Integer factoring and modular square roots

Emil Jeřábek[*]

Institute of Mathematics of the Academy of Sciences
Žitná 25, 115 67 Praha 1, Czech Republic, email: jerabek@math.cas.cz

July 22, 2012

Abstract

Buresh-Oppenheim proved that the NP search problem to find nontrivial factors of integers of a special form belongs to Papadimitriou's class PPA, and is probabilistically reducible to a problem in PPP. In this paper, we use ideas from bounded arithmetic to extend these results to arbitrary integers. We show that general integer factoring is reducible in randomized polynomial time to a PPA problem and to the problem WeakPigeon $\in$ PPP. Both reductions can be derandomized under the assumption of the generalized Riemann hypothesis. We also show (unconditionally) that PPA contains some related problems, such as square root computation modulo n, and finding quadratic nonresidues modulo n.

1 Introduction

Integer factoring is one of the best-known problems in complexity theory which is in NP, but is not known to be polynomial-time computable. In particular, the assumed hardness of factoring has various applications in cryptography. Papadimitriou [10] introduced several classes of search problems based on parity arguments and related combinatorial principles. He showed that many natural search problems from diverse areas of mathematics belong to one of these classes, and he posed as an open problem whether the same holds for integer factoring.

The first step to answer Papadimitriou's question was undertaken by Buresh-Oppenheim [5]. He proved that factoring of "good" integers (odd integers n such that -1 is not a quadratic residue modulo n) such that $n \equiv 1 \ (4)$ belongs to the search class PPA, and factoring of good integers is probabilistically poly-time reducible to a PPP problem. (Note that an odd integer is good iff it has a prime divisor $p \equiv -1 \ (4)$.)

The purpose of this paper is to exhibit similar reductions exist for factoring of arbitrary integers. We show that factoring is probabilistically poly-time reducible to a PPA problem,

[*]Supported by grant IAA100190902 of GA AV ČR, Center of Excellence CE-ITI under the grant P202/12/G061 of GA ČR, and RVO: 67985840. Part of the research was done while visiting the Isaac Newton Institute in Cambridge.

as well as to WeakPigeon (which is a PPP problem). We isolate a convenient intermediate problem, which we call FacRoot: given integers n and a such that the Jacobi symbol $(a|n) = 1$, find either a proper divisor of n, or a square root of a modulo n. It is not hard to show that factoring is probabilistically poly-time reducible to FacRoot.

The main technical ingredient of our work is to demonstrate that FacRoot $\in$ PPA. The high-level idea of the proof comes from bounded arithmetic. Jeřábek [8] introduced an arithmetical theory $S^1_2 + \mathit{Count}_2(PV)$ related to PPA, and established that this theory can prove the quadratic reciprocity theorem and other properties of the Jacobi symbol, which together imply the soundness of the usual poly-time algorithm for the Jacobi symbol. In particular, $S^1_2 + \mathit{Count}_2(PV)$ proves the totality of FacRoot, and then an application of a garden-variety witnessing theorem yields FacRoot $\in$ PPA. However, since this paper is intended for a general computational complexity audience, we include a self-contained direct proof of this result, we do not assume any prior knowledge (or posterior, for that matter) of bounded arithmetic on the part of the reader.

All probabilistic reductions in this paper can be derandomized if we assume the generalized Riemann hypothesis (*GRH*). In particular, *GRH* implies that factoring is in PPA $\cap$ PPP (and moreover, it is poly-time reducible to WeakPigeon). We also show unconditionally that several problems concerning quadratic residues have deterministic Turing reductions to FacRoot, and as such are in PPA: for one, given n and a, we can find either a square root of a modulo n, or a suitable witness that a is a quadratic nonresidue. For another, given an odd n which is not a perfect square, we can find an a such that $(a|n) = -1$ (in particular, a is a quadratic nonresidue modulo n).

The paper is organized as follows. In Section 2, we review basic concepts used in the paper to fix the notation. Section 3 presents our main results, except for the somewhat complex proof of FacRoot $\in$ PPA, which is given separately in Section 4. Some concluding remarks follow in Section 5.

2 Preliminaries

An NP *search problem* is given by a poly-time computable relation $R(x, y)$ such that $R(x, y)$ implies $\|y\| \le \|x\|^c$ for some constant c, the problem is to find a y satisfying $R(x, y)$ given x. (We use $\|x\|$ to denote the length of x; most of our algorithms work with integers, and we reserve $|x|$ for the absolute value of x. We also warn the reader that we will often call our binary integers n, we will not use the convention that n implicitly denotes the length of the input.) For brevity, we may use R to denote the search problem itself. A search problem R is *total* if for every x there exists a y such that $R(x, y)$. Unless indicated otherwise, all search problems below will be assumed to be total NP search problems.

We will often specify NP search problems in the form "given an x such that $P(x)$, find a y satisfying $R(x, y)$", where P is a poly-time condition. In order to make it formally a total search problem, this formulation will be understood to denote the problem associated with the relation $(\neg P(x) \wedge y = 0) \vee (P(x) \wedge R(x, y))$.

A search problem R is *many-one reducible* to a search problem S, written as $R \le_m S$, if there are poly-time functions f, g such that $S(f(x), y)$ implies $R(x, g(x, y))$. R is *Turing-reducible* to S, written as $R \leq_T S$, if there exists a poly-time oracle Turing machine M (where the oracle returns strings rather than yes/no answers) such that on input x, M computes a y solving $R(x, y)$ whenever all answers of the oracle are correct solutions of S. The class of all search problems R such that $R \leq_T S$ will be denoted FP^S. If C is a class of search problems, we write $R \leq_m C$ if $R \leq_m S$ for some $S \in C$, and similarly for $R \leq_T C$, FP^C, as well as other reduction notions mentioned below.

Let a circuit $C\colon \mathbf{2}^n \to \mathbf{2}^n$ (here, $\mathbf{2} = \{0, 1\}$) encode an undirected graph $G = \langle V, E\rangle$, where $V = \mathbf{2}^n \smallsetminus \{0^n\}$, and $\{u, v\} \in E$ iff $u, v \in V$, $u \neq v$, $C(u) = v$, and $C(v) = u$. Notice that G is a partial matching. Lonely is the following search problem: given C, find $u \in V$ unmatched by G. The class PPA (for "polynomial parity argument") consists of all search problems many-one reducible to Lonely. (This is not Papadimitriou's definition of PPA, it comes from [3], where it is shown to be equivalent to the original one.) By abuse of notation, we will also use Lonely to denote the following variant of the problem. Let $f(a, x)$, $g(a)$ be poly-time functions such that for every a, $g(a)$ is an odd natural number, and the function $f_a(x) := f(a, x)$ is an involution (i.e., $f_a(f_a(x)) = x$) on the integer interval $[0, g(a))$. Then the problem is, given a to find an $x < g(a)$ which is a fixpoint of f_a (i.e., $f_a(x) = x$). We will often use the fact that PPA is closed under Turing reductions:

Theorem 2.1 (Buss and Johnson [7]) $\mathrm{FP}^{\mathrm{PPA}} = \mathrm{PPA}$. □

The class PPP (for "polynomial pigeonhole principle") consists of problems many-one reducible to Pigeon, which is the following problem: given a circuit $C\colon \mathbf{2}^n \to \mathbf{2}^n$, find either a pair $u \neq v$ such that $C(u) = C(v)$, or a u such that $C(u) = 0^n$. If $p(n)$ is any polynomial such that $p(n) > n$ for every n, let $\textsc{WeakPigeon}_{2^n}^{2^{p(n)}}$ denote the following problem: given a circuit $C\colon \mathbf{2}^{p(n)} \to \mathbf{2}^n$, find $u \neq v$ such that $C(u) = C(v)$. We define $\textsc{WeakPigeon} := \textsc{WeakPigeon}_{2^n}^{2^{n+1}}$; the choice of $n + 1$ here does not matter:

Lemma 2.2 *For any polynomial p as above,* $\textsc{WeakPigeon} \equiv_m \textsc{WeakPigeon}_{2^n}^{2^{p(n)}}$.

Proof: Given a circuit $C(\vec{x}, u)\colon \mathbf{2}^n \times \mathbf{2} \to \mathbf{2}^n$, we put $m = p(n) - n$, and we construct a circuit $D\colon \mathbf{2}^n \times \mathbf{2}^m \to \mathbf{2}^n$ by $D(\vec{x}, u_0, \dots, u_{m-1}) = C(\cdots (C(C(\vec{x}), u_0), u_1) \dots, u_{m-1})$. Given $\langle \vec{x}, \vec{u}\rangle \neq \langle \vec{x}', \vec{u}'\rangle$ such that $D(\vec{x}, \vec{u}) = D(\vec{x}', \vec{u}')$, we find the largest $i < m$ such that $\langle \vec{y}, u_i\rangle \neq \langle \vec{y}', u_i'\rangle$, where $\vec{y}^{(\prime)} = C(\cdots (C(C(\vec{x}^{(\prime)}), u_0^{(\prime)}), u_1^{(\prime)}) \dots, u_{i-1}^{(\prime)})$. Then $C(\vec{y}, u_i) = C(\vec{y}', u_i')$. □

The class of all search problems many-one reducible to WeakPigeon does not seem to have an established name in the literature, although it clearly deserves one. In analogy with PPP, we can call it PWPP for "polynomial weak pigeonhole principle". Note that neither PPP nor PWPP is known to be closed under Turing reductions. The proof of Lemma 2.2 also implies that problems of the following kind belong to PWPP; we will denote them all as WeakPigeon by abuse of notation. Let $\varepsilon > 0$ be a constant, and f, g poly-time function such that for any a, $g(a) > 0$, and $f_a(x) := f(a, x)$ maps the interval $\bigl[0, \lceil (1 + \varepsilon) g(a)\rceil\bigr)$ into $\bigl[0, g(a)\bigr)$. Then the problem is, given a, to find $u < v < \lceil (1+\varepsilon) g(a)\rceil$ such that $f_a(u) = f_a(v)$.

