Improvements to RSA key generation and CRT on embedded devices

Mike Hamburg, Mike Tunstall, and Qinglai Xiao

Rambus, Inc. mhamburg,mtunstall,qxiao@rambus.com

Abstract. RSA key generation requires devices to generate large prime numbers. The naïve approach is to generate candidates at random, and then test each one for (probable) primality. However, it is faster to use a *sieve* method, where the candidates are chosen so as not to be divisible by a list of small prime numbers $\{p_i\}$.

Sieve methods can be somewhat complex and time-consuming, at least by the standards of embedded and hardware implementations, and they can be tricky to defend against side-channel analysis. Here we describe an improvement on Joye et al.'s sieve based on the Chinese Remainder Theorem (CRT). We also describe a new sieve method using quadratic residuosity which is simpler and faster than previously known methods, and which can produce values in desired RSA parameter ranges such as $(2^{n-1/2}, 2^n)$ with minimal additional work. The same methods can be used to generate strong primes and DSA moduli.

We also demonstrate a technique for RSA private key operations using the Chinese Remainder Theorem (RSA-CRT) without $q^{-1} \bmod p$. This technique also leads to inversion-free batch RSA and inversion-free RSA mod $p^k q$.

We demonstrate how an embedded device can use our key generation and RSA-CRT techniques to perform RSA efficiently without storing the private key itself: only a symmetric seed and one or two short hints are required.

Keywords: RSA · prime generation

1 Introduction

To generate private keys for the RSA cryptosystem [RSA78], devices must choose random, secret prime numbers. Prime number generation is also required for finite-field Diffie-Hellman (DH) and DSA parameter generation [DH76,KG13]. DH and DSA parameter generation has become a more common requirement since the Logjam attack [ABD$^+$15], which allows multiple DH and DSA keys to be attacked together if they use the same parameter set.

Prime generation algorithms may use sieving techniques to reduce the number of candidates that must be tested. [JPV00] describes two sieving methods: one based on the Chinese Remainder Theorem (CRT) and one based on Carmichael's λ function; the latter is improved in [JP06]. Here we describe an improvement to the CRT sieve to mitigate its largest downside, namely a large precomputed table of CRT coefficients. We also describe a novel sieving algorithm based on quadratic residuosity, which may be more resistant to side-channel attack than a CRT-based sieve.

Our improved sieving algorithms work well with the other known techniques for generating RSA keys, DSA keys and strong primes on embedded devices [JPV00,JP03]. With some modification, our CRT-based sieve can be used to efficiently generate safe primes as well.

The RSA private operation is also often implemented using the CRT, which quarters the computation time. The CRT requires an extra value, $q^{-1} \bmod p$, which is typically computed during key generation and stored with the private key. We show how to modify a side-channel countermeasure to perform RSA-CRT efficiently without this value, simplifying key generation and storage. The technique generalizes trivially to multi-prime RSA. Less trivially, it generalizes to inverse-free RSA modulo $p^k q$ [Tak98,Tak04], which previously required inversion not only of $q \bmod p$ but also of $e \bmod p$.

In fact, these are applications of a more general batching technique [Ham12] which we briefly recap in Section 3.2. This batching technique can also be applied to implement batch RSA [Fia90] without inversion.

For embedded devices whose nonvolatile memory consists only of fuses, the cost of storing an RSA private key is significant. It would be preferable if the private key could be expanded from a secret seed — perhaps even a PUF key — instead of being stored. Our two techniques combine to enable RSA-CRT from a compressed form of the private key, consisting of a symmetric seed plus one or two 16-bit hints. This compression incurs only a few percent performance loss.

1.1 Notation

Let $\mathbb{R}$ denote the real numbers. Let $\mathbb{Z}$ and $\mathbb{Z}/n$ denote the integers and the ring of integers mod an integer n, respectively. Let $\mathbb{F}_{p^e}$ denote the Galois field of p^e elements. Call two integers (m, n) *coprime* if their greatest common divisor is 1. Let $(\mathbb{Z}/n)^*$ be the multiplicative group of $\mathbb{Z}/n$, which contains the elements $m \in \mathbb{Z}/n$ which are coprime to n.

For positive integers (p, e, n), let $p|n$ or $p \nmid n$ mean that p divides or does not divide n, respectively, and let $p^e||n$ mean that p^e divides n but p^{e+1} does not. In all cases where this notation is used, p is a prime number. For brevity we sometimes omit that qualification in notations such as "for all $p^e||n$".

Let $\phi(n)$ and $\lambda(n)$ denote the Euler and Carmichael totient functions, respectively:

$$\phi(n) := \prod_{p^e||n} p^{e-1} \cdot (p-1) \qquad \text{and} \qquad \lambda(n) := \mathrm{LCM}\ \{p^{e-1} \cdot (p-1) \text{ for all } p^e||n\}$$

For all $x \in (\mathbb{Z}/n)^*, x^{\phi(n)} = x^{\lambda(n)} = 1$.

For integers (x, p), we say that x is a *quadratic residue* (resp *quadratic non-residue*) mod p if there exists (resp does not exist) an integer y such that $x \equiv y^2 \bmod p$. We will only consider quadratic (non)residues modulo prime p.

For any ring R, and any group G with group operation $\odot$, and any functions $F_1, F_2 : G \to R$, their convolution $F_1 * F_2$ is defined as:

$$(F_1 * F_2)(x) := \sum_{x_1 \odot x_2 = x} F_1(x_1) \cdot F_2(x_2).$$

The power-convolution F_1^{*k} is defined as the convolution $F_1 * F_1 * \ldots * F_1$ of k copies of F_1.

A probability distribution $\mathcal{D}$ on a finite set S may be seen as a *stochastic* function $S \to \mathbb{R}$, meaning a function such that $\mathcal{D}(x) \geqslant 0$ for each $x \in S$, and $\sum_{x \in S} \mathcal{D}(x) = 1$. If S is a group, then this allows us to convolve distributions. This gives the distribution of the product of two samples:

$$\mathcal{D}_1 * \mathcal{D}_2 = \{x_1 \odot x_2 : x_1 \leftarrow \mathcal{D}_1, x_2 \leftarrow \mathcal{D}_2\}.$$

The notation $x \xleftarrow{\$} S$ means to choose an element uniformly at random from a set S. The notation $[A, B]$ means the interval from A to B, inclusive.

2 Generating prime numbers

2.1 Naïve algorithm

Generating random prime numbers is, in some sense, simple. There are well-established probabilistic primality tests[1] [Rab80,PSW80,AM93] that work for large numbers, and an approximately $1/(n \ln 2)$ fraction of the numbers less than 2^n are prime. So we can just choose random numbers and test them for (probable) primality. If a 1024-bit prime is desired, it will take about $1024 \cdot \ln 2 \approx 710$ tries in expectation, but may take much longer if the generator is unlucky.

This naïve algorithm is shown in Algorithm 1. However, typically the test "if p is prime" is somewhat slow, requiring an exponentiation in the case of a Fermat or Miller-Rabin test. The primality test may be sped up somewhat by using trial division by several small primes $\{p_i\}$ before testing, but this is not especially fast either. Furthermore, it risks revealing information about p mod p_i via a side-channel such as power consumption.

2.2 Sieving algorithms

The naïve algorithm's performance can be improved by choosing p in a way that is guaranteed not to be divisible by small primes; for example, we might choose $x \in (\mathbb{Z}/M)^*$, for a constant M which is divisible by many small primes. Since we wish to generate primes in a certain range — an interval $[L, H]$ — we can then adjust x to be in that range without changing its value mod M.

[1] Most of these algorithms exhibit false positives in rare cases. That is, when given a prime number they always say that it is prime, but they may accept a composite number as prime with some tiny probability. The present work does not address this issue.

Algorithm 1 Naïve prime generation

procedure PrimeGen(L, H, t) $\triangleright$ Try t times to generate a prime in $[L, H]$
  for $i = 1$ to t **do**
    $p \xleftarrow{\$} [L, H]$
    if p is prime **then return** p
  end for
  return Failure
end procedure

This sieving method is shown in Algorithm 2, which is a variant of Joye et al.'s sieving algorithm [JPV00, Figure 6]. This algorithm samples from the slightly narrower interval $\left[L, L + \left\lfloor \frac{H-L}{M} \right\rfloor \cdot M\right]$. If this is close enough to H, it may be acceptable; otherwise we can instead sample from the slightly wider interval $\left[L, L + \left\lceil \frac{H-L}{M} \right\rceil \cdot M\right]$ and reject candidates that are greater than H.

Algorithm 2 Prime generation using sieve

procedure PrimeGen(L, H, t) $\triangleright$ Try t times to generate a prime in $[L, H]$
  Let M be a product of small primes.
  $x \xleftarrow{\$} (\mathbb{Z}/M)^*$ $\triangleright$ This step is tricky
  for $i = 1$ to t **do**
    $\alpha \xleftarrow{\$} [0, \lfloor (H - L)/M \rfloor - 1]$
    $p \leftarrow L + (x - L \bmod M) + \alpha M$ $\triangleright$ Choose $p \equiv x \bmod M$
    if p is prime **then return** p
    $x \leftarrow$ Next(x) $\triangleright$ May just be $x \xleftarrow{\$} (\mathbb{Z}/M)^*$ again
  end for
  return Failure
end procedure

This sieving algorithm provides a considerable speedup, of approximately $M/\phi(M)$. For example, to taking M the 1019-bit product of the first 131 primes, this is a factor of ≈ 11.8, improving 1024-bit prime generation from 710 tries to 60 tries in expectation.

