathbb{Z}$). If $b = 0$, then $\|\mathbf{x}\| \geq \|\mathbf{r}\|$. Assume $b \neq 0$, and write the Euclidean division $a = xb + y$ with $0 \leq y < b$. Then

$$\mathbf{x} = y\mathbf{r} + b(\mathbf{s} + x\mathbf{r})$$

Then, by the triangle inequality, by lemma 2, and because $y \geq 0$ and $b - y \geq 1$, we find:

$$\begin{aligned}\|\mathbf{x}\| &\geq |b| \cdot \|\mathbf{s} + x\mathbf{r}\| - y\|\mathbf{r}\| \\ &\geq (|b| - y) \cdot \|\mathbf{s} + x\mathbf{r}\| + y(\|\mathbf{s} + x\mathbf{r}\| - \|\mathbf{r}\|) \\ &\geq (|b| - y) \cdot \|\mathbf{s}\| + y(\|\mathbf{s}\| - \|\mathbf{r}\|) \\ &\geq \|\mathbf{s}\|\end{aligned}$$

Because $\|\mathbf{s}\| \geq \|\mathbf{r}\|$, this shows that $\mathbf{x}$ is longer than $\mathbf{r}$ in all cases. $\mathbf{r}$ is therefore the shortest non-zero vector of the lattice. Then, if $\mathbf{x}$ is not a multiple of $\mathbf{r}$, meaning if $b \neq 0$, then it is longer than $\mathbf{s}$. This shows that $\mathbf{s}$ is the second shortest vector. □

The following lemma shows that a Lagrange-reduced basis has bounded orthogonality defect.

Lemma 4. *If $(\mathbf{r}, \mathbf{s})$ is the basis returned by the algorithm and V is the volume of the lattice, then*

$$\|\mathbf{r}\| \cdot \|\mathbf{s}\| \leq \sqrt{4/3} \cdot V.$$

Proof. Write the Gram-Schmidt orthogonalization of the basis:

$$\mathbf{s} = \mathbf{s}^\star + \mu\mathbf{r} \qquad \text{with} \qquad \mu = \frac{\mathbf{r}\cdot\mathbf{s}}{\|\mathbf{r}\|^2}.$$

We know that $|\mu| \leq 1/2$ (the basis is Lagrange-reduced), and because of this

$$\|\mathbf{s}\|^2 \leq \|\mathbf{s}^\star\|^2 + \frac{1}{4}\|\mathbf{r}\|^2 \leq \|\mathbf{s}^\star\|^2 + \frac{1}{4}\|\mathbf{s}\|^2$$

In other terms, $\|\mathbf{s}\| \leq \sqrt{4/3}\|\mathbf{s}^\star\|$. The lemma then follows from (9). □

Lemma 3 establishes the correctness of the algorithm under the (implicit) assumption that the algorithm terminates, a fact that we now establish.

Lemma 5. *$|q| \geq 2$ in all iterations of the loop, except potentially the first and the last one.*

Proof. Because of the loop exit condition, we know that $\|\mathbf{u}\| \leq \|\mathbf{v}\|$ at the beginning of all iterations except potentially the first one. Consider the value of q computed in line 6.

If $q = 0$, then the iteration just swaps $\mathbf{u}$ and $\mathbf{v}$. After the first iteration, this triggers the loop exit condition, so it can only happen in the last iteration.

If $|q| = 1$, we distinguish two cases:

1) Either $\|\mathbf{v} - q\mathbf{u}\| \geq \|\mathbf{u}\|$. This also triggers the loop exit condition, and thus can only happen in the last iteration.
2) Or $\|\mathbf{v} - q\mathbf{u}\| < \|\mathbf{u}\|$. We claim that this can only happen during the first iteration. Indeed, if this is not the first iteration, the reasoning in the proof of lemma 1 shows that $|\mathbf{u}\cdot\mathbf{v}| \leq \|\mathbf{u}\|^2/2$ holds at the end of the previous iteration, and thus at the beginning of the current one. But then lemma 2 tells us that $\|\mathbf{v} - q\mathbf{u}\| \geq \|\mathbf{v}\|$. Because $q = \pm 1$, multiplying by q yields

$$\|\mathbf{u} - q\mathbf{v}\| \geq \|\mathbf{v}\| \geq \|\mathbf{u}\|.$$

□

Lemma 6. *Each iteration (except possibly the first and last one) decreases the norm of $\mathbf{v}$ by a multiplicative factor at least $\sqrt{3}$.*

Proof. In these iterations, we know that $|q| \geq 2$ thanks to lemma 5. The statement of the lemma is equivalent to $\|\mathbf{v}\|^2 > 3\|\mathbf{v} - q\mathbf{u}\|^2$.