Apart from $\le_m$ and $\le_T$, we will also need randomized reductions. We will use several different versions to be able to state our results precisely; the definitions below are not standard, but we believe they are quite natural.

For any constant $0 < \varepsilon < 1$, we say that R is *probabilistically many-one reducible to S with error ε*, written as $R \le_m^{\mathrm{RP},\varepsilon} S$, if there is a polynomial p and poly-time functions $f(x, r)$ and $g(x, r, y)$ such that for every x,

$$\Pr_{\|r\|=p(\|x\|)}[\forall y\,[S(f(x,r),y) \Rightarrow R(x, g(x,r,y))]] \ge 1 - \varepsilon.$$

We say that R is *probabilistically many-one reducible to S with controlled error*, written as $R \le_m^{\mathrm{RP}} S$, if there is a polynomial p and poly-time functions $f(x, 1^k, r)$ and $g(x, 1^k, r, y)$ such that for every x and k,

$$\Pr_{\|r\|=p(\|x\|,k)}[\forall y\,[S(f(x,1^k,r),y) \Rightarrow R(x, g(x,1^k,r,y))]] \ge 1 - 2^{-k}.$$

R is *probabilistically Turing-reducible to S*, written as $R \le_T^{\mathrm{RP}} S$, if there exists a polynomial p and a poly-time oracle Turing machine M such that

$$\Pr_{\|r\|=p(\|x\|)}[\text{every sound run of } M(x,r) \text{ solves } R(x,y)] \ge 1/2,$$

where a run is sound if all oracle answers are correct solutions of S. Note that the constant $1/2$ here is arbitrary, as we can decrease the error from any constant $\varepsilon > 0$ to any other constant (or to controlled error as above) in the usual way: we can check solutions of R, hence we can run the machine several times with independent choices of r, and return the first correct solution to the search problem. We denote by TFRP^S the class of all R such that $R \le_T^{\mathrm{RP}} S$. We observe that we can split a randomized Turing reduction as a randomized many-one reduction followed by a deterministic Turing reduction; this is particularly useful when S is from a Turing-closed class such as PPA.

Lemma 2.3 *$\mathrm{TFRP}^S \le_m^{\mathrm{RP}} \mathrm{FP}^S$.*

Proof: Let T be the following search problem: given x and r, find a sound run of $M^S(x,r)$. It is easy to see that T is a total NP search problem, and $R \le_m^{\mathrm{RP}} T \le_T S$. □

Lemma 2.4 *$\mathrm{TFRP}^{\mathrm{TFRP}^S} = \mathrm{TFRP}^S$.*

Proof: In view of Lemma 2.3 and the obvious transitivity of $\le_m^{\mathrm{RP}}$, it suffices to show that TFRP^S is closed under deterministic Turing reductions. Let thus $T \in \mathrm{TFRP}^S$, and M^T be a poly-time oracle machine solving $R(x, y)$. Since answers of the oracle have polynomial length, the total number of sound runs of M on input x is bounded by $2^{\|x\|^c}$ for some constant c. Using the above-mentioned amplification of success rate, we can find a randomized poly-time machine N^S solving T with error $2^{-\|x\|^{c+1}}$. If we then use N to answer M's oracle queries while reusing the same pool of random bits for every call, all but a fraction of $2^{\|x\|^c}2^{-\|x\|^{c+1}} \ll 1$ of the random choices will be good for every possible run of the combined machine. □

A many-one reduction of R to S is supposed to construct a valid instance of S from whose solution it can recover a solution to the original problem. In the case of $\leq_m^{\mathrm{RP}}$, the reduction algorithm succeeds in doing this only with some bounded probability. It will be also useful to consider stronger notions of reduction where we can check before consulting the oracle whether the particular choice of random bits leads to the desired result. The reduction function may abandon the computation with some bounded probability, but if it does not, then any valid solution of S gives a solution of R. Alternatively, we could repeat the computation until we find a "good" instance of S, and only then pass the query to the oracle; in this way, the reduction always succeeds, but only its expected running time is polynomial.

Formally, R is *probabilistically zero-error many-one reducible to* S, written as $R \leq_m^{\mathrm{ZPP}} S$, if there is a polynomial p, poly-time functions $f(x, r)$ and $g(x, r, y)$, and a poly-time predicate $h(x, r)$, such that

(i) $\Pr_{\|r\|=p(\|x\|)}[h(x, r)] \geq 1/2$,

(ii) if $h(x, r)$ and $S(f(x, r), y)$, then $R(x, g(x, r, y))$.

Similarly, R is *probabilistically zero-error Turing-reducible to* S, written as $R \leq_T^{\mathrm{ZPP}} S$, if there is a polynomial p, a poly-time predicate $h(x, r)$, and a poly-time oracle Turing machine M, such that (i), and if $h(x, r)$, then every sound run of $M^S(x, r)$ solves $R(x, y)$. Again, the constant $1/2$ is arbitrary, we can amplify the success rate from any constant $\varepsilon > 0$ to $1 - 2^{-k}$ (even for many-one reductions). Let TFZPP^S denote the class of all problems R such that $R \leq_T^{\mathrm{ZPP}} S$. Note that if there is no oracle, TFZPP = TFRP.

Factoring is the following search problem: given a composite integer n, find a nontrivial divisor of n. We define FullFac to be the following problem: given an integer $n > 0$, find a sequence $\langle p_i : i < k \rangle$ of primes such that $n = \prod_{i<k} p_i$ (here and below, the empty product is defined to be 1). Note that Factoring and FullFac are total NP search problems as primality testing is poly-time (Agrawal, Kayal, and Saxena [1]). Clearly, Factoring $\leq_m$ FullFac $\leq_T$ Factoring.

We will denote the divisibility relation by $d \mid n$, modular congruences by $a \equiv b \ (n)$, and greatest common divisors by (a, b). An integer a is a *quadratic residue* modulo n if $a \equiv b^2$ (n) for some b. The *Legendre symbol* is defined for any integer a and an odd prime p by

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & p \mid a, \\ 1 & p \nmid a \text{ and } a \text{ is a quadratic residue mod } p, \\ -1 & p \nmid a \text{ and } a \text{ is a quadratic nonresidue mod } p. \end{cases}$$

More generally, the *Jacobi symbol* is defined for any odd $n > 0$ by

$$\left(\frac{a}{n}\right) = \prod_{i<k} \left(\frac{a}{p_i}\right),$$

where $n = \prod_{i<k} p_i$ is the prime factorization of n. We will also write $(a|n)$ instead of $\left(\frac{a}{n}\right)$ for typographical convenience. A *Dirichlet character* of modulus n is a group homomorphism $\chi\colon (\mathbb{Z}/n\mathbb{Z})^* \to \mathbb{C}^*$. A character is *principal* if it only assumes the value 1, and *quadratic* if

```
$r \leftarrow 1$
**while** $a \neq 0$ **do**:
    **if** $a < 0$ **then**:
        $a \leftarrow -a$
        $r \leftarrow -r$ **if** $n \equiv -1 \ (4)$
    **while** $a$ is even **do**:
        $a \leftarrow a/2$
        $r \leftarrow -r$ **if** $n \equiv \pm 3 \ (8)$
    swap $a$ and $n$
    $r \leftarrow -r$ **if** $a \equiv n \equiv -1 \ (4)$
    reduce $a$ modulo $n$ so that $|a| < n/2$
**if** $n > 1$ **then** output 0 **else** output $r$
```

Figure 1: An algorithm for the Jacobi symbol $(a|n)$

it takes values in $\{1, -1\}$. Characters can be lifted to mappings $\mathbb{Z} \to \mathbb{C}$ by putting $\chi(a) = 0$ when $(a, n) \neq 1$. Note that for any odd positive n, $(\cdot|n)$ is a quadratic character of modulus n (in particular, $(a|n)(b|n) = (ab|n)$), and it is principal iff n is a perfect square. The *quadratic reciprocity theorem* states that for any coprime odd $n, m > 0$,

$$\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) = \begin{cases} -1 & \text{if } n \equiv m \equiv -1 \quad (4) \\ 1 & \text{otherwise.} \end{cases}$$

Together with the supplementary laws

$$\left(\frac{-1}{n}\right) = \begin{cases} 1 & n \equiv 1 \quad (4) \\ -1 & n \equiv -1 \quad (4) \end{cases} \qquad \left(\frac{2}{n}\right) = \begin{cases} 1 & n \equiv \pm 1 \quad (8) \\ -1 & n \equiv \pm 3 \quad (8) \end{cases}$$

it implies that the Jacobi symbol is poly-time computable (see Figure 1).