To increase performance, the sieving algorithm does not necessarily repeat the sampling procedure for each candidate p. Instead, it updates the sample $x \leftarrow$ Next(x), where Next is some (possibly randomized) update function. Joye et al. take Next$(x) := 2 \cdot x \bmod M$. This forces them to take M odd; to avoid running the primality test on even p, they add M if p is even. The choice of a deterministic update function is problematic, because it allows side-channel attackers to accumulate information about x across several iterations [CC07]. It also reduces the entropy of the resulting primes, because the algorithm is more likely to choose primes p such that $p/2^i \bmod M$ is composite for the first few i.

The difficulty remains in sampling efficiently from $(\mathbb{Z}/M)^*$. The samples should be nearly uniform[2] in $(\mathbb{Z}/M)^*$. Rejection sampling would work, but it is slow for large M, and calculating $\mathrm{GCD}(x, M)$ to test coprimality has side-channel concerns [CAB20].

Joye-Paillier-Vaudenay CRT sieve Joye, Paillier and Vaudenay suggest to sample $(\mathbb{Z}/M)^*$ using the Chinese Remainder Theorem [JPV00, Figure 3]. Let $[\![M_i]\!]_{i=1}^n$ be a sequence of mutually coprime integers – Joye et al. take them to be prime powers. Let $M := \prod_i M_i$, and precompute a sequence $[\![\theta_i]\!]_{i=1}^n$ where $\theta_i \equiv 1 \mod M_i$ and $\theta_i \equiv 0 \mod M_j$ for all $j \neq i$. Then one can sample $x \xleftarrow{\$} (\mathbb{Z}/M)^*$ as

$$x \leftarrow \left(\sum x_i \cdot \theta_i\right) \mod M \qquad \text{where each} \qquad x_i \xleftarrow{\$} (\mathbb{Z}/M_i)^*.$$

Here sampling from $(\mathbb{Z}/M_i)^*$ may be much faster and simpler than sampling from $(\mathbb{Z}/M)^*$. If M_i is a prime power, we just need to choose a sample that is not divisible by p. For M_i of other forms, sampling algorithms will still be simpler and faster with short M_i (e.g. one machine word) than with long ones. The simplest approach is just to sample at random and then reject if $\mathrm{GCD}(M_i, x_i) \neq 1$.

However, this method has a significant disadvantage: it requires precomputing and storing a list of large numbers $[\![\theta_i]\!]_{i=1}^n$. It may also face issues with template attacks on the values of x_i.

Improved CRT sieve However, we observe that it is not required to take $\theta_i \equiv 1 \mod M_i$. Indeed, it is only required that θ_i is coprime to M_i, and divisible by each M_j for $j \neq i$. So we can instead take $\theta_i := M/M_i$, avoiding the need to store it. That is, we can take

$$x \leftarrow \left(\sum x_i \cdot (M/M_i)\right) \mod M \qquad \text{where each} \qquad x_i \xleftarrow{\$} (\mathbb{Z}/M_i)^*.$$

In fact, we can avoid the division by computing the sum iteratively, as shown in Algorithm 3. This algorithm is at least as fast as the Joye-Paillier-Vaudenay version, but does not require storage of $[\![\theta_i]\!]_{i=1}^n$.

We can use a similar technique to improve the NEXT algorithm, so that it is randomized to deter side-channel attacks. We can do this by choosing a random M_i, sampling $y_i \xleftarrow{\$} (\mathbb{Z}/M_i)^*$, and returning

$$x \cdot (y_i \cdot (M/M_i) + M_i) \bmod M.$$

This works because the factor $y_i \cdot (M/M_i) + M_i$ is always coprime to M:

- It is congruent to $y_i \cdot (M/M_i) \mod M_i$, and this value is coprime to M_i by construction.
- It is congruent to $M_i \mod M_j$ for $j \neq i$, and again M_i is coprime to M_j.

[2] They need not be cryptographically indistinguishable from uniform. In practice, a wide variety of not-quite-uniform distributions are used [SNS+16]. This seems to be sufficient so long as (p, q) are close enough to uniform and are uncorrelated [NSS+17].

Algorithm 3 Improved sampling from $(\mathbb{Z}/M)^*$ using CRT

procedure $\textsc{Sample}([\![M_i]\!]_{i=1}^{n})$
    $x \leftarrow 0$
    $M \leftarrow 1$
    for $i = 1$ to n **do**
        $x_i \xleftarrow{\$} (\mathbb{Z}/M_i)^*$
        $x \leftarrow (x \cdot M_i + x_i \cdot M) \bmod (M \cdot M_i)$
        $M \leftarrow M \cdot M_i$
    end for
    return x
end procedure

Joye-Paillier sieve with Carmichael's $\boldsymbol{\lambda}$ However, there is still a side-channel issue with CRT-based samplers: the domain of each x_i is small, and so may be subject to template attacks. It would be preferable to implement a sieve that uses only large random numbers.

Joye and Paillier suggest to sample from $(\mathbb{Z}/M)^*$ as shown in [JP06, Figure 4], reproduced in Algorithm 4. This algorithm is based on Carmichael's observation that for prime $p^e|M$,

$$x^{\lambda(M)} \bmod p^e = \begin{cases} 0 \text{ if } p|x \\ 1 \text{ otherwise} \end{cases}$$

So the update $x \leftarrow x + r \cdot (1 - x^{\lambda(M)})$ only affects $x \bmod p^e$ if $p|x$.

Algorithm 4 Sampling from $(\mathbb{Z}/M)^*$ using Carmichael's λ

procedure $\textsc{Sample}(M, \lambda(M))$
    $x \xleftarrow{\$} \mathbb{Z}/M$
    $z \leftarrow 1 - x^{\lambda(M)} \bmod M$
    while $z \neq 0$ **do**
        $r \xleftarrow{\$} \mathbb{Z}/M$
        $x \leftarrow x + rz$
        $z \leftarrow 1 - x^{\lambda(M)} \bmod M$
    end while
    return x
end procedure

The sampling algorithm is somewhat slow: 2.15 iterations are required in expectation, and each iteration requires an exponentiation mod M. If M is again the 1019-bit product of the first 131 primes, then $\lambda(M)$ has 276 bits. Therefore overall sampling from $(\mathbb{Z}/M)^*$ is about 58% as expensive as a Fermat or Miller-Rabin primality test of the same size, so sampling independently before every primality test would cause a noticeable slowdown. Because the performance decreases as $\lambda(M)$ increases, this method works best if M has only small prime factors; or at least if for all primes $p|M, p-1$ has only small prime factors.

2.3 New sampling algorithm with quadratic residuosity

Here we describe a novel sieving algorithm using quadratic residuosity. We expect this method to resist side-channel attacks because it performs only a few calculations, and all intermediate values have high entropy.

Let M be an odd number; a good choice is the product of the first n odd primes, but we can use any odd number of known factorization. Let u be chosen such that $-u$ is a quadratic nonresidue mod each prime $p|M$. Call such a u "valid" mod M. If the factorization of M is known, then it is straightforward to find valid u using the Chinese Remainder Theorem, as we will soon describe. The values (M, u) can be precomputed, and stored in read-only memory (ROM) on the device that needs to generate primes, or they can be calculated on the fly to save ROM.

Then for all $r \in \mathbb{Z}$, by definition $r^2 \not\equiv -u$ mod each $p|M$. So $r^2 + u$ is not divisible by any $p|M$: it is coprime to M. With (M, u) precomputed, the prime generation algorithm can very easily sample from $(\mathbb{Z}/M)^*$, simply by choosing x at random and computing $r^2 + u$ mod M. The same technique could be used with any other polynomial function that does not have a root modulo p_i, such as $ur^2 + 1$, but $r^2 + u$ is simple and requires only one multiplication.

These samples are not uniformly random: in particular, they cover only about half of $(\mathbb{Z}/p^e)^*$ for each $p^e||m$. So if M is divisible by n distinct primes, the range is only slightly more than a 2^{-n} fraction of $(\mathbb{Z}/M)^*$. But we will show that the product of several independent samples approaches a uniformly random distribution on $(\mathbb{Z}/M)^*$. Since prime generation algorithms usually do not require perfectly uniform output, a product of between 4 and 10 such samples will be close enough to uniform for most practical purposes, as shown in Figure 1. We suggest using 6 samples, which loses less than 0.11 bits of min-entropy for all M.

If a system is equipped with a fast random number generator, then the new sieving technique is fast enough (11 multiplies mod M for 6 samples, compared to several hundred for Algorithm 4) that we do not need to use an update function $\textsc{Next}(x)$. We can just choose a fresh sample x every time. However, if the random number generator is somewhat slow, we can set $\textsc{Next}(x) = x \cdot (y^2 + u) \bmod M$, where y is a fresh random sample. This improves on $\textsc{Next}(x) = 2x \bmod M$: it is more uniform, and it mitigates side-channel leakage related to x. This version is shown in Algorithm 5. Note that Line 6 guarantees that p is odd and coprime to M, and that $p \in [L \cdot 2s, L \cdot 2s + 2M - 1]$.

Note also that it is easy to sample $r \leftarrow \mathbb{Z}/M$ with a high degree of uniformity. Simply set R to be a power of 2 (or of the machine's word size) such that $R > 2^{64} \cdot M$ (or an even larger bound); choose $r \xleftarrow{\$} [0, R-1]$; and then reduce r mod M.

Algorithm 5 Prime generation with novel sieving algorithm

procedure PrimeGen(L, H, s, t) ▷ Generate a nearly random prime in $[L \cdot 2s, H \cdot 2s]$
    Let M be odd of known factorization, such that $M < H - L$ but only slightly.
    Choose u so that $-u$ is a QNR mod all odd primes dividing M.
    $x \leftarrow \prod_{j=1}^{6}(r_j^2 + u) \bmod M$, where each $r_j \xleftarrow{\$} \mathbb{Z}/M$.
    for $i = 1$ to t **do**
        $p \leftarrow L \cdot 2s + (2x + M - L \cdot 2s \bmod 2M)$
        $\alpha \xleftarrow{\$} [0, s-1]$
        $p \leftarrow p + 2M\alpha$
        if p is prime **then return** p
        $r \xleftarrow{\$} \mathbb{Z}/M$
        $x \leftarrow x \cdot (r^2 + u) \bmod M$
    end for
    return Failure
end procedure

Variants With M odd, this approach works with no modifications when using power-of-2 Montgomery multiplication and Montgomery reduction mod M: if x is coprime to M, then so is MontReduce(x). Before primality testing, x can be made odd, or 3 mod 4 for easier Miller-Rabin implementation, by adding a suitable multiple of M.