Write the Gram-Schmidt orthogonalization: $\mathbf{v} = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|^2}\mathbf{u} + \mathbf{v}^\star$, where $\mathbf{v}^\star$ and $\mathbf{u}$ are orthogonal. Because $|q| \geq 2$, we necessarily have $|\mathbf{u}\cdot\mathbf{v}| \,/\, \mathbf{u}\|^2 \geq 3/2$. It follows that

$$\|\mathbf{v}\|^2 \geq (3/2)^2\|\mathbf{u}\|^2 + \|\mathbf{v}^\star\|.$$

Then, because

$$\mathbf{v} - q\mathbf{u} = \left(\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|^2} - \left\lfloor\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|^2}\right\rceil\right)\mathbf{u} + \mathbf{v}^\star$$

We find that

$$\|\mathbf{v} - q\mathbf{u}\|^2 \leq \frac{1}{4}\|\mathbf{u}\|^2 + \|\mathbf{v}^\star\|^2$$

It follows that

$$\|\mathbf{v}\|^2 \geq 2\|\mathbf{u}\|^2 + \|\mathbf{v} - q\mathbf{u}\|^2.$$

If this is not the last iteration, then $\|\mathbf{v} - q\mathbf{u}\| \geq \|\mathbf{u}\|$, and the proof is complete. □

Lemma 6 shows that in each iteration except possibly the first and last ones, the products of the norms of $\mathbf{u}$ and $\mathbf{v}$ decreases by a multiplicative factor at least $\sqrt{3}$. Let τ denote the total number of iterations and n denote the total bit size of both $\mathbf{u}$ and $\mathbf{v}$. It follows from lemma 6 that $\tau = \mathcal{O}(n)$. This is sufficient to prove that the complexity of the simple version shown in figure 2 is $\mathcal{O}(n^3)$, because all operations have integer operands of less than n bits and each iteration costs $\mathcal{O}(n^2)$.

To conclude the proof of theorem 3, it remains to show that the total number of operations of the more efficient version shown in figure 6 is quadratic in the bit size of the input basis.

Computing G_{11}, G_{12} and G_{22} in the setup phase requires a constant number of arithmetic operations on n-bit integers. The setup phase thus requires less than $\mathcal{O}(n^2)$ operations.

Lemma 7. *The number of elementary operations performed by a single iteration of the loop is upper-bounded by*

$$\lambda(4\mu + 2 + 10n)(\mu + 2 + \log\|\mathbf{v}\| - \log\|\mathbf{u}\|)$$

($\mathbf{u}$ and $\mathbf{v}$ are the values at the beginning of the iteration).

Proof. During the whole execution of the algorithm, the entries of $\mathbf{u}$ and $\mathbf{v}$ require n bits or less, but never more: they are bounded by the norms, and the norms only decrease.

G_{11} and G_{12} require at most $2n$ bits, and the Cauchy-Schwartz inequality states that

$$|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \cdot \|\mathbf{v}\|,$$

which implies that G_{12} occupies at most $2n$ bits. In addition, this shows that $|q| \leq 1/2 + \|\mathbf{v}\| \;/\; \|\mathbf{u}\|$, where the $1/2$ term comes from the rounding. It follows that

$$\log |q| \leq 1 + \log \|\mathbf{v}\| - \log \|\mathbf{u}\|.$$

Computing (q, y) requires less than $\lambda(\mu + \log |q|)(\mu + 2n)$ operations. The two multiplications needed to compute $\mathbf{w}$ require less than $2\lambda(\mu + \log |q|)(\mu + n)$ operations. Note that the entries of $q\mathbf{u}$ are smaller than $2\|\mathbf{v}\|$, and the two final subtractions require less than $2\lambda(\mu + 1 + 2n)$ operations.

The other non-trivial part of the algorithm is the update of G_{11} on line 12. Both y and G_{12} require less than $2n$ bits, so adding them requires less than $\lambda(\mu + 4n)$ operations. The multiplication with q requires less than $\lambda(\mu + \log |q|)(\mu + 1 + 2n)$ operations. Finally, the last subtraction operates on numbers smaller than n-bits, so it costs $\lambda(\mu + 2n)$ operations.

Adding everything results in the bound given in the lemma. □

Let $\mathbf{u}_i, \mathbf{v}_i$ denote the value of $\mathbf{u}$ and $\mathbf{v}$ at the beginning of the i-th iteration. Then $\mathbf{v}_{i+1} = \mathbf{u}_i$. The total cost of all iterations of the algorithm is upper-bounded by

$$\lambda(4\mu + 2 + 10n) \sum_{i=1}^{\tau} \mu + 2 + \log \|\mathbf{v}_i\| - \log \|\mathbf{v}_{i+1}\|$$

This telescoping sum simplifies. We obtain an upper bound on the total number of operations of all iterations

$$\lambda(4\mu + 2 + 10n)((\tau - 1)(\mu + 2) + n)$$

Which is quadratic in n, because $\tau = \mathcal{O}(n)$. Theorem 3 is then finally proved.