The *generalized Riemann hypothesis*[1] (*GRH*) states that for every Dirichlet character χ, all zeros of its associated L-function $L(\chi, s)$ with $0 < \mathrm{Re}(s) < 1$ have in fact $\mathrm{Re}(s) = 1/2$. Let GRH_q denote the the special case of *GRH* for quadratic characters χ. We will use the following result of Bach [2].

Theorem 2.5 *Assume GRH_q. If χ is a nonprincipal quadratic Dirichlet character with modulus n, there exists* $0 < a < 2(\ln n)^2$ *such that* $\chi(a) \neq 1$. □

[1] Also called the extended Riemann hypothesis (*ERH*). The nomenclature of various extensions of *RH* varies wildly in the literature. We chose to denote the *RH* for Dirichlet L-functions by *GRH* as this name seems to be more specific, whereas *ERH* is often used for other generalizations of *RH*, such as the *RH* for Dedekind ζ-functions, or L-functions of Hecke characters.

3 Search complexity of factoring

In this section, we are going to describe our main result (Theorem 3.6) on the relationship of factoring to the classes PPA and PPP (PWPP). Rather than working directly with Factoring, it will be convenient to consider other related problems.

Definition 3.1 Let FacRoot denote the following problem: given an odd integer $n > 0$ and an integer a such that $(a|n) = 1$, find either a nontrivial divisor of n, or a square root of a modulo n.

We also give names to some special cases of FacRoot. FacRootMul denotes the problem, given odd $n > 0$ and integers a and b, to find a nontrivial divisor of n or a square root of one of a, b, or ab modulo n.

WeakFacRoot is the following problem: given an odd $n > 0$ and a, b such that $(a|n) = 1$ and $(b|n) = -1$, find a nontrivial divisor of n, or a square root of a modulo n.

We start with basic dependencies between these problems.

Lemma 3.2

(i) WeakFacRoot $\leq_m$ FacRootMul $\leq_m$ FacRoot*,*

(ii) WeakFacRoot $\leq_m$ Factoring*.*

Proof: (i): WeakFacRoot is a special case of FacRootMul, since $(a|n) = 1$ and $(b|n) = -1$ imply that neither b nor ab is a quadratic residue modulo n. Given an instance of FacRootMul, the multiplicativity of the Jacobi symbol implies that $(x|n) = 1$ for some $x \in \{a, b, ab\}$. We can choose such an x as the Jacobi symbol is poly-time computable, and then we pass it to FacRoot.

(ii): If n is prime, we can compute a square root of a modulo n in polynomial time using the Shanks–Tonelli algorithm. This algorithm is deterministic if we provide it with a quadratic nonresidue, which we can: b. If n is composite, we pass it to Factoring. □

Lemma 3.3

(i) FacRoot $\leq_m^{\mathrm{ZPP}}$ WeakFacRoot*,*

(ii) Factoring $\leq_m^{\mathrm{RP},3/4}$ FacRoot*,*

(iii) FullFac $\leq_T^{\mathrm{RP}}$ WeakFacRoot*.*

Proof: (i): If n is a perfect square, we can return $\sqrt{n}$ as its nontrivial divisor (unless it is 1, in which case we can return 0 as the square root of a). Otherwise $(\cdot|n)$ is a nontrivial quadratic character, hence with probability at least $1/2$, a randomly chosen $0 < b < n$ either shares a factor with n (in which case we can return (n, b) as a nontrivial divisor) or satisfies $(b|n) = -1$, and we can pass it to WeakFacRoot.

(ii): If n is even or a perfect power, we can factor it directly, hence we may assume n is odd and it has $k \geq 2$ distinct prime divisors. We consider the following reduction. We choose a random $0 < a < n$. If $(a, n) \neq 1$, we can return it as a nontrivial divisor of n, otherwise we pass n, a to a FacRoot oracle.

Since $(\cdot|n)$ is a nontrivial quadratic character, we have $(a|n) = 1$ for a half of all residues from $(\mathbb{Z}/n\mathbb{Z})^*$. On the other hand, if $n = \prod_{i<k} p_i^{e_i}$, where the p_i are distinct primes, then a coprime to n is a quadratic residue modulo n iff $(a|p_i) = 1$ for every $i < k$. Using the Chinese remainder theorem, a fraction 2^{-k} of $(\mathbb{Z}/n\mathbb{Z})^*$ are quadratic residues. Thus, with probability at least $1/2 - 2^{-k} \geq 1/4$, the chosen a either shares a factor with n, or it satisfies $(a|n) = 1$ while not being a quadratic residue, hence the FacRoot oracle must give us a factor of n.

(iii) follows from (i), (ii), and Lemma 2.4. □

We remark that there is another well-known randomized reduction of factoring to square root computation modulo n by Rabin [11]. Adapted to our situation, the basic idea is that we choose a random $1 < a < n$, and if coprime to n, we pass n, a^2 to the FacRoot oracle. If the oracle were implemented as a (deterministic or randomized) algorithm, we would have a $1/2$ chance that the root b of a^2 returned by the oracle satisfies $a \not\equiv \pm b \ (n)$, allowing us to factor n. However, this does not work in general. According to the definition of search problem reduction, the reduction function must be able to cope with *any* valid answer to the oracle query, there is no implied guarantee that the oracle answers are computed independently of the environment. In particular, we may assume the oracle is devious enough to always return the root $b = a$ we already know.

What remains now is to show that FacRoot or some of its variants belongs to PPA and PWPP.

Theorem 3.4 FacRoot $\in$ PPA.

We will prove Theorem 3.4 in the next section, as the argument is a bit involved.

For the pigeonhole principle, we have the following reduction, whose idea comes from the proof of the multiplicativity of the Legendre symbol in $I\Delta_0 + \mathit{WPHP}(\Delta_0)$ by Berarducci and Intrigila [4].

Theorem 3.5 FacRootMul $\in$ PWPP.

Proof: Assume we are given an odd $n > 1$, and integers a, b. If a or b shares a factor with n, we can return (n, a) or (n, b), resp., as a nontrivial divisor of n, we thus assume both are coprime to n. Consider the following poly-time function $f\colon \{0, 1, 2\} \times [1, (n-1)/2] \to [1, n-1]$:

$$f(i, x) = \begin{cases} a_i x^2 \bmod n & \text{if } (n, x) = 1, \\ x & \text{otherwise,} \end{cases}$$

where $a_0 = 1$, $a_1 = a$, $a_2 = b$. Since the domain of f is $3/2$ times larger than its range, we can use WeakPigeon to find a collision $f(i, x) = f(j, y)$, $\langle i, x\rangle \neq \langle j, y\rangle$. We may assume $(n, x) = (n, y) = 1$, as otherwise we can factor n. If $i = j$, then $x^2 \equiv y^2 \ (n)$, but $x \not\equiv \pm y \ (n)$, hence $(n, x - y)$ is a nontrivial divisor of n. If $i < j$, then $a_j a_i^{-1} \equiv (xy^{-1})^2 \ (n)$ (where the inverses are also modulo n), hence xy^{-1} is a square root of a, b, or ba^{-1} modulo n. In the last case, axy^{-1} is a square root of ab. □

We obtain the main result of this paper by putting everything together:

Theorem 3.6

(i) $\textsc{Factoring}, \textsc{FullFac} \le_m^{\mathrm{RP}} \mathrm{PPA}$*,*

(ii) $\textsc{Factoring} \le_m^{\mathrm{RP},3/4+o(1)} \mathrm{PWPP} \subseteq \mathrm{PPP}$ *and* $\textsc{FullFac} \le_m^{\mathrm{RP}} \mathrm{FP}^{\mathrm{PWPP}} \subseteq \mathrm{FP}^{\mathrm{PPP}}$.

Proof: (i): We have $\textsc{FullFac} \in \mathrm{TFRP}^{\mathrm{PPA}}$ by Theorem 3.4 and Lemmas 3.2 and 3.3 (iii), hence $\textsc{FullFac} \le_m^{\mathrm{RP}} \mathrm{FP}^{\mathrm{PPA}} = \mathrm{PPA}$ by Lemma 2.3 and Theorem 2.1.

(ii): $\textsc{FacRoot} \le_m^{\mathrm{RP}} \textsc{WeakFacRoot}$ by Lemma 3.3 (i) and amplification of the success rate of $\le_m^{\mathrm{ZPP}}$, hence $\textsc{Factoring} \le_m^{\mathrm{RP},3/4+o(1)}$ PWPP by Theorem 3.5 and Lemmas 3.2 and 3.3. Similarly, $\textsc{FullFac} \in \mathrm{TFRP}^{\mathrm{PWPP}}$ by Theorem 3.5 and Lemma 3.3 (iii), hence $\textsc{FullFac} \le_m^{\mathrm{RP}} \mathrm{FP}^{\mathrm{PWPP}}$ by Lemma 2.3. □

It would be desirable to derandomize the results in Theorem 3.6. We are only able to do it under an extra assumption.