On systems where modular multiplication does not use Montgomery reduction, the modulus $2M$ can be used instead, and the candidates can then be guaranteed to be odd. Specifically, we can sample candidate primes as

$$\prod_{i=1}^{k}(2(r_i^2 + u) + M) \bmod 2M.$$

Likewise, x can be constrained to be 3 mod 4. Constrain u to be 1 mod 4, and sample candidate primes as

$$-\prod_{i=1}^{k}((2r_i)^2 + u) \bmod 4M.$$

Or again, we can sample x from $(\mathbb{Z}/M)^*$ as usual and then test $(4x + cM)$ mod $4M$ for primality, where $c \in \{1, 3\}$ is chosen such that $cM \equiv 3$ mod 4. The same techniques can be used to ensure that $x \equiv 2$ mod 3, which is required for RSA with $e = 3$.

Uniformity mod M Algorithm 5 draws samples from the distribution

$$\mathcal{D}_{M,k,u} := \prod_{i=1}^{k}(x_i^2 + u) \bmod M : x_i \xleftarrow{\$} [0, M).$$

How close is $\mathcal{D}_{M,k,u}$ to the uniform distribution $\mathcal{U}_M$ on $(\mathbb{Z}/M)^*$? We will bound the maximum difference in probability to sample each x mod a prime power:

$$\|\mathcal{D}_{p^e,k,u} - \mathcal{U}_{p^e}\|_\infty := \max_{x \in (\mathbb{Z}/p^e)^*} |\Pr[\mathcal{D}_{p^e,k,u} = x] - \Pr[\mathcal{U}_{p^e} = x]|$$

This in turn will allow us to bound the L_1 distance

$$\begin{aligned}\|\mathcal{D}_{M,k,u} - \mathcal{U}_M\|_1 &:= \sum_{x \in (\mathbb{Z}/M)^*} |\Pr[\mathcal{D}_{M,k,u} = x] - \Pr[\mathcal{U}_M = x]| \\ &\leqslant \sum_{p^e \| M} \phi(p^e) \cdot \|\mathcal{D}_{p^e,k,u} - \mathcal{U}_{p^e}\|_\infty\end{aligned}$$

and the min-entropy loss

$$\begin{aligned}\delta H_\infty &:= \max_{x \in (\mathbb{Z}/M)^*} \frac{\Pr[\mathcal{D}_{M,k,u} = x]}{\Pr[\mathcal{U}_M] = x]} \\ &\leqslant \sum_{p^e \| M} \frac{\phi(p^e) \cdot \|\mathcal{D}_{p^e,k,u} - \mathcal{U}_{p^e}\|_\infty}{\ln 2}\end{aligned}$$

These three measures do not depend on which u is chosen, so long as it is valid mod M. In practice, min-entropy loss is probably the most relevant: if the adversary can break a single RSA key with probability ϵ when $p \leftarrow \mathcal{U}_M$, then it will succeed with probability at most $\epsilon \cdot 2^{\delta H_\infty}$ when $p \leftarrow \mathcal{D}_{M,k,u}$.

We can bound the L_1 distance using the following theorem, which we prove in Appendix A:

Theorem 1 (Uniformity of $\mathcal{D}_{M,k,u}$). *Let M be a positive odd integer, let u be valid mod M, and let $k \geqslant 4$. Let $\mathcal{U}_M$ be the uniform distribution on $(\mathbb{Z}/M)^*$. Let*

$$\epsilon_{M,k} := \sum_{\text{prime } p | M} \left(\frac{2}{\sqrt{p}}\right)^{\lfloor k/2 \rfloor}$$

Then

$$\|\mathcal{D}_{M,k,u} - \mathcal{U}_M\|_1 < \epsilon_{M,k} \quad \text{and} \quad \delta H_\infty < \frac{\epsilon_{M,k}}{\ln 2}.$$

Note that for $k > 6$, the sum converges for all primes p, so it allows us to prove a bound that does not depend on M.

For concrete (M, k) this theorem is somewhat loose, so we also took an empirical approach to calculate the L_∞ distance. For this approach, we calculated the distribution $\mathcal{D}_{M,k,u}$ for $k \in \{1, 2\}$ with M the product of the first 200 or 1000 odd primes. Then for $3 \leqslant k \leqslant 10$, we were additionally able to extend the bound to powers of those primes using equation (5) from the proof of Theorem 1 (the bound from this equation does not converge for $k \leqslant 2$). Theorem 1 itself then bounds the maximum additional distance that can be seen with even larger M. The result is shown in Figure 1.

.

	First 200 odd primes		First 1000 odd primes		All larger primes	
k	L_1	δH_∞	L_1	δH_∞	L_1	δH_∞
1	2	197.3305	2	996.9990	-	-
2	1.4362	29.1962	1.6889	65.6709	-	-
3	2	4.6741	2	5.6428	-	-
4	0.5510	0.7963	0.5659	0.8164	-	-
5	0.1252	0.1806	0.1255	0.1810	-	-
6	0.0453	0.0653	0.0453	0.0653	0.02989	0.04312
7	0.0157	0.0226	0.0157	0.0226	-	-
8	0.0058	0.0084	0.0058	0.0084	0.00022	0.00031
9	0.0023	0.0033	0.0023	0.0033	-	-
10	0.0010	0.0015	0.0010	0.0015	$3.1 \cdot 10^{-6}$	$4.5 \cdot 10^{-6}$

Fig. 1. Bounds on L_1 distance and min-entropy loss between $\mathcal{D}_{M,k,u}$ and $\mathcal{U}_M$. For $k \geqslant 3$, this includes any power of the given primes, but for $k \in \{1, 2\}$ it only includes the first power. The "all larger primes" column is for all powers of all odd primes beyond the first 1000; the bound in Theorem 1 converges for even $k \geqslant 6$. Note that the L_1 distance cannot be greater than 2.

Choosing *M* The value of M is relatively unconstrained, beyond being odd and of known factorization. If p is random in some range and is coprime to M, then it is prime with probability about $M/(\phi(M)\ln p)$, or twice that if M is odd and p is made odd before testing. For efficiency, M should be chosen as a multiple of the first several odd primes, so that $M/\phi(M)$ is as large as possible. But suppose we wish to generate primes in an interval $[L, H]$. We could generate M by first taking, say, $M_1 < (H - L)/2^{32}$ as a product of the first n odd primes, and then calculating

$$M = M_1 \cdot \left\lfloor \frac{H - L}{2M_1} \right\rfloor.$$

This would result in an M very close to $(H - L)/2$, so that adding $2M \cdot \alpha$ can be skipped, and the distribution would still be close to uniform on $[L, H]$. Or we could choose M such that $(H - L)/(2M)$ is very nearly a power of 2, so that at least sampling α is easier. This improvement is incorporated into Algorithm 5. The flexibility in M is an improvement on the Joye-Paillier sieve, where M should be chosen smooth so that $\lambda(M)$ is small.

Another option is to follow Joye-Paillier by setting M somewhat smaller than $(H - L)/2$, and then adjust L and H to be multiples of M. In that case, α is not typically chosen from a power-of-2 range, but subtracting $2L \bmod M$ can be skipped.

When generating RSA keys, the range is usually chosen as

$$[L, H] = [2^{(b-1)/2}, 2^{b/2} - 1]$$

for some even integer b. That way, if $L \leqslant p, q \leqslant H$, then $2^{b-1} \leqslant p \cdot q < 2^b$; that is, $N = pq$ has exactly b bits. To support this case, we can set M to slightly less than $(H - L)/2$ for the lowest supported value of b. For higher values, $H - L$ is very nearly a power of 2 times M. This makes the sieve efficient in both cases. This technique is similar to [JP06, Figure 5].

Choosing u We must choose a valid u, meaning one such that $-u$ is a quadratic nonresidue mod each prime $p|M$. This can be performed by finding such a u_p mod each p, and then combining these using the CRT. However, we do not need the full CRT, because we do not care exactly what u is mod p. It is sufficient to calculate

$$u = \sum_{p^e||M} u_p \cdot (M/p^e)^2 \mod M$$

where each u_p is a quadratic nonresidue mod p. Then for each $p|M$,

$$-u \equiv -u_p \cdot k_p^2 \mod p \text{ for some nonzero } k_p,$$

so u is also a quadratic nonresidue mod p. This u may also be calculated iteratively, much as in Algorithm 3. For each $p \equiv 3 \bmod 4$, we can take $u_p = 1$.

It is also an interesting question to choose u as small as possible. This issue is discussed in Appendix B.

Supporting multiple parameter sets with less storage If a device supports key generation for multiple sizes, it is preferable (but not necessary) to use a specific M for each size. That is, use larger values of M to generate larger primes, so that more small divisors can be sieved out. The parameters could be stored separately for each M, but there is an opportunity to save space as the larger M values should be (at least nearly) divisible by the smaller ones. So we can sample mod M_1 for the smallest supported parameter size, mod $M_1 \cdot M_2$ for the next size, and in general mod $M = \prod_{i=1}^{n} M_i$ for the nth smallest size or tier of sizes.

There are a few different options for how to do this. The simplest is to store a u which is valid mod all the M_i, and thus mod their product. The u value can be (Montgomery) reduced modulo M before use. It is also possible to store a separate u_i (or reduce u separately) mod each M_i; we could then sample separately mod each M_i and combine them into one sample mod M using Section 2.2. This is likely faster for the first sample due to smaller multiplications, but slower for the `Next` function if it is used.