Theorem 3.7 *Assume GRH_q.*

(i) $\textsc{Factoring} \equiv_m \textsc{FacRoot} \equiv_m \textsc{WeakFacRoot} \equiv_m \textsc{FacRootMul}$*,*

(ii) $\textsc{Factoring}, \textsc{FullFac} \in \mathrm{PPA}$*,*

(iii) $\textsc{Factoring} \in \mathrm{PWPP}$*,* $\textsc{FullFac} \in \mathrm{FP}^{\mathrm{PWPP}}$*.*

Proof: It suffices to derandomize the reductions in Lemma 3.3 (i,ii). For $\textsc{FacRoot} \le_m \textsc{WeakFacRoot}$, note that Theorem 2.5 guarantees that we can find a suitable $b < 2(\ln n)^2 = O(\|n\|^2)$.

For $\textsc{Factoring} \le_m \textsc{FacRoot}$, it suffices to show that for any odd n which is not a prime power, there exists an $0 < a < (\ln n)^{O(1)}$ such that either $(a, n) > 1$, or $(a|n) = 1$ and a is a quadratic nonresidue modulo n; the latter means that $(a|p) = -1$ for some prime $p \mid n$.

We can assume that $(a, n) = 1$ for every $0 < a < 2(\ln n)^2$, otherwise we are done. Let p be a prime divisor of n such that, if possible, the exponent of p in the prime factorization of n is even, so that n/p is not a perfect square. Then $(\cdot|n/p)$ is a nonprincipal quadratic character, and there is $0 < u < 2(\ln(n/p))^2$ such that $(u|n/p) = -1$ by Theorem 2.5. This implies $(u|n) = -(u|p)$. If $(u|n) = 1$, we can take $a = u$. Otherwise, we have $(u|n) = -1$ and $(u|p) = 1$. Since $(\cdot|p)$ is also a nonprincipal quadratic character, there is $0 < v < 2(\ln p)^2$ such that $(v|p) = -1$. If $(v|n) = 1$, we can take $a = v$, otherwise we take $a = uv$. Either way, $a < 4(\ln p)^2(\ln(n/p))^2 < \frac{1}{4}(\ln n)^4$. □

It is not clear whether one can derandomize Theorem 3.6 unconditionally. While no deterministic polynomial-time algorithm to find quadratic nonresidues is known without GRH, in PPA we can do better:

Lemma 3.8 *The following problem is in* $\mathrm{FP}^{\textsc{FacRoot}} \subseteq \mathrm{PPA}$*: given an odd* $n > 1$*, find an* a *such that* $(a|n) = -1$*, or a nontrivial divisor of* n*.*

Proof: Consider the following algorithm. Put $a = -1$. While $(a|n) = 1$, repeat the following steps: call the FacRoot oracle, if it provides a factor of n, we are done, otherwise we replace a with its square root modulo n.

The algorithm must halt within $\log_2 n$ iterations: if a is a 2^kth root of -1, its order in $(\mathbb{Z}/n\mathbb{Z})^*$ is $2^{k+1} < n$. □

Notice that, conversely, FacRoot is Turing-reducible to WeakFacRoot together with the problem from Lemma 3.8.

In fact, FacRoot does the dual job of factoring and computing square roots. In Theorem 3.6 we have exploited its factoring capacity by supplying it with quadratic nonresidues, but we can also use it the other way round to obtain algorithms for finding square roots and quadratic nonresidues modulo arbitrary integers. We start with the latter.

Theorem 3.9 *The following problem is in* $\mathrm{FP}^{\textsc{FacRoot}} \subseteq \mathrm{PPA}$*: given an odd* n *which is not a perfect square, find an* a *such that* $(a|n) = -1$*.*

Proof: The algorithm maintains a sequence $\langle n_i : i < k\rangle$ of integers $n_i > 1$ such that $n = \prod_{i<k} n_i$, and a sequence $\langle a_i : i < k\rangle$, where some of the a_i may be undefined, but if a_i is defined, then $(a_i|n_i) = -1$. We initialize it with $k = 1$, $n_0 = n$, a_0 undefined, and we repeat in arbitrary order the following steps until neither is applicable any more:

- If $n_i \neq n_j$ are such that $(n_i, n_j) > 1$, we delete n_i, n_j from the sequence and replace them with (n_i, n_j) (two copies), $n_i/(n_i, n_j)$, and $n_j/(n_i, n_j)$, omitting those equal to 1 (this can happen only for one of the four numbers, hence the length of the sequence always increases). The a_i entries corresponding to the new numbers are undefined.
- If a_i is undefined, we call as an oracle the search problem from Lemma 3.8 on n_i. If it returns a nontrivial divisor of n_i, we expand the n_j sequence as in the previous step. Otherwise, it provides a value for a_i.

Since $k \leq \log n$, the algorithm must halt in $O(\|n\|)$ steps. When it does, all a_i are defined, and the n_i entries are pairwise equal or coprime, hence we can write $n = \prod_{i\in I} n_i^{e_i}$ for some $I \subseteq \{0, \dots, k-1\}$ and $e_i > 0$, where n_i, $i \in I$, are pairwise coprime. Since n is not a perfect square, we can pick $i \in I$ such that e_i is odd. By the Chinese remainder theorem, we can compute an a such that $a \equiv a_i \ (n_i^{e_i})$ and $a \equiv 1 \ (n/n_i^{e_i})$. Then

$$\left(\frac{a}{n}\right) = \prod_{j\in I} \left(\frac{a}{n_j}\right)^{e_j} = (-1)^{e_i} = -1. \qquad \Box$$

Corollary 3.10 *The following problem is in* $\mathrm{FP}^{\textsc{FacRoot}} \subseteq \mathrm{PPA}$*: given* $n > 2$*, find an* a *coprime to* n *which is a quadratic nonresidue modulo* n*.*

Proof: If n is a power of 2, we can return 3. Otherwise, we can write $n = 2^e m^{2^k}$, where m is odd and not a perfect square. By Theorem 3.9, we can find a such that $(a|m) = -1$. By adding m to a if necessary, we can make sure a is odd, hence $(n, a) = 1$. Since a is a quadratic nonresidue modulo $m \mid n$, it is also a nonresidue modulo n. □

Another problem we are going to reduce to FacRoot is the computation of square roots modulo n. A priori it is not clear how to formulate it as a total NP search problem, as the quadratic residuosity problem is neither known nor assumed to be poly-time decidable. We can remedy this by requiring the search problem to find something sensible also for quadratic nonresidues.

Definition 3.11 Let n be a positive integer. If $(a, n) = 1$, a divisor $m \mid n$ is a *coprime nonsquare witness for a modulo n* if

- m is odd and $\left(\frac{a}{m}\right) = -1$, or
- $m = 4$ and $a \equiv 3 \ (4)$, or
- $m = 8$ and $a \equiv 5 \ (8)$.

If a is an arbitrary integer, an m is a *nonsquare witness for a modulo n*, if m is not a perfect square, m is odd or 2, and there are e, b, and $j < e$ such that $m^e \mid n$, $a = m^j b$, $(m, b) = 1$, and if j is even, m (if odd) or 4 or 8 (if $m = 2$) is a coprime nonsquare witness for b modulo m^{e-j}.

It is easy to see that the property of being a nonsquare witness is poly-time decidable.

Lemma 3.12 *If there exists a nonsquare witness for a modulo n, then a is a quadratic nonresidue modulo n.*

Proof: If m is a coprime nonsquare witness for a, then a is a quadratic nonresidue modulo m, and a fortiori modulo n.

Let m be a nonsquare witness for a, and let e, b, and j be as in Definition 3.11. Assume for contradiction $a \equiv (uc)^2 \ (n)$, where $(m, c) = 1$, and $u \mid m^k$ for some k. We have $m^j \mid (uc)^2$, hence $m^j \mid u^2$. Moreover, if we write $u^2 = m^j v$, then $b \equiv vc^2 \ (m^{e-j})$, hence $(m, v) = 1$, i.e., $v = 1$ and $m^j = u^2$. Since m is not a perfect square, this implies j is even. However, $b \equiv c^2$ (m^{e-j}) contradicts the fact that b has a coprime nonsquare witness modulo m^{e-j}. □

Theorem 3.13 *The following problem is in* $\mathrm{FP}^{\textsc{FacRoot}} \subseteq \mathrm{PPA}$*: given $n > 0$ and a, find either a square root of a modulo n, or a nonsquare witness for a modulo n.*

Proof: Write $n = 2^e m$ with m odd. In the first stage of our algorithm, we keep a sequence $\langle n_i : i < k \rangle$ of integers $n_i > 1$ such that $m = \prod_{i<k} n_i$, and a sequence of integers $\langle u_i : i < k \rangle$ where some u_i may be undefined. We maintain the property that whenever u_i is defined, we can write $a = n_i^{j_i} a_i$ for some j_i so that $(a_i, n_i) = 1$, and we have $a_i \equiv u_i^2 \ (n_i)$. We start with $k = 1$, $n_0 = m$ and u_0 undefined, and we repeat the following steps until none of them applicable any more:

- If $n_i \neq n_j$ are such that $(n_i, n_j) > 1$, we delete n_i, n_j from the sequence and replace them with two copies of (n_i, n_j), $n_i/(n_i, n_j)$, and $n_j/(n_i, n_j)$ as in the proof of Theorem 3.9.
- If n_i is a perfect square, we replace n_i with two copies of $\sqrt{n_i}$.
- If $a = n_i^{j_i} a_i$ where $n_i \nmid a_i$, but $(n_i, a_i) > 1$, we replace n_i with (n_i, a_i) and $n_i/(n_i, a_i)$.
- If $a = n_i^{j_i} a_i$ where $(a_i|n_i) = 1$, but u_i is undefined, we call a FacRoot oracle on n_i, a_i. If it returns a nontrivial divisor $d \mid n_i$, we replace n_i with d and n_i/d. Otherwise, it returns a square root of a_i modulo n_i, which we store as u_i.