Or we could combine the parameters as

$$M := \prod_{i=1}^{n} M_i, \quad u := \sum_{i=1}^{n} u_i \cdot (M/M_i)^2 \bmod M$$

and then sample using only the QR sieve mod M.

2.4 Applications

Generating primes for RSA keys Our new sieve simplifies finding primes in a particular range such as $[2^{(b-1)/2}, 2^b]$, which is the slowest step in RSA key generation. Previous work discusses efficient generation of RSA keys once the prime generation step is done [JP03].

One additional issue with RSA key generation is that we must have $e \nmid p-1$. When $e = 3$ this means that $p \equiv 2 \bmod 3$, which can be accommodated as discussed in Section 2.3. Otherwise it can be accomplished by rejection sampling. Or if e is coprime to M, one could sample $x \leftarrow (\mathbb{Z}/M)^*$ and $y \leftarrow \mathbb{Z}/e$ such that both y and $yM - 1$ are coprime to e; and then set the candidate prime to $p \leftarrow x \cdot e + y \cdot M$.

Generating DSA moduli, safe primes and strong primes Some standards require generation of primes with specific properties, such as "strong primes" where $p + 1$ and/or $p - 1$ have large prime factors. Either of our sieve methods can be used to replace the g function in [JPV00, Figures 8, 12] to generate DSA moduli and strong primes respectively.

Generating "safe primes" $p = 2q + 1$, for which q is also prime, is more difficult if we wish to sieve both p and q. However, our CRT-based sieve can be adapted easily enough to match [JPV00, Figure 10]. Joye et al. solve the CRT equations $x \equiv x_i \bmod M_i$ as

$$x \leftarrow \sum_{i=1}^{n} x_i \cdot \theta_i \quad \text{where} \quad \theta_i \bmod M_j = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

Joye et al. rejection sample each x_i such that x_i and $2x_i + 1$ are both in $(\mathbb{Z}/M_i)^*$. We instead compute

$$x \leftarrow \sum_{i=1}^{n} x_i \cdot \theta_i \quad \text{where} \quad \theta_i = \prod_{j \neq i} M_j$$

so we need x_i and $2(x_i \cdot \theta_i) + 1$ both to be in $(\mathbb{Z}/M_i)^*$.

Blinding inversions mod M The sieve can be used for techniques other than prime generation. For example, if for some algorithm we must invert a value x modulo a public constant M, we can use this technique to generate a nearly-uniform r which is coprime to M. We can then compute $x^{-1} \equiv r \cdot (rx)^{-1} \bmod M$ to mitigate side-channel attacks on the inversion process.

3 RSA-CRT without q^{-1} mod p

Let (N, e) be an RSA public key. The RSA private permutation computes $m = x^d \bmod N$, where $d \equiv e^{-1} \bmod \lambda(N)$. However, since the party with the private key also knows the factorization $N = pq$, it is more efficient to compute $m_p = x^{d_p} \bmod p$, where $d_p \equiv e^{-1} \bmod p - 1$, and likewise with q. This information may be combined using the Chinese Remainder Theorem (CRT):

$$m = ((m_p - m_q) \cdot q^{-1} \mod p) \cdot q + m_q.$$

This technique is called RSA-CRT. The RSA-CRT computation requires $q^{-1} \mod p$, which is typically stored as part of the private key; it can also be computed when the key is loaded, but this has performance and potentially side-channel problems [CAB20].

CRT could also be performed as

$$m \equiv (m_p \cdot q^{-1} \mod p) \cdot q \quad + \quad (m_q \cdot p^{-1} \mod q) \cdot p \mod N$$

but this appears to require even more information. However, there is a trick to compute $m_p \cdot q^{-1} \mod p$ without knowing $q^{-1} \mod p$. Choose any $y \in (\mathbb{Z}/p)^*$ and let

$$\begin{aligned} \alpha &:= (xy)^{e-1} \mod p \\ \beta &:= (\alpha \cdot y)^{p-1-d_p} \equiv (\alpha \cdot y)^{-1/e} \mod p \\ m_{p,y} &:= \beta \cdot x \equiv x^{1/e} \cdot y^{-1} \mod p \end{aligned} \tag{1}$$

This computes $m_{p,y}$ using one long exponentiation and one short one, and three multiplications. Setting $y = q$ gives a way to compute RSA-CRT without any inversions.

For multiplicative masking we can instead set $y = rq$ where $r \xleftarrow{\$} (\mathbb{Z}/N)^*$, so that:[3]

$$m_{p,rq} \equiv x^d \cdot (rq)^{-1} \mod p.$$

We can compute $m_{q,rp}$ analogously, and combine to calculate $mr^{-1} \mod N$. That is,

$$m \equiv r \cdot (m_{p,rq} \cdot q + m_{q,rp} \cdot p) \mod N.$$

This allows us to compute RSA-CRT decryption with message blinding, using only (p, q, e, d_p, d_q). The technique is compatible with other blinding techniques for (p, q, d_p, d_d), such as [EL10], and for techniques which skip the step of converting to Montgomery form.

Our technique generalizes to multi-prime with $N = \prod p_i$, where the reconstruction equation is

$$m \equiv \sum \left(m_{p_i} \cdot \left(\frac{N}{p_i}\right)^{-1} \mod p_i \right) \cdot \frac{N}{p_i} \mod N.$$

The inner term $m_{p_i} \cdot (N/p_i)^{-1} \mod p_i$ can be computed using our blinding and inversion technique. Here N/p_i is perhaps better written as $\prod_{j \neq i} p_j$.

3.1 Inverse-free RSA mod $p^k q$

Another fast variant of RSA uses $N = p^k q$ [Tak98]. Our inversion-free CRT technique applies here as well, apparently trivially: we can use equation (1) to

[3] A random $r \xleftarrow{\$} \mathbb{Z}/N$ will be coprime to N with overwhelming probability. But if we wanted to be sure then we could reuse one of our sieve techniques.

compute $x^{d \bmod \phi(p^k)} \cdot (qr)^{-1} \bmod p^k$, and combine this with $x^{d \bmod \phi(q)} \cdot (p^k r)^{-1} \bmod q$.

However, the point of RSA mod $p^k q$ is that $x^d \bmod p^k$ can be accelerated. Instead of computing x^d directly mod p^k, the technique is to calculate $x^{d_p} \bmod p$, where $d_p \equiv e^{-1} \bmod p-1$. This gives a solution to the equation

$$m_1^e \equiv x \mod p^1,$$

which can then be iteratively lifted to a solution $m_k^e \equiv x \bmod p^k$ using Hensel's lemma. This means that the trivial application of our technique will perform poorly, and we still need to compute $e^{-1} \bmod p$ [Tak04].

We will instead compute $x^d \cdot y^{-1}$, by solving the equation

$$(ym)^e \equiv x \mod p^k,$$

again with Hensel lifting. Given a nonzero solution $m_\ell \bmod p^\ell$, we can lift it to a solution $m_{\ell+1} \bmod p^{\ell+1}$ using the Hensel iteration

$$m_{\ell+1} \equiv m_\ell + \frac{x - (ym)_\ell^e}{e \cdot y^e \cdot m_1^{e-1} \mod p} \mod p^{\ell+1},$$

whose denominator $\delta := e \cdot y^e \cdot m_1^{e-1} \bmod p$ is the derivative of $(ym)^e$ with respect to m. We can do this in an inverse-free manner given $m_1 = x^d \cdot y^{-1} \bmod p$ and $\delta^{-1} \bmod p$, where

$$\begin{aligned}
\delta^{-1} &\equiv (e \cdot y^e \cdot m_1^{e-1})^{-1} \mod p \\
&\equiv (ym)^{1-e} \cdot (ye)^{-1} \mod p \\
&\equiv x^{d \cdot (1-e)} \cdot (ye)^{-1} \mod p \\
&\equiv x^{d-1} \cdot (ye)^{-1} \equiv x^d \cdot (yex)^{-1} \mod p
\end{aligned}$$

This value $\delta^{-1} \equiv x^d \cdot (yex)^{-1} \bmod p$ can be computed using the blinding and inversion method from equation (1), and from it we can compute $m_1 \equiv x^d \cdot y^{-1} \equiv \delta^{-1} \cdot ex \bmod p$. As before, we can do this with $y := qr$ for random r, to achieve a blinded, inverse-free CRT algorithm.

Thus, we can extend our technique to inverse-free RSA modulo general products of powers of primes.

3.2 Generalized batching

Our inverse-free CRT technique is an application of a general framework for inversion and root calculations [Ham12] including

$$(x, y) \rightarrow (x^{1/e}, y^{-1}) \mod p$$

when x and y are nonzero. We can do this by calculating

$$\begin{aligned}
\alpha &:= (xy)^{e-1} \\
\beta &:= (\alpha \cdot y)^{-1/e} \equiv x^{1/e-1} \cdot y^{-1} \mod p \\
x^{1/e} &\equiv \beta \cdot xy \mod p \\
y^{-1} &\equiv \alpha \cdot \beta^e \mod p
\end{aligned}$$

The fundamental principle is to consider the exponential lattice $\mathcal{L}$ of expressions the form $x^a \cdot y^b$ for $a, b \in \mathbb{Z}$. For more inputs, a higher-dimensional lattice may be used. The target expression(s) such as $\{x^{1/e}, y^{-1}\}$ lie in a superlattice $\mathcal{L}'$ of volume $1/e$. If (as in this example) $\mathcal{L}'/\mathcal{L}$ is one-dimensional, then we can find an element $z \in \mathcal{L}'$, such that $\{x, y, z\}$ span $\mathcal{L}'$, the coefficients of z are either all positive or all negative, and the target element is spanned by $\{x, y, z\}$ with (small) non-negative coefficients. Typically this is best done by giving z strictly negative coefficients, so that non-negative linear combinations of $\{x, y, z\}$ cover all of $\mathcal{L}'$.

Then z can be computed by calculating $\pm ez$ as a non-negative integer combination of $\{x, y\}$, and then applying the $\pm 1/e$ map (or more generally, using the $\pm k/e$ map for some integer k) at the cost of a single large exponentiation. Since now $\{x, y, z\}$ span the target expressions with small non-negative coefficients, these targets can be calculated using only multiplications and small exponentiations.

This principle generalizes batch RSA [Fia90], Montgomery's batched inversion, and batch inversion and square root [Ham12]. It directly provides an inversion-free variant of batch RSA: for example, batching a message $m_3 = x_3^{1/3}$ and $m_5 = x_5^{1/5}$ can be calculated as:

$$z := (x_3^5 \cdot x_5^3)^{-1/15} = x_3^{-1/3} \cdot x_5^{-1/5}; \quad m_3 = z^5 \cdot x_3^2 \cdot x_5; \quad m_5 = z^9 \cdot x_3^3 \cdot x_5^2.$$

This can be further optimized with an appropriate addition chain, and possibly by choosing a different generator z of the lattice.

These techniques can batch multiple small roots and/or inverses using one large exponentiation if and only if the roots are of relatively prime degrees. Otherwise the quotient $\mathcal{L}'/\mathcal{L}$ has multiple generators, so while a batching technique might provide a speedup in some cases, it will require more than one large exponentiation.

We note that batching techniques can also be used to avoid conversions to Montgomery form. The Montgomery form of a number x is $x \cdot R \bmod p$ for some R. Multiplication and exponentiation are typically faster when the inputs are given in Montgomery form. Division by $R \bmod p$ is fast: it is Montgomery reduction. But multiplication by $R \bmod p$ requires Barrett reduction, which is slower and more complex in hardware. However, consider that x is itself the Montgomery form of another number $\hat{x} := x/R \bmod p$. So we can compute

$$x^{1/e} = (\hat{x} \cdot R)^{1/e} = (\hat{x}^{e-1}/R)^{-1/e} \cdot \hat{x}$$

where the input $\hat{x}$ is given by its Montgomery form x, and now we are only dividing by R instead of multiplying by it. This technique may not be worthwhile by itself, because it requires an extra short exponentiation, but it is essentially free if batching is already in use. As a special case of this, random blinding values can be assumed to already be in Montgomery form.

4 RSA with compressed private keys

Our new sieve and RSA-CRT algorithms give an interesting improvement to *compressed* RSA private keys for devices with limited nonvolatile storage. This can be done easily enough just by replacing the random numbers in the usual RSA key generation algorithm with a pseudorandom generator, and storing only the secret seed for that generator. The private key can then be regenerated from the seed whenever it is needed. But RSA key generation is notoriously slow, so this compression mechanism is usually unacceptable. However, if we record hints indicating on which iterations h_p resp h_q we found p resp q, then p and q can be reconstructed very quickly, skipping all the primality tests. This is easiest if each iteration samples an independent candidate p, so that only the h_pth and h_qth iterations must be performed to reconstruct (p, q).

More specifically, in the key generation algorithm we can replace the random number generator with a pseudorandom function $F_k(i, h, j;\ R)$. Its arguments are:

- the secret seed k;
- a flag $i \in \{0, 1\}$ indicating whether we're generating p or q (or from a larger domain for multi-prime RSA);
- a hint $h \in [0, t-1]$ where t is the maximum number of attempts to find a prime in key generation (e.g. $t = \ln \frac{\phi(M)}{\epsilon \cdot M} \cdot \ln p$ for a failure rate near ϵ);
- a counter $j \in [0, m]$ where m is the number of samples required for uniformity (e.g. $m = 6$);
- and the size R of the desired range.

F_k should return a uniformly pseudorandom integer in $[0, R-1]$. This enables us to sample pseudorandom integers in $[L \cdot 2s, H \cdot 2s]$ which are coprime to M using the SieveSample routine shown in Algorithm 6. We use Algorithm 5 for SieveSample, but other algorithms such as Algorithm 3 would work as well.

The secret primes (p, q) can then be represented by the parameters (L, H, s, e), the secret seed k and the hints h_p and h_q. The private key can be reconstructed by calling SieveSample:

$$p = \textsc{SieveSample}(L, H, s, k, 0, h_p) \quad \text{and} \quad q = \textsc{SieveSample}(L, H, s, k, 1, h_q).$$

The other values in the private key, $d \bmod p-1$ and $\bmod\ q-1$, can be reconstructed efficiently using Arazi's lemma and Hensel's lemma as shown in [JP03], reproduced as DMod. A complete compressed RSA algorithm is shown in Algorithm 6. If the negligible probability of failure from line 37 is unacceptable, we

Algorithm 6 RSA with compressed private keys

1: **procedure** SieveSample(L, H, s, k, i, h) ▷ Sample a value in $[L \cdot 2s, H \cdot 2s]$ using $F_k(i, h, \cdot)$
2: Let M be a multiple of many small primes, such that $M < H - L$ but only slightly.
3: Let u be odd such that $-u$ is a QNR mod all odd primes dividing M.
4: $x \leftarrow \prod_{j=1}^{6} \left(F_k(i, h, j;\ M)^2 + u\right) \bmod M$.
5: $\alpha \xleftarrow{\$} F_k(i, h, 0;\ s)$
6: **return** $p \leftarrow L \cdot 2s + (2x + M - L \cdot 2s \bmod 2M) + 2\alpha M$
7: **end procedure**

8: **procedure** CompressedRSAKeygen(L, H, s, e, t, k)
9: **for** $h_p = 0$ to $t - 1$ **do**
10: $p \leftarrow$ SieveSample$(L, H, s, k, 0, h_p)$
11: **if** $e \nmid p - 1$ and p is prime **then goto** line 14
12: **end for**
13: **return** Failure
14:
15: **for** $h_q = 0$ to $t - 1$ **do**
16: $q \leftarrow$ SieveSample$(L, H, s, k, 1, h_q)$
17: **if** $e \nmid q - 1$ and q is prime **then goto** line 20
18: **end for**
19: **return** Failure
20:
21: **return** public key $(p \cdot q, e)$ and compressed private key $(L, H, s, e;\ \ k, h_p, h_q)$
22: **end procedure**

23: **procedure** DMod(e, ϕ, H) ▷ Computes $e^{-1} \bmod \phi < H$ if e is prime and $e \nmid \phi$
24: $R \leftarrow 2^{\lceil \lg H \rceil}$
25: $\bar{e} \leftarrow 1$
26: **for** $i = 1$ to $\lceil \lg \lg R \rceil$ **do** ▷ Compute $\bar{e} \leftarrow e^{-1} \bmod R$
27: $\bar{e} \leftarrow \bar{e} \cdot (2 - e \cdot \bar{e}) \bmod 2^{2^i}$
28: **end for** ▷ In practice, share $\bar{e}$ for the two calls
29: **return** $(1 + (-\phi^{e-2} \bmod e) \cdot \phi) \cdot \bar{e} \bmod R$ ▷ Arazi's lemma
30: **end procedure**

31: **procedure** CompressedRSAPrivate$((L, H, s, e;\ \ k, h_p, h_q),\ \ x))$
32: $p \leftarrow$ SieveSample$(L, H, s, k, 0, h_p)$
33: $q \leftarrow$ SieveSample$(L, H, s, k, 1, h_q)$
34: $d_p \leftarrow$ DMod$(e, p - 1, H \cdot 2s)$
35: $d_q \leftarrow$ DMod$(e, q - 1, H \cdot 2s)$
36: $N \leftarrow pq$
37: $r \xleftarrow{\$} (\mathbb{Z}/N)^*$ ▷ Or $r \xleftarrow{\$} \mathbb{Z}/N$ works with overwhelming probability
38: $\alpha_p \leftarrow (qrx)^{e-1} \bmod p$
39: $m_p \leftarrow (qr \cdot \alpha_p)^{p-1-d_p} \bmod p$
40: $\alpha_q \leftarrow (prx)^{e-1} \bmod q$
41: $m_q \leftarrow (pr \cdot \alpha_q)^{q-1-d_q} \bmod q$
42: **return** $rx \cdot (m_p \cdot q + m_q \cdot p) \bmod N$ ▷ Returns $x^{1/e} \bmod N$
43: **end procedure**

can instead generate $(p, q) \equiv 3 \bmod 4$, and implement that line using Algorithm 5 with $u = 1$.

Suppose we wish to generate 1536-bit primes for RSA-3072, roughly corresponding to 128-bit security. If M is divisible by the first 180 primes so that $\phi(M)/M \approx 0.08$, then each candidate will be prime with probability

$$\Pr[\text{prime}] \approx \frac{M}{1536 \cdot \phi(M) \cdot \ln 2} \approx \frac{1}{85}.$$

If we set $t = 2^{16}$, then CompressedRSA will fail to find a suitable p or q with probability about $2 \cdot e^{-t \cdot \Pr[\text{prime}]} < 2^{-1111}$. So a 3072-bit private RSA key may be compressed to 160 bits with no loss of security: a 128-bit key and two 16-bit hints.

To prevent mistakes, it may also be useful to store (s, e), or to make the pseudorandom function F depend on them, or both. In hardware deployed to a hostile environment, it is also worth adding fault countermeasures, for example a checksum on (p, q, d_p, d_q), to prevent fault attacks [ABF$^+$03].

If k is derived — for example from hardware constants, a master key or a PUF — then only h_p and h_q need to be stored. If k can be chosen by the generator (i.e. it is not a derived key), then storage requirements can be further reduced by removing h_p, and instead re-randomizing k in the first loop. Various other arrangements can be used to trade hint size for key generation performance, such as using a shorter hint h_q and incrementing h_p if no prime q can be found.