This stage terminates after $O(\|n\|)$ steps. When it does, we can write $m = \prod_{i \in I} n_i^{e_i}$ for some $e_i > 0$, $I \subseteq \{0, \dots, k-1\}$, where n_i, $i \in I$, are pairwise coprime, none of them is a perfect square, and we have $a = n_i^{j_i} a_i$ for some j_i and $(a_i, n_i) = 1$. For each i, we try to compute a square root z_i of a modulo $n_i^{e_i}$ as follows:

- If $j_i \geq e_i$, we put $z_i = 0$.
- If $j_i < e_i$, and j_i is odd or $(a_i|n_i) = -1$, we return n_i as a nonsquare witness for a.
- If $j_i < e_i$ is even and $(a_i|n_i) = 1$, then u_i is defined, and $u_i^2 \equiv a_i \ (n_i)$. We put $z_i = n_i^{j_i/2} v_i$, where $v_i^2 \equiv a_i \ (n_i^{e_i})$ is computed using Hensel's lifting, which is an iteration of the following procedure: if we have u such that $u^2 \equiv a_i \ (n_i^c)$, we compute $w \equiv (2u)^{-1}$ (n_i^c), and we put $u' = (u^2 + a_i)w$. Then $u'^2 \equiv a_i \ (n_i^{2c})$.

We also try to find a square root z of a modulo 2^e. We write $a = 2^j b$ with b odd, and then:

- If $j \geq e$, we put $z = 0$.
- If $j < e$, we return 2 as a nonsquare witness for a whenever one of the following cases happens: j is odd, or $e - j \geq 2$ and $b \equiv 3 \ (4)$, or $e - j \geq 3$ and $b \equiv 5 \ (8)$.
- Otherwise, $j < e$ is even, and $1^2 \equiv b \ (2^{\min\{e-j,3\}})$. We put $z = 2^{j/2} v$, where $v^2 \equiv b$ (2^{e-j}); if $e - j > 3$, we compute v using the following variant of Hensel's lifting. If we have u such that $u^2 \equiv b \ (2^c)$, we compute $w \equiv u^{-1} \ (2^{c-2})$, and we put $u' = ((u^2 + b)/2)w$. Then $u'^2 \equiv b \ (2^{2c-2})$.

Finally, using the Chinese remainder theorem, we compute x such that $x \equiv z \ (2^e)$ and $x \equiv z_i$ $(n_i^{e_i})$ for every i, then $x^2 \equiv a \ (n)$. □

4 FacRoot is in PPA

The purpose of this section is to prove Theorem 3.4. As already mentioned in the introduction, the original idea of the proof comes from previous work of the author on the provability of the quadratic reciprocity theorem in variants of bounded arithmetic, and in fact, FacRoot $\in$ PPA is a simple corollary of these results. This connection is described in detail in Section 4.1. In order to make this paper more self-contained, we give a direct combinatorial proof of Theorem 3.4 in Section 4.2. Readers uncomfortable with bounded arithmetic may safely skip straight there.

4.1 Bounded arithmetic

We assume familiarity with basic facts about subsystems of bounded arithmetic, in particular Buss' theory S^1_2. We refer the reader to [6, 9] for more background.

Jeřábek [8] introduced a theory $S^1_2 + \mathit{Count}_2(PV)$, axiomatized over S^1_2 by the following principle: for every number a and circuit C, C does not define an involution on $\{0, \dots, 2a\}$ without fixpoints. Notice that the axiom is Σ^b_1, and the corresponding search problem is a minor variant of Lonely.

Lemma 4.1 *If $S^1_2 + \mathit{Count}_2(PV) \vdash \forall x \, \exists y \, \varphi(x, y)$, where $\varphi \in \Sigma^b_1$, then the search problem to find a y satisfying $\varphi(x, y)$ given x is in* PPA.

Proof: By the assumption, S^1_2 proves

$$\exists a, C \, \forall u \leq 2a \, (C(C(u)) = u \neq C(u) \leq 2a) \vee \exists y \, \varphi(x, y),$$

hence $S^1_2(h)$ proves its Herbrandization

$$\exists a, C \, (h(a, C) \leq 2a \to C(C(h(a, C))) = h(a, C) \neq C(h(a, C)) \leq 2a) \vee \exists y \, \varphi(x, y).$$

This is an $\exists\Sigma^b_1(h)$ formula, hence using Parikh's theorem and Buss' witnessing theorem, there exists a polynomial-time oracle function f^h such that

$$(*) \qquad \exists a, C \, (h(a, C) \leq 2a \to C(C(h(a, C))) = h(a, C) \neq C(h(a, C)) \leq 2a) \vee \varphi(x, f^h(x))$$

holds in $\mathbb{N}$ for any choice of h. Let us run f on an input x with an oracle solving the PPA-problem corresponding to Count_2 in place of h, and let y be its output. We may assume that f never asks the same question more than once, hence the oracle answers in any particular run can be extended to a function h which satisfies

$$h(a, C) \leq 2a \wedge (C(C(h(a, C))) \neq h(a, C) \vee h(a, C) = C(h(a, C)) \vee C(h(a, C)) > 2a).$$

Then $(*)$ implies $\varphi(x, y)$. Thus, the search problem associated to φ is in $\mathrm{FP}^{\mathrm{PPA}} = \mathrm{PPA}$ using Theorem 2.1. □

Let $J(a, n)$ denote a PV-function formalizing the algorithm in Figure 1. As shown in [8], $S^1_2 + \mathit{Count}_2(PV)$ proves that $J(a, n)$ agrees with the definition of the Jacobi symbol in terms of factorization of n and quadratic residues. In particular, the theory proves that for prime n, $J(a, n) = 1$ implies that a is a quadratic residue, which can be expressed as the following Σ^b_1 formula:

Theorem 4.2 (Jeřábek [8]) *$S^1_2 + \mathit{Count}_2(PV)$ proves*

$$J(a, n) = 1 \to \exists x \, (x^2 \equiv a \quad (n)) \vee \exists u, v < n \, (uv = n). \qquad \square$$

Theorem 3.4 readily follows.

4.2 Explicit algorithm

Before turning to FacRoot proper, we will describe PPA algorithms for some of its special cases which we will need as ingredients in the main construction.

We introduce some notation for conciseness. If n is a fixed odd integer $n > 1$, we consider

$$N = \{x : |x| < n/2, (n, x) = 1\}$$

as a set of unique representatives of $(\mathbb{Z}/n\mathbb{Z})^*$. We also write $N^+ = \{x \in N : x > 0\}$, $N^- = \{x \in N : x < 0\}$, $N_0 = N \cup \{0\}$, and similarly for N_0^+, N_0^-. We assume operations on residues are computed modulo n with a result in N, so that, e.g., $ab^{-1} \in N^+$ means that $a \equiv bx \ (n)$ for some $x \in N^+$.

Lemma 4.3 *There is a poly-time function $f(n, a, x)$ such that for any odd $n > 1$ and an integer a coprime to n, the function $f_{n,a}(x) = f(n, a, x)$ defines an involution on*

$$\{x \in N^- : ax \in N^-\} \cup N_0^+$$

whose fixpoints are of the form x^{-1}, where

(*i*) *$x \in N^+ \smallsetminus \{1\}$ and $x^2 = 1$, or*

(*ii*) *$x \in N^-$ and $x^2 = a$.*

Proof: We define $f'_{n,a}$ on $\{x \in N^- : ax \in N^-\} \cup N^+$ by

$$f'_{n,a}(x) = \begin{cases} x^{-1} & x, x^{-1} \in N^+, \\ a^{-1}x^{-1} & ax, x^{-1} \in N^-, \\ -x & (x, ax \in N^+ \wedge x^{-1} \in N^-) \vee (x, ax \in N^- \wedge x^{-1} \in N^+). \end{cases}$$

It is easy to see that the three conditions define a partition of $\{x \in N^- : ax \in N^-\} \cup N^+$, and $f'_{n,a}$ is an involution on each part. The fixpoints of $f'_{n,a}$ in the first two parts have the forms (i) (without the restriction $x \neq 1$) and (ii), respectively, and there are no fixpoints in the third part. Finally, we put

$$f_{n,a}(x) = \begin{cases} 1 & x = 0, \\ 0 & x = 1, \\ f'_{n,a}(x) & x \neq 0, 1. \end{cases} \qquad \square$$

Definition 4.4 For any constant a, let FacRoot$_a$ denote the following special case of FacRoot: given an odd positive n such that $(a|n) = 1$, find either a nontrivial divisor of n, or a square root of a modulo n.

Lemma 4.5 $\textsc{FacRoot}_{-1}$ *and* $\textsc{FacRoot}_2$ *are in* PPA.

Proof: Given $n \equiv \pm 1 \pmod 8$, observe that

$$\{x \in N^- : 2x \in N^-\} = N \cap [-(n-2\pm 1)/4, -1].$$

We define an involution r on $[-(n-2\pm 1)/4, (n-1)/2]$ by

$$r(x) = \begin{cases} x & x \neq 0, (x,n) \neq 1, \\ f_{n,2}(x) & \text{otherwise.} \end{cases}$$

The domain of r is an interval of size $(3n \pm 1)/4$, which is odd, hence we can use Lonely to find a fixpoint x of r. Using Lemma 4.3, we see that either x^{-1} is a square root of 2, or it is a square root of 1 distinct from ± 1, or $(x,n) \neq 1$. In the last two cases, we can factorize n.

For $\textsc{FacRoot}_{-1}$, we define similarly an involution on $[0, (n-1)/2]$ using $f_{n,-1}$. □

Using a similar construction, it is possible to show $\textsc{FacRoot}_a \in \text{PPA}$ for every constant a. We skip the details, as we will not directly need this fact, and Theorem 3.4 is more general. However, notice that $\textsc{FacRoot}_{-1} \in \text{PPA}$ restates Buresh-Oppenheim's original result, and any constant a yields a similar special case of factoring:

Example 4.6 The following search problems are in PPA.

Given $n \equiv \pm 1 \pmod 8$ such that 2 is a quadratic nonresidue modulo n (i.e., n has a divisor $p \equiv \pm 3 \pmod 8$), find a nontrivial divisor of n.

Given $n \equiv 1 \pmod 3$ such that -3 is a quadratic nonresidue modulo n (i.e., n has a divisor $p \equiv 2 \pmod 3$), find a nontrivial divisor of n.

Lemma 4.7 FacRootMul (*and thus* WeakFacRoot) *is in* PPA.

Proof: Let $n > 1$ be odd, and a, b coprime to n. Define

$$g(x) = \begin{cases} \langle 0, -x \rangle & x \in N_0^+, \\ \langle 1, -x \rangle & x, ax, b^{-1}x \in N^-, \\ \langle 2, -x \rangle & x, ax \in N^-, b^{-1}x \in N^+. \end{cases}$$

Then g is a poly-time bijection from $\{x \in N^- : ax \in N^-\} \cup N_0^+$ onto

$$\begin{aligned} A = (\{0\} \times N_0^-) &\cup (\{1\} \times \{x : x, ax, b^{-1}x \in N^+\}) \\ &\cup (\{2\} \times \{x \in N^+ : ax \in N^+, b^{-1}x \in N^-\}) \end{aligned}$$

with a poly-time inverse. Similarly,

$$h(x) = \begin{cases} \langle 0, x \rangle & x \in N^+, \\ \langle 1, x \rangle & x = 0 \text{ or } x, b^{-1}x, ax \in N^-, \\ \langle 2, x \rangle & x, b^{-1}x \in N^-, ax \in N^+ \end{cases}$$

is a bijection from $\{x \in N^- : b^{-1}x \in N^-\} \cup N_0^+$ onto

$$\begin{aligned} B = (\{0\} \times N^+) \cup (\{1\} \times (\{0\} \cup \{x : x, ax, b^{-1}x \in N^-\})) \\ \cup (\{2\} \times \{x \in N^- : ax \in N^+, b^{-1}x \in N^-\}), \end{aligned}$$

$x \mapsto \langle 2, bx\rangle$ is a bijection from $\{x \in N^- : abx \in N^-\} \cup N_0^+$ onto

$$C = \{2\} \times (\{0\} \cup \{x : ax \in N^- \vee b^{-1}x \in N^+\}),$$

and $\langle 1, x\rangle \mapsto \langle 1, -x\rangle$ is a fixpoint-free involution on

$$D = \{1\} \times \{x \in N : x, ax, b^{-1}x \text{ do not have the same sign}\}.$$

We can thus define a poly-time involution r on $\{0, 1, 2\} \times [-(n-1)/2, (n-1)/2]$ by

$$r(e, x) = \begin{cases} g(f_{n,a}(g^{-1}(e, x))) & \langle e, x\rangle \in A, \\ h(f_{n,b^{-1}}(h^{-1}(e, x))) & \langle e, x\rangle \in B, \\ \langle 2, bf_{n,ab}(b^{-1}x)\rangle & \langle e, x\rangle \in C, \\ \langle 1, -x\rangle & \langle e, x\rangle \in D, \\ \langle e, x\rangle & x \neq 0, (x, n) > 1. \end{cases}$$

Since $3n$ is odd, we can use LONELY to find a fixpoint $\langle e, x\rangle$ of r. We cannot have $\langle e, x\rangle \in D$. If $x \neq 0$, $(x, n) > 1$, we can factor n. If $\langle e, x\rangle \in A$, then $y := g^{-1}(e, x)$ is a fixpoint of $f_{n,a}$. Thus, either $y^2 = 1$, $y \neq \pm 1$, in which case we can factor n, or y^{-1} is a square root of a. Similarly, if $\langle e, x\rangle \in B \cup C$, we can factor n, or compute a square root of b or ab. □

Lemma 4.7 is enough to prove our main result, Theorem 3.6. However, we will proceed with the proof of Theorem 3.4, as we are interested in the possibility of unconditional derandomization of the reduction of factoring to PPA, and placing FACROOT in PPA can be seen as a partial step to achieve that goal. Moreover, randomized versions of Theorems 3.9 and 3.13 would not be interesting.

Lemma 4.8 *The following problems are in* PPA.

(*i*) FACROOTODD*: given an odd $n > 0$, a sequence $\langle a_i : i < k\rangle$ of integers coprime to n such that k is odd, and a square root x of $\prod_{i<k} a_i$ modulo n, find a nontrivial divisor of n, or a square root of some a_i modulo n.*

(*ii*) FACROOTEVEN*: given an odd $n > 0$, and a sequence $\langle a_i : i < k\rangle$ of integers coprime to n such that k is even, find a nontrivial divisor of n, or a square root of $\prod_{i<k} a_i$ or of some a_i modulo n.*

Proof: (i): Put $I = \{0, \dots, k-1\}$ and $y = 1$, and repeat the following steps. If $I = \{i\}$, return xy^{-1} as a square root of a_i. If $|I| > 1$, pick $i, j \in I$, $i \neq j$, and call FACROOTMUL on n, a_i, a_j. If it gives us a nontrivial divisor of n, or a square root of a_i or a_j, we return it. Otherwise, it provides a square root z of $a_i a_j$. We multiply y by z, remove i, j from I, and repeat the loop.

(ii): Put $x = a_k = \prod_{i<k} a_i$, and call FACROOTODD. □

Definition 4.9 QuadRec is the following problem: given odd coprime $n, m > 0$ such that $n \equiv 1\ (4)$, and a square root a of n modulo m, find a nontrivial divisor of n or m, or a square root of m modulo n.

Notice that QuadRec is a special case of FacRoot: the input data ensure $(n|m) = 1$, hence $(m|n) = 1$ by quadratic reciprocity.

Lemma 4.10 QuadRec $\in$ PPA.

Proof: We may assume $n, m > 1$ and $a \in M^-$, so that $b = a^{-1} \in M$ is a fixpoint of $f_{m,n}$. Put $n_2 = (n+1)/2$, $m_2 = (m+1)/2$. The function

$$g(x) = \begin{cases} \langle x, m_2 \rangle & x \in N_0^+, \\ \langle -x, \lfloor -mx/n \rfloor \rangle & x, mx \in N^- \end{cases}$$

is a poly-time bijection with poly-time inverse from $\{x \in N^- : mx \in N^-\} \cup N_0^+$ onto

$$A = \bigl(N_0^+ \times \{m_2\}\bigr) \cup \bigl\{\langle x, y \rangle \in N_0^+ \times [0, m_2) : mx - ny \in N^+\bigr\},$$

where $mx - ny$ is *not* evaluated modulo n, but literally. Likewise,

$$h(y) = \begin{cases} \langle n_2, y \rangle & y \in M_0^+, \\ \langle \lfloor -ny/m \rfloor, -y \rangle & y, ny \in M^- \end{cases}$$

is a bijection from $\{y \in M^- : ny \in M^-\} \cup M_0^+$ onto

$$B = \bigl(\{n_2\} \times M_0^+\bigr) \cup \bigl\{\langle x, y \rangle \in [0, n_2) \times M_0^+ : mx - ny \in M^-\bigr\}.$$

The function $k(x, y) = \langle n_2 - 1 - x, m_2 - 1 - y \rangle$ is a poly-time involution with no fixpoints on

$$C = \bigl\{\langle x, y \rangle \in [0, n_2) \times [0, m_2) : mx - ny \ge n_2 \text{ or } mx - ny \le -m_2\bigr\}.$$