Combining the new RSA-CRT technique with Algorithm 6, we can implement RSA efficiently with compressed private keys. For RSA-3072 with $e = 65537$, the calculations of (p, q, d_p, d_q) and the recovery of the final m costs:

- 11 multiplications mod M to sample p, and as many for q.
- 4 multiplications mod R, and several smaller ones, to compute d_p and d_q.
- 19 multiplications mod p, plus one long exponentiation mod p, to compute $q \cdot r$, $\alpha_p \leftarrow (x \cdot qr)^{e-1}$ and $m_p = (qr \cdot \alpha_p)^{p-1-d_p}$; and the same to compute m_q.
- 2 integer multiplications and two multiplications mod N to calculate the final output $m \equiv x \cdot r \cdot (m_p \cdot q + m_q \cdot p) \bmod N$.

Counting the wider multiplications mod N as four, the additional cost of private key compression and blinding together is around 72 large multiplications (mostly squarings) plus a few smaller ones. The exponentiations mod p and q collectively cost some 12882 or 3715 multiplications with the Montgomery ladder and sliding window approaches, respectively, meaning that the additional cost is between 0.6% and 3% of the total runtime.

The same techniques generalize naturally to multi-prime RSA and RSA mod $p^k q$.

5 Future work

We leave to future work the task of evaluating the exact performance, side-channel resistance and fault resistance of these methods, as well as any application to post-quantum RSA [BHLV17,Sch18].

6 Acknowledgements

Special thanks to Denis Pochuev for feedback on RSA with $p^k \cdot q$.

7 Intellectual property disclosure

Some of these techniques may be covered by US and/or international patents.

References

ABD$^+$15. David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, October 2015.

ABF$^+$03. Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 260–275. Springer, Heidelberg, August 2003.

AM93. A Oliver L Atkin and François Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61(203):29–68, 1993.

BHLV17. Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 311–329. Springer, Heidelberg, 2017.

CAB20. Alejandro Cabrera Aldaya and Billy Brumley. When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA. volume 2020, 03 2020.

CC07. Christophe Clavier and Jean-Sébastien Coron. On the implementation of a fast prime generation algorithm. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 443–449. Springer, Heidelberg, September 2007.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

EL10. Nevine Maurice Ebeid and Rob Lambert. A new CRT-RSA algorithm resistant to powerful fault attacks. In *WESS 2010*, page 8. ACM, 2010.

Elk11. Noam D. Elkies. Sum of squares modulo a prime, 2011. `https://mathoverflow.net/questions/69576/sum-of-squares-modulo-a-prime`.

Fia90. Amos Fiat. Batch RSA. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 175–185. Springer, Heidelberg, August 1990.

Ham12. Mike Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/2012/309`.

JP03. Marc Joye and Pascal Paillier. GCD-free algorithms for computing modular inverses. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 243–253. Springer, Heidelberg, September 2003.

JP06. Marc Joye and Pascal Paillier. Fast generation of prime numbers on portable devices: An update. In Louis Goubin and Mitsuru Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 160–173. Springer, Heidelberg, October 2006.

JPV00. Marc Joye, Pascal Paillier, and Serge Vaudenay. Efficient generation of prime numbers. In Çetin Kaya Koç and Christof Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 340–354. Springer, Heidelberg, August 2000.

KG13. Cameron F. Kerry and Patrick D Gallagher. Digital signature standard (DSS). FIPS Pub 186-4, 2013. `https://dx.doi.org/10.6028/NIST.FIPS.186-4`.

NSS+17. Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1631–1648. ACM Press, October / November 2017.

PSW80. Carl Pomerance, John L Selfridge, and Samuel S Wagstaff. The pseudoprimes to $25 \cdot 10^9$. *Mathematics of Computation*, 35(151):1003–1026, 1980.

Rab80. Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

RSA78. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

Sch18. John M. Schanck. Multi-power post-quantum RSA. Cryptology ePrint Archive, Report 2018/325, 2018. `https://eprint.iacr.org/2018/325`.

SNS+16. Petr Svenda, Matús Nemec, Peter Sekan, Rudolf Kvasnovský, David Formánek, David Komárek, and Vashek Matyás. The million-key question - investigating the origins of RSA public keys. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 893–910. USENIX Association, August 2016.

Tak98. Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 318–326. Springer, Heidelberg, August 1998.

Tak04. Tsuyoshi Takagi. A fast RSA-type public-key primitive modulo $p^k q$ using hensel lifting. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 87(1):94–101, 2004.

VOW99. Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12(1):1–28, 1999.

Wag02. David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

A Proof of Theorem 1

Let's start with an easy lemma.

Lemma 1. *Given two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ and a uniform distribution $\mathcal{U}$ on a group of size n, we have*

$$\|\mathcal{D}_1 * \mathcal{D}_2 - \mathcal{U}\|_1 \leqslant \|\mathcal{D}_1 - \mathcal{U}\|_1 \cdot \|\mathcal{D}_2 - \mathcal{U}\|_1 .$$

and

$$\|\mathcal{D}_1 * \mathcal{D}_2 - \mathcal{U}\|_\infty \leqslant n \cdot \|\mathcal{D}_1 - \mathcal{U}\|_\infty \cdot \|\mathcal{D}_2 - \mathcal{U}\|_\infty .$$

Proof. This is true for distributions if and only if it is true for stochastic functions. For $i \in \{1, 2\}$, let $F_i := \mathcal{D}_i - \mathcal{U}$ as functions. Since it is the difference of two stochastic functions, each F_i sums to 0. Then

$$\begin{aligned}\mathcal{D}_1 * \mathcal{D}_2 &= (\mathcal{U} + F_1) * (\mathcal{U} + F_2)\\ &= \mathcal{U} * \mathcal{U} + \mathcal{U} * (F_1 + F_2) + F_1 * F_2\\ &= \mathcal{U} + F_1 * F_2\end{aligned}$$

because F_1 and F_2 each sum to 0. Thus

$$\|\mathcal{D}_1 * \mathcal{D}_2 - \mathcal{U}\|_1 \quad = \quad \|F_1 * F_2\|_1 \quad \leqslant \quad \|F_1\|_1 * \|F_2\|_1 \quad = \quad \|\mathcal{D}_1 - \mathcal{U}\|_1 \cdot \|\mathcal{D}_2 - \mathcal{U}\|_1 .$$

Likewise,

$$\|\mathcal{D}_1 * \mathcal{D}_2 - \mathcal{U}\|_\infty \quad = \quad \|F_1 * F_2\|_\infty \quad \leqslant \quad n \cdot \|F_1\|_\infty * \|F_2\|_\infty \quad = \quad n \cdot \|\mathcal{D}_1 - \mathcal{U}\|_\infty \cdot \|\mathcal{D}_2 - \mathcal{U}\|_\infty$$

as claimed.

We will next bound $\mathcal{D}_{p,2,u}$. The rough argument is that the probability of picking a particular $z \in \mathbb{Z}/p$ is related to the number of solutions to a certain algebraic equation, which we can show is about the expected amount using the Hasse bound.

Lemma 2. *Let p be an odd prime. Then for each $z \in (\mathbb{Z}/p)^*$, we have*

$$\left|\Pr[\mathcal{D}_{p,2,u} = z] - \frac{1}{p-1}\right| \leqslant 2p^{-3/2}$$

That is,

$$\|\mathcal{D}_{p,2,u} - \mathcal{U}_p\|_\infty \leqslant 2p^{-3/2}.$$

Proof. Let p be an odd prime, and let's bound $\Pr[\mathcal{D}_{p,2,u} = z]$ for $z \in \mathbb{F}_p^* := (\mathbb{Z}/p)^*$. This is n/p^2, where n is the number of solutions to

$$E : (x^2 + u) \cdot (y^2 + u) = z : x, y \in \mathbb{F}_p.$$

E isomorphic to an Edwards curve over $\mathbb{F}_{p^2}$, and it is elliptic unless $z = 0$ or $z = u^2$. The case $z = 0$ is ruled out because $z \in (\mathbb{Z}/p)^*$, and we will deal with

$z = u^2$ later. For now, suppose that E is elliptic. E has no points at infinity over $\mathbb{F}_p$: they would have $x^2 + u = \infty$ and $y^2 + u = 0$ or vice-versa, but the latter has no solutions over $\mathbb{F}_p$. Therefore, by the Hasse bound,

$$|n - (p+1)| \leqslant 2\sqrt{p}. \tag{2}$$

We want to show that

$$\left| n - \frac{p^2}{p-1} \right| \leqslant 2\sqrt{p} \tag{3}$$

so that

$$\begin{aligned} \left| \Pr[\mathcal{D}_{p,2,u} = z] - \frac{1}{p-1} \right| &= \frac{1}{p^2} \left| n - \frac{p^2}{p-1} \right| \\ &\leqslant \frac{2\sqrt{p}}{p^2}. \\ &= 2p^{-3/2} \end{aligned}$$

The only way that n could meet bound (2) but not the claimed (3) is if

$$n - (p+1) \in \left[-2\sqrt{p}, -2\sqrt{p} + \frac{1}{p-1} \right]$$

When can this interval contain integers? Empirically it does for $p \in \{2, 3\}$, and does not for $3 < p < 19$. For larger prime p, it also cannot contain integers: if it contained an integer m, then we would have

$$m^2 \in \left[4p - \frac{4\sqrt{p}}{p-1} + \frac{1}{(p-1)^2}, 4p \right]$$

With $p > 18$ we have:

$$\begin{aligned} p^2 &> 18p - 1 \\ (p-1)^2 &= p^2 - 2p + 1 \\ &> 16p \\ p - 1 &> 4\sqrt{p} \end{aligned}$$

so the interval width

$$\frac{4\sqrt{p}}{p-1} - \frac{1}{(p-1)^2} < 1.$$

Therefore it contains no integers other than $4p$, and we cannot have $m^2 = 4p$ because p is prime.

For the case $p = 3$, we must have $u = 1$. Then $\Pr[\mathcal{D}_{p,2,u} = z]$ is $1/3$ for $z = 1$ and $2/3$ for $z = 2$, and in both cases it differs from $1/2$ by $1/6 < 2/3^{3/2}$ as claimed.

Finally, what about the case $z = u^2$, so that E is not elliptic? In this case, we will show that there are exactly p or $p + 2$ solutions in $\mathbb{Z}/p$. The solutions have either $x = 0$, in which case also $y = 0$, or (x, y) nonzero and satisfying:

$$\begin{aligned} x^2 \cdot y^2 + u(x^2 + y^2) &= 0 \\ x^2(y^2 + u) &= -uy^2 \\ y^2 + u &= -u(y/x)^2 \end{aligned}$$

which is a non-degenerate ellipse in variables y and $w := y/x$. Every non-degenerate ellipse has exactly $p + 1$ points (y, w) in the projective plane, and none of the points on this ellipse are at infinity, but only the points with (y, w) nonzero lift to unique solutions in $(x, y) \in E$. None of the points have $w = 0$ because $y^2 + u \neq 0$. If $y = 0$ the equation reduces to $w^2 = -1$, which has two solutions for $p \equiv 1 \bmod 4$ and none for $p \equiv 3 \bmod 4$. So there are either $p - 1$ or $p + 1$ nonzero solutions (y, w) in these respective cases, for a total of p or $p + 2$ solutions respectively. So in this case

$$|n - (p + 1)| = 1 < 2\sqrt{p}$$

as well. This completes the proof of Lemma 2.

The following lemma extends Lemma 2 to $\mathcal{D}_{p^e,k,u}$ for even $k \geqslant 4$. The idea is to begin with $e = 1$, where we can bound the convolution of several copies of $\mathcal{D}_{p^e,2,u}$ using Lemma 1. For larger e, we can then apply Hensel lifting. However, the Hensel argument fails when all the x's are equal to zero. That case introduces an additional term, which violates the bound when $k = 2$ but becomes tiny when $k \geqslant 4$.

Lemma 3. *Let p be an odd prime, $k \geqslant 4$ be an even integer, and e be a positive integer. Then*

$$\|\mathcal{D}_{p^e,k,u} - \mathcal{U}_{p^e}\|_\infty \leqslant \frac{2^{k/2}}{p^{e+k/4}}$$

Proof. First, let's handle the case that $e = 1$ by induction on k. Here Lemma 2 gives us a base case for $k = 2$:

$$\|\mathcal{D}_{p,2,u} - \mathcal{U}_p\|_\infty \leqslant \frac{2^{2/2}}{p^{1+2/4}}$$

For larger even values of k, we apply Lemma 1 with $n = p - 1$ to get:

$$\begin{aligned} \|\mathcal{D}_{p,k,u} - \mathcal{U}_p\|_\infty &\leqslant (p-1) \cdot \|\mathcal{D}_{p,k-2,u} - \mathcal{U}_p\|_\infty \cdot \|\mathcal{D}_{p,2,u} - \mathcal{U}_p\|_\infty \\ &\leqslant (p-1) \cdot \frac{2^{k/2-1}}{p^{1+(k-2)/4}} \cdot \frac{2}{p^{1+1/2}} \\ &= \frac{2^{k/2}}{p^{1+k/4}} \cdot \left(1 - \frac{1}{p}\right), \end{aligned} \tag{4}$$

which is a slightly stronger version of the claim.

For $e > 1$, we will divide samples into equivalence classes according to a certain relation mod p^e. If two samples are equivalent mod p^e they all output the same value $z \equiv \prod(x_i^2 + u) \bmod p^e$, and also they are equivalent mod $p^{e'}$ for each $e' < e$. We call each class "zero" or "nonzero" according to whether it contains the solution $(0, 0, \ldots, 0)$.

Let a class C' mod p^{e-1} output some z' mod p^{e-1}. We define the *lifting probability* for C' to z mod p^e as

$$\Pr[\text{lift to } z : \ C] := \Pr\left[S \text{ outputs } z \bmod p^e : S \xleftarrow{\$} C\right]$$

We will show two proposition:

Proposition 1. *Nonzero solutions mod p^{e-1} will lift to nonzero solutions mod p in the following way. Let C be a nonzero equivalence class of solutions to $\prod(x_i^2 + u) \equiv z \bmod p^{e-1}$. Then for all $z' \in (\mathbb{Z}/p^e)^*$ with $z' \equiv z \bmod p^{e-1}$,*

$$\Pr[\text{lift to } z : \ C] = \frac{1}{p}.$$

Proposition 2. *If a sample S is chosen uniformly at random, then:*

$$\Pr[S \in C_{0,e-1}] = p^{-\lceil (e-1)/2 \rceil k}.$$

Furthermore,

$$\left| \Pr[\text{lift to } z : \ C_{0,e-1}] - \frac{1}{p} \right| \leqslant \alpha_e$$

where $\alpha_e \leqslant 1$ for even e, and $\alpha_e \leqslant 1/p^{k/2} \leqslant 1/p^2$ for odd e.

Once we have proven these two propositions, we can prove the main theorem by strong induction. Let "zero(p^e)" denote the event that a sample lies in $C_{0,e}$. Abbreviate

$$\delta_{p^e,k} := p^e \cdot \|\Pr\left[\mathcal{D}_{p^e,k,u}\right] - U_{p^e}\|$$

Applying the propositions, we have

$$\begin{aligned}
\delta_{p^e,k} &\leqslant \delta_{p^{e-1},k} + p^e \cdot \alpha_e \cdot \Pr[\text{zero}(p^{e-1})] \\
&\leqslant \delta_{p^{e-1},k} + p^e \cdot \begin{cases} p^{-e/2 \cdot k} & \text{if } e \text{ is even} \\ 1/p \cdot p^{-(e-1)/2 \cdot k} & \text{if } e \text{ is odd} \end{cases}
\end{aligned}$$

Letting e be odd and applying this twice, we thus have

$$\begin{aligned}
\delta_{p^e,k} &\leqslant \delta_{p^{e-2},k} + p^e \cdot p^{-1-(e-1)/2 \cdot k} + p^{e-1} \cdot p^{-(e-1)/2 \cdot k} \\
&= \delta_{p^{e-2},k} + 2p^{e-1-(e-1)/2 \cdot k} \\
&= \delta_{p^{e-2},k} + 2p^{-(e-1)/2 \cdot (k-2)}
\end{aligned}$$

Summing this up from $e = 3$ to ∞, we have that for all e,

$$\begin{aligned}\delta_{p^e,k} &\leqslant \delta_{p,k} + 2 \sum_{e=3 \text{ odd}}^{\infty} p^{-(e-1)/2\cdot(k-2)} \\ &= \delta_{p,k} + \frac{2}{p^{k-2}-1}\end{aligned} \tag{5}$$

For $p \geqslant 3$ and $k \geqslant 4$, plugging in (4) and

$$\frac{2}{p^{k-2}-1} < \frac{4}{p^{k-2}} < \frac{2^{k/2}}{p^{1+k/4}}$$

gives $\delta_{p,k} \leqslant \frac{2^{k/2}}{p^{k/4}}$ as claimed. But it remains to prove Propositions 1 and 2.

Proof of Proposition 1 Next we will define the equivalence classes and prove Proposition 1. This step is essentially a Hensel lift. Consider two tuples of the form

$$X := (x_1, x_2, \ldots, x_k) \text{ and } X' := (x'_1, x'_2, \ldots, x'_k),$$

For each i, let f_i be the maximum integer such that $f_i \leqslant \lceil e/2 \rceil$ and $p^{f_i} | x_i$. Define $x_i =: p^{f_i} \cdot y_i$ and likewise define f'_i and y'_i. Then for all d_i,

$$x_i^2 + u \equiv (p^{f_i}(y_i + d_i p^{e-2f_i}))^2 + u \mod p^e$$

and, on the contrary, if $f_i < \lceil e/2 \rceil$ then values of the form

$$(p^{f_i}(y_i + d_i p^{e-2f_i-1}))^2 + u$$

span all values equivalent to $x_i^2 + u \bmod p^{e-1}$. Thus, we will call these tuples equivalent if for each i, $f_i = f'_i$ and $y'_i \equiv y_i \bmod p^{e-2f_i}$. We call an equivalence class C nonzero if it doesn't contain $(0, 0, \ldots, 0)$, which is equivalent to having at least one $f_i < \lceil e/2 \rceil$.

Suppose that X is in a nonzero class $C \bmod p^{e-1}$, such that $\prod(x_i^2 + u) \equiv z \bmod p^{e-1}$. Then for each $z' \equiv z \bmod p^{e-1}$, the class C samples $z' \bmod p^e$ with probability $1/p$. To see this, let $f_j < (e-1)/2$ and write

$$\prod_i (x_i^2 + u) = \left((p^{f_j}(y_j + d_j p^{e-2f_j}))^2 + u\right) \cdot \prod_{i \neq j} (x_i^2 + u)$$

The latter term is nonzero mod p and thus invertible, and the former term samples each value equivalent to $z / \prod_{i \neq j}(x_i^2 + u) \bmod p^{e-1}$ each with probability $1/p$. This completes the proof of the Proposition 1.

Proof of Proposition 2 As defined, a class is zero mod p^{e-1} if and only if each $f_i = \lceil (e-1)/2 \rceil$, which is to say if each x_i is divisible by $p^{\lceil (e-1)/2 \rceil}$. This happens with probability $p^{k\lceil (e-1)/2 \rceil}$.