We define a poly-time involution r on $([0, n_2] \times [0, m_2]) \smallsetminus \{\langle n_2, m_2 \rangle\}$ by

$$r(x, y) = \begin{cases} g(f_{n,m}(g^{-1}(x, y))) & \langle x, y \rangle \in A, \\ h(f_{m,n}(h^{-1}(x, y))) & \langle x, y \rangle \in B \smallsetminus \{h(b)\}, \\ \langle 0, 0 \rangle & \langle x, y \rangle = h(b), \\ h(b) & x = y = 0, \\ k(x, y) & \langle x, y \rangle \in C, \\ \langle x, y \rangle & \text{otherwise.} \end{cases}$$

Notice that if $x \in [0, n_2)$ and $y \in [0, m_2)$ are such that $mx - ny = 0$, then $x = y = 0$, as $(n, m) = 1$. It follows that the last clause in the definition of r applies to elements of the set

$$\begin{aligned} D = \bigl(&([1, n_2) \smallsetminus N^+) \times \{m_2\}\bigr) \cup \bigl(\{n_2\} \times ([1, m_2) \smallsetminus M^+)\bigr) \\ &\cup \bigl\{\langle x, y \rangle \in [0, n_2) \times [0, m_2) : mx - ny \in ([1, n_2) \smallsetminus N^+) \cup ((-m_2, -1] \smallsetminus M^-)\bigr\}. \end{aligned}$$

The domain of r has odd size $(n_2+1)(m_2+1)-1$, hence using Lonely, we can find a fixpoint $\langle x, y\rangle$ of r. If $\langle x, y\rangle \in A$, it gives us a square root of m modulo n, or a square root of 1 distinct from ± 1, in which case we can factorize n. If $\langle x, y\rangle \in B$, we get a square root of n modulo m distinct from $\pm a$, or a square root of 1 distinct from ± 1, and both cases give a factor of m. If $\langle x, y\rangle \in D$, (n, x) or (m, y) is a nontrivial divisor of n or m, respectively. □

We are ready now to prove Theorem 3.4. Assume we are given an odd $n > 0$, and an integer a such that $(a|n) = 1$. We first compute the sequences $\langle a_i : i \le t\rangle$, $\langle n_i : i \le t\rangle$ of values of a and n during the execution of the algorithm in Figure 1. That is, we put $\langle a_0, n_0\rangle = \langle a, n\rangle$, and then we define $\langle a_i, n_i\rangle$ by induction on i as follows. If $|a_i| > n_i/2$, we let $n_{i+1} = n_i$, and $a_{i+1} \equiv a_i \pmod{n_i}$ such that $|a_{i+1}| < n_i/2$. If $0 < |a_i| < n_i/2$, we define

$$\langle a_{i+1}, n_{i+1}\rangle = \begin{cases} \langle -a_i, n_i\rangle & a_i < 0, \\ \langle a_i/2, n_i\rangle & a_i > 0 \text{ is even}, \\ \langle n_i, a_i\rangle & a_i > 0 \text{ is odd}. \end{cases}$$

We stop when we reach $a_t = 0$. Since $(a|n) = 1$, we have $(a_i, n_i) = 1$ for each i, in particular $n_t = 1$. Notice that $t = O(\|n\|)$. Write $R = \{i < t : a_i \text{ is odd}, 0 < a_i < n_i/2\}$.

In the main part of the algorithm, we maintain a double sequence $\langle n_{i,j} : i \le t, j < s_i\rangle$ of integers $n_{i,j} > 1$ such that $n_i = \prod_{j<s_i} n_{i,j}$, and $n_{i,j} \le n_{i,j'}$ for $j < j'$. Moreover, we maintain sequences $\langle u_{i,j} : i \le k, j < s_i\rangle$, $\langle v_{i,j,k}, w_{i,j,k} : i \in R, j < s_i, k < s_{i+1}\rangle$, where some of the $u_{i,j}$, $v_{i,j,k}$, and $w_{i,j,k}$ may be undefined. Where they are defined, we have $u_{i,j}^2 \equiv a_i \pmod{n_{i,j}}$, $v_{i,j,k}^2 \equiv n_{i+1,k} \pmod{n_{i,j}}$, and $w_{i,j,k}^2 \equiv n_{i,j} \pmod{n_{i+1,k}}$, respectively.

We initialize the sequences with $s_i = 1$, $n_{i,0} = n_i$ for $n_i > 1$, $s_i = 0$ for $n_i = 1$, and all $u_{i,j}$, $v_{i,j,k}$, and $w_{i,j,k}$ undefined. We repeat in arbitrary order the following updating steps until none of them is applicable any more.

- Assume $n_i = n_{i+1}$, $n_{i,j} \ne n_{i+1,k}$, and $d = (n_{i,j}, n_{i+1,k}) > 1$. If $d \ne n_{i,j}$, we increase s_i, replace $n_{i,j}$ with d and $n_{i,j}/d$, and undefine all associated $u_{i,j}$, $v_{i-1,l,j}$, and $w_{i-1,l,j}$. If $d \ne n_{i+1,k}$, we deal with it similarly. Notice that we cannot have $n_{i,j} = d = n_{i+1,k}$.

  Moreover, if this step is not applicable, then $n_i = n_{i+1}$ implies that $s_i = s_{i+1}$ and $\langle n_{i,j} : j < s_i\rangle$ and $\langle n_{i+1,k} : k < s_{i+1}\rangle$ are permutations of each other, hence in view of their monotonicity, we have $n_{i,j} = n_{i+1,j}$ for each j.

- For $i < t$ such that $\langle n_{i,j} : j < s_i\rangle = \langle n_{i+1,k} : k < s_{i+1}\rangle$ (which implies $n_i = n_{i+1}$):
  - If $a_i \equiv a_{i+1} \pmod{n_i}$, and exactly one of $u_{i,j}$, $u_{i+1,j}$ is defined, we define the other to the same value.
  - If $a_i = \alpha a_{i+1}$, $\alpha \in \{-1, 2\}$, $(\alpha|n_{i,j}) = -1$, and neither $u_{i,j}$ nor $u_{i+1,j}$ is defined, we call FacRootMul$(n_{i,j}, a_i, a_{i+1})$. If it returns a nontrivial divisor of $n_{i,j}$, we expand the $n_{i,j}$ sequence as in the first step. Otherwise, it gives a square root of a_i or a_{i+1} modulo $n_{i,j}$, which we store as $u_{i,j}$ or $u_{i+1,j}$, respectively.
  - If $a_i = \alpha a_{i+1}$, $\alpha \in \{-1, 2\}$, $(\alpha|n_{i,j}) = 1$, and exactly one of $u_{i,j}$ or $u_{i+1,j}$ is defined, we call FacRoot$_\alpha(n_{i,j})$. If it returns a nontrivial divisor of $n_{i,j}$, we expand the

$n_{i,j}$ sequence. Otherwise, it gives $\beta^2 \equiv \alpha\ (n_{i,j})$, and we define $u_{i,j} := \beta u_{i+1,j}$ or $u_{i+1,j} := \beta^{-1} u_{i,j}$, respectively.

- For $i \in R$:
  - If $u_{i,j}$ is defined and $|I|$ is odd, where $I = \{k < s_{i+1} : v_{i,j,k} \text{ is undefined}\}$, we put $x = u_{i,j} \prod_{k \notin I} v_{i,j,k}^{-1}$, and call FacRootOdd on $n_{i,j}, \langle n_{i+1,k} : k \in I\rangle, x$. If it returns a factor of $n_{i,j}$, we expand the $n_{i,j}$ sequence. Otherwise, it returns a square root of some $n_{i+1,k}$, $k \in I$, modulo $n_{i,j}$, which we store as $v_{i,j,k}$.
  - If $u_{i,j}$ is undefined and $|I|$ is even, where I is as above, we call FacRootEven on $n_{i,j}, \langle n_{i+1,k} : k \in I\rangle$. If it returns a factor of $n_{i,j}$, we expand the $n_{i,j}$ sequence. If it returns a square root of some $n_{i+1,k}$, $k \in I$, modulo $n_{i,j}$, we store it as $v_{i,j,k}$. Otherwise, it returns a square root x of $\prod_{k \in I} n_{i+1,k}$, and then we define $u_{i,j} = x \prod_{k \notin I} v_{i,j,k}$.
  - If $u_{i+1,k}$ is defined and $|I|$ is odd, or $u_{i+1,k}$ is undefined and $|I|$ is even, where $I = \{j < s_i : w_{i,j,k} \text{ is undefined}\}$, we proceed in a similar way to expand the $n_{i+1,k}$ sequence or to define some $w_{i,j,k}$ or $u_{i+1,k}$.
  - If $n_{i,j} \equiv -1\ (4)$, $(n_{i+1,k}|n_{i,j}) = 1$, and $v_{i,j,k}$ is undefined, we call WeakFacRoot on $n_{i,j}, n_{i+1,k}, -1$. If it returns a factor of $n_{i,j}$, we expand the $n_{i,j}$ sequence, otherwise it returns a square root of $n_{i+1,k}$ modulo $n_{i,j}$, which we store as $v_{i,j,k}$.
  - If $n_{i+1,k} \equiv -1\ (4)$, $(n_{i,j}|n_{i+1,k}) = 1$, and $w_{i,j,k}$ is undefined, we proceed similarly.
  - If $n_{i,j} \equiv 1\ (4)$, $w_{i,j,k}$ is defined, and $v_{i,j,k}$ is undefined, we call QuadRec on $n_{i,j}, n_{i+1,k}, w_{i,j,k}$. If it returns a factor of $n_{i,j}$ or $n_{i+1,k}$, we expand the $n_{i,j}$ or $n_{i+1,k}$ sequence (respectively), otherwise it returns a square root of $n_{i+1,k}$ modulo $n_{i,j}$, which we store as $v_{i,j,k}$.
  - If $n_{i+1,k} \equiv 1\ (4)$, $v_{i,j,k}$ is defined, and $w_{i,j,k}$ is undefined, we proceed similarly.

In each step, either $\sum_{i \le t} s_i \le O(\|n\|^2)$ strictly increases, or it stays the same, and we define some previously undefined $u_{i,j}$, $v_{i,j,k}$, or $w_{i,j,k}$. It follows that the update procedure stops after $\|n\|^{O(1)}$ steps.

Let us write $[a_i|n_{i,j}] = 1$ if $u_{i,j}$ is defined, and $[a_i|n_{i,j}] = -1$ otherwise. We define $[n_{i+1,k}|n_{i,j}]$ and $[n_{i,j}|n_{i+1,k}]$ similarly using $v_{i,j,k}$ and $w_{i,j,k}$, respectively. Notice that $[a_i|n_{i,j}] = 1$ implies $(a_i|n_{i,j}) = 1$, and likewise for $[n_{i+1,k}|n_{i,j}]$, $[n_{i,j}|n_{i+1,k}]$.

Lemma 4.11 *When the update procedure stops, the following properties hold.*

(i) *If* $n_i = n_{i+1}$, *then* $s_i = s_{i+1}$ *and* $n_{i,j} = n_{i+1,j}$.

(ii) *If* $n_i = n_{i+1}$ *and* $a_i \equiv a_{i+1}\ (n_i)$, *then* $[a_i|n_{i,j}] = [a_{i+1}|n_{i+1,j}]$.

(iii) *If* $n_i = n_{i+1}$ *and* $a_i = \alpha a_{i+1}$, $\alpha \in \{-1, 2\}$, *then*

$$\left[\frac{a_{i+1}}{n_{i+1,j}}\right] = \left(\frac{\alpha}{n_{i,j}}\right)\left[\frac{a_i}{n_{i,j}}\right].$$

(*iv*) *If $i \in R$, then*

$$\left[\frac{a_i}{n_{i,j}}\right] = \prod_{k<s_{i+1}} \left[\frac{n_{i+1,k}}{n_{i,j}}\right], \qquad \left[\frac{a_{i+1}}{n_{i+1,k}}\right] = \prod_{j<s_i} \left[\frac{n_{i,j}}{n_{i+1,k}}\right].$$

(*v*) *If $i \in R$ and $n_{i,j} \equiv n_{i+1,k} \equiv -1$ (4), then*

$$\left[\frac{n_{i+1,k}}{n_{i,j}}\right]\left[\frac{n_{i,j}}{n_{i+1,k}}\right] = -1.$$

(*vi*) *If $i \in R$ and $n_{i,j} \equiv 1$ (4) or $n_{i+1,k} \equiv 1$ (4), then*

$$\left[\frac{n_{i+1,k}}{n_{i,j}}\right]\left[\frac{n_{i,j}}{n_{i+1,k}}\right] = 1.$$

Proof: (i), (ii), and (iv) are clear.

(iii): The statement is clear if $(\alpha|n_{i,j}) = 1$. If $(\alpha|n_{i,j}) = -1$, the inapplicability of update steps implies that $[a_i|n_{i,j}] = 1$ or $[a_{i+1}|n_{i+1,j}] = 1$. We cannot have both, since this would imply $(a_i|n_{i,j}) = (a_{i+1}|n_{i,j}) = 1$, hence $(\alpha|n_{i,j}) = 1$.

(v): By quadratic reciprocity, exactly one of $(n_{i+1,k}|n_{i,j}) = 1$, $(n_{i,j}|n_{i+1,k}) = 1$ holds. The inapplicability of update steps then implies that $[n_{i+1,k}|n_{i,j}] = 1$ or $[n_{i,j}|n_{i+1,k}] = 1$. We cannot have both, as this would mean $(n_{i+1,k}|n_{i,j}) = (n_{i,j}|n_{i+1,k}) = 1$.

(vi): The statement is clear if $n_{i,j} \equiv n_{i+1,k} \equiv 1$ (4). Assume $n_{i,j} \equiv 1$ (4) and $n_{i+1,k} \equiv -1$ (4), the other case is symmetric. By the inapplicability of update steps, $[n_{i,j}|n_{i+1,k}] = 1$ implies $[n_{i+1,k}|n_{i,j}] = 1$. On the other hand, if $[n_{i+1,k}|n_{i,j}] = 1$, then $(n_{i+1,k}|n_{i,j}) = 1$, hence $(n_{i,j}|n_{i+1,k}) = 1$ by quadratic reciprocity, thus $[n_{i,j}|n_{i+1,k}] = 1$ by the inapplicability of update steps. □

Using Lemma 4.11, we can show

$$\prod_{j<s_i}\left[\frac{a_i}{n_{i,j}}\right] = \left(\frac{a_i}{n_i}\right)$$

by reverse induction on i. The induction step for $i \in R$ goes as follows:

$$\begin{aligned}\prod_{j<s_i}\left[\frac{a_i}{n_{i,j}}\right] &= \prod_{\substack{j<s_i\\k<s_{i+1}}}\left[\frac{n_{i+1,k}}{n_{i,j}}\right]\\ &= \prod_{\substack{j<s_i\\k<s_{i+1}}}\left[\frac{n_{i,j}}{n_{i+1,k}}\right](-1)^{(n_{i,j}-1)(n_{i+1,k}-1)/4}\\ &= (-1)^{(a_{i+1}-1)(n_{i+1}-1)/4}\prod_{k<s_{i+1}}\left[\frac{a_{i+1}}{n_{i+1,k}}\right]\\ &= (-1)^{(a_{i+1}-1)(n_{i+1}-1)/4}\left(\frac{a_{i+1}}{n_{i+1}}\right)\\ &= \left(\frac{n_{i+1}}{a_{i+1}}\right) = \left(\frac{a_i}{n_i}\right).\end{aligned}$$

In particular, either $s_0 > 1$, in which case $n_{0,0}$ is a nontrivial divisor of n, or $s_0 = 1$ and $[a_0|n_{0,0}] = 1$, where $a_0 = a$ and $n_{0,0} = n$, in which case $u_{0,0}^2 \equiv a \ (n)$. This completes the proof of Theorem 3.4.

5 Conclusion

We have shown that integer factoring has randomized reductions to the classes PPA and PPP (more precisely, PWPP). We also provided evidence that there in fact exist deterministic reductions, namely this is true under the widely believed assumption of the generalized Riemann hypothesis for quadratic Dirichlet characters.

Problem 5.1 *Is* FACTORING *in* PPA *or* $\mathrm{FP}^{\mathrm{PPP}}$*?*

Some of our other results can be seen as partial indication that such an unconditional deterministic reduction might be possible at least in the case of PPA. In particular, the fact that FACROOT $\in$ PPA bypasses the randomized reduction of WEAKFACROOT to FACROOT, and we have shown that PPA contains the search problems to find square roots modulo arbitrary integers (which is probabilistically Turing-equivalent to factoring) and to find quadratic nonresidues (which is easily solvable in randomized polynomial time). Nevertheless, it remains open whether Problem 5.1 can be resolved unconditionally.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, PRIMES *is in* P, Annals of Mathematics 160 (2004), no. 2, pp. 781–793.

[2] Eric Bach, *Explicit bounds for primality testing and related problems*, Mathematics of Computation 55 (1990), no. 191, pp. 355–380.

[3] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi, *The relative complexity of* NP *search problems*, Journal of Computer and System Sciences 57 (1998), no. 1, pp. 3–19.

[4] Alessandro Berarducci and Benedetto Intrigila, *Combinatorial principles in elementary number theory*, Annals of Pure and Applied Logic 55 (1991), no. 1, pp. 35–50.

[5] Joshua Buresh-Oppenheim, *On the* **TFNP** *complexity of factoring*, unpublished note, `http://www.cs.toronto.edu/~bureshop/factor.pdf`, 2006.

[6] Samuel R. Buss, *First-order proof theory of arithmetic*, in: Handbook of Proof Theory (S. R. Buss, ed.), Studies in Logic and the Foundations of Mathematics vol. 137, Elsevier, Amsterdam, 1998, pp. 79–147.

[7] Samuel R. Buss and Alan S. Johnson, *Propositional proofs and reductions between* NP *search problems*, Annals of Pure and Applied Logic 163 (2012), no. 9, pp. 1163–1182.

[8] Emil Jeřábek, *Abelian groups and quadratic residues in weak arithmetic*, Mathematical Logic Quarterly 56 (2010), no. 3, pp. 262–278.

[9] Jan Krajíček, *Bounded arithmetic, propositional logic, and complexity theory*, Encyclopedia of Mathematics and Its Applications vol. 60, Cambridge University Press, 1995.

[10] Christos H. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs of existence*, Journal of Computer and System Sciences 48 (1994), no. 3, pp. 498–532.

[11] Michael O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.