It remains to show that for odd e and $z = u^k + cp^{e-1}$, the probability that the zero class mod p^{e-1} samples z mod p^e is between $1/p - 1/p^{k/2}$ and $1/p + 1/p^{k/2}$. The resulting equation is

$$\prod \left((p^{\frac{e-1}{2}} d_i)^2 + u \right) \equiv u^k + cp^{e-1} \mod p^e$$

which is equivalent to

$$u^{k-1} \sum_{i=1}^{k} d_i^2 \equiv c \mod p.$$

The leading u^{k-1} is invertible and may be discarded. The remaining distribution may be bounded by [Elk11], which considers a convolution S_p^{*k} of k copies of the distribution $S_p := \{d_i^2 \bmod p : d_i \leftarrow \mathbb{Z}/p\}$. The Fourier coefficients

$$\widehat{S}(j) := \frac{1}{p} \sum_{x=0}^{p-1} e^{2\pi j x^2/p}$$

of S are a Gauss sum, and so are equal to 1 for $j = 0$ and to $1/\sqrt{\pm p}$ for $j \neq 0$. There are $p - 1$ coefficients with $j \neq 0$. The Fourier transform of the uniform distribution U_p has coefficients $\widehat{U}_p(0) = 1$ and 0 elsewhere, so

$$\left\| \widehat{S_p^{*k}} - \widehat{U}_p \right\|_2^2 = \frac{p-1}{p^k}.$$

Therefore

$$\begin{aligned}
\left| \Pr[S_p^{*k} = c] - \frac{1}{p} \right| &\leqslant \left\| S_p^{*k} - U \right\|_2 \\
&= \sqrt{\frac{1}{p} \left\| \widehat{S_p^{*k}} - \widehat{U} \right\|_2^2} \\
&= \sqrt{\frac{p-1}{p^{k+1}}} \quad < \quad \frac{1}{p^{k/2}}
\end{aligned}$$

This completes the proof of Proposition 2 and Lemma 3.

We are now ready to prove the theorem.

Theorem 1 (Uniformity of $\mathcal{D}_{M,k,u}$). *Let M be a positive odd integer, let u be valid mod M, and let $k \geqslant 4$. Let $\mathcal{U}_M$ be the uniform distribution on $(\mathbb{Z}/M)^*$. Let*

$$\epsilon_{M,k} := \sum_{\text{prime } p|M} \left(\frac{2}{\sqrt{p}} \right)^{\lfloor k/2 \rfloor}$$

Then

$$\left\| \mathcal{D}_{M,k,u} - \mathcal{U}_M \right\|_1 < \epsilon_{M,k} \quad \textit{and} \quad \delta H_\infty < \frac{\epsilon_{M,k}}{\ln 2}.$$

Proof. We note that the claimed bounds are at most additive for powers $p^e||M$, so it suffices to prove them for each $p^e||M$. It also suffices to consider only even k, because convolving with another copy of $\mathcal{D}_{M,1,u}$ cannot increase either quantity, nor does going from even k to $k+1$ change $\epsilon_{M,k}$. The L_1 distance follows immediately from Lemma 3, because

$$\|\mathcal{D}_{p^e,k,u} - U_{p^e}\|_1 < p^e \|\mathcal{D}_{p^e,k,u} - U_{p^e}\|_\infty \leqslant \frac{2^{k/2}}{p^{k/4}}.$$

For the min-entropy loss, let $n := \phi(p^e)$, and note that for each $z \in (\mathbb{Z}/p^e)^*$,

$$\begin{aligned}\delta H_\infty &\leqslant \ln zn \\ &= \log_2(1 + n(z - 1/n)) \\ &\leqslant \frac{1}{\ln 2} \cdot n(z - 1/n) \\ &\leqslant \frac{1}{\ln 2} \cdot n \left\|\mathcal{D}_{p^e,k,u} - \mathcal{U}_p^e\right\|_\infty\end{aligned}$$

This completes the theorem.

B Minimizing u

We say that u is "valid" mod M if $\left(\frac{-u}{p}\right) = -1$ for all primes $p|M$. If M's factorization is known, then it is easy to find a valid u_p modulo each $p|M$ (e.g. by checking the Jacobi symbol $\left(\frac{-u_p}{p}\right)$ until a valid u_p is found), and to combine them using the Chinese Remainder Theorem. But what is the minimum valid u? Using a smaller u could allow the same u to be used for several values of M, or could reduce memory usage and compute time, but mostly it is a mathematically interesting question. For simplicity, we assume here that M is square-free.

If there are n primes dividing M, then a random element of $(\mathbb{Z}/M)^*$ is valid with probability 2^{-n}, so we expect the minimum valid u to be around $u_{\text{minexp}} := 2^n \cdot M/\phi(M)$. A brute-force strategy would require about u_{minexp} work, which is infeasible past the first 50 primes or so. But this work can be reduced somewhat, particularly if we settle for a small but not minimal u.

B.1 Sparse solutions to linear equations

The most effective method we found was to search for valid u of the form $u = q_1 \cdot q_2 \cdots q_m$ where the q_i's are in some set Q. The validity criterion is that:

$$\text{for each prime } p|M, \quad \left(\frac{-u}{p}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{q_1}{p}\right) \cdots \left(\frac{q_m}{p}\right) \tag{6}$$

If each q_i is coprime to M, then the Jacobi symbols are all either -1 or 1; mapping these to 1 and 0 respectively translates the validity criterion to a system of affine equations over $\mathbb{F}_2$. This allows us to solve for u with xor-list or sparse solution techniques, such as:

– A birthday attack or stronger collision technique [VOW99] for $m = 2$ and Q a large set (e.g. $|Q| \approx 2^{32}$).
– Wagner's xor-list algorithm [Wag02] for m small and Q a large set.
– Information set decoding for large m and a relatively small set Q (e.g. the first 1000 primes not dividing M).

Using a birthday attack, we discovered that the 59-bit value

$$u = \texttt{0x4b0555d761f3f52}$$

is valid mod the 383-bit product of the first 59 odd primes. We also used Wagner's algorithm to search for u a product of four 32-bit odd numbers, requiring it to be valid mod at least the first 72 odd primes. We ran the algorithm for a day on a 64-core Amazon EC2 Graviton2 instance, producing some 5 million results. Notably,

$$u = \texttt{0xe3b0f73b0050ab294417001ad1e63d}$$

is valid mod the 729-bit product of the first 99 odd primes. Our search was tuned to find u relatively close to u_{minexp}; tuning it differently would have been faster or found valid u mod more primes, but the resulting u would be significantly larger.

It isn't necessary to choose M before u. One could start with a small u which is valid mod the first several primes, and then choose further primes $p|M$ so that u is valid. This sacrifices some performance, because discarding small primes reduces $M/\phi(M)$. Our search using Wagner's algorithm found that

$$u = \texttt{0x23e9ee9bd621b0b248e8b59a4c80bb55}$$

performs well across a range of bit sizes, losing about 0.5% of performance compared to an unconstrained (M, u) at 1024 bits and 3% at 2048 bits.

The quality of results from Wagner's algorithm should fall off exponentially with the number of primes dividing M, because at each step the algorithm multiplies two intermediate values to produce another intermediate that solves b more equations, for some block size b. So while it performs well for the first 100 primes, ISD appears to perform better for the first 400 primes.

B.2 Multiple u

Instead of using linear equations to search for a single u, we could choose a few small u such that at least one of them is valid for every $p|M$. For example, for each of the first 133 odd primes, at least one of $u \in U := \{1, 2, 5, 19\}$ is valid. We could factor M into $\prod_{u \in U} M_u$ such that u is valid mod the corresponding M_u. Then we could sample values $x_u \xleftarrow{\$} (\mathbb{Z}/M_u)^*$ and combine them as in Section 2.2.

B.3 Quadratic minimization

Two other techniques are based on finding small values of quadratic functions over the integers. One is to factor M as $M_1 \cdot M_3$ where M_1 contains the 1-mod-4 factors and M_3 contains the 3-mod-4 factors of M. Valid u are of the form $u \equiv x^2 \bmod M_3$ for some x coprime to M_3. We may plug in $x = \lfloor\sqrt{kM_3}\rfloor + \ell$ for small positive integers k, ℓ as a more efficient brute force technique. This technique gives many candidate values of u which are around $\sqrt{M_3} \approx \sqrt[4]{M}$, but it still takes exponential time as M increases.

The second approach is to choose small, coprime, square-free positive integers (α, β), and then partition M as $M_0 \cdot M_1$, such that

$$u = \alpha M_0 - \beta M_1$$

is valid. This will be true if:

1. For all primes $p|M$, if $p|\alpha$ then $p|M_0$ and likewise if $p|\beta$ then $p|M_1$.
2. For all other primes $p|M_0$, $\left(\frac{\beta}{p}\right) \cdot \prod_{q|M_1} \left(\frac{q}{p}\right) = -1$ and vice versa.

These equations are actually affine: switching a prime p from M_0 to M_1 or back has the same effect on all the equations regardless of where the other primes are assigned. It can therefore be solved efficiently for a given (α, β) with probability about $(1 - \frac{1}{2}) \cdot (1 - \frac{1}{4}) \cdots \approx 0.29$.

To further reduce u, we make two improvements. First, we extend the equation to $u = \alpha M_0 x^2 - \beta M_1 y^2$ where x is coprime to $\beta M_1 y^2$ and vice versa. By setting x/y as convergents to $\sqrt{\beta M_1/(\alpha M_0)}$, we can find many valid values of $u \approx \sqrt{\alpha \cdot \beta \cdot M_0 \cdot M_1}$. Furthermore, we don't need to set $M = M_0 \cdot M_1$ exactly: it suffices to instead choose $M_2|M$ upfront and set $M = M_0 \cdot M_1 \cdot M_2$. This method produces many u which are valid mod $M_0 \cdot M_1$, and we can continue until by chance we find one which is also valid mod M_2. Overall, this approach finds u which are slightly smaller than $\sqrt{M}$, as does ISD, but ISD seems to work better in practice.