Safety in Numbers: On the Need for Robust Diffie-Hellman Parameter Validation

Steven Galbraith[1], Jake Massimo[2], and Kenneth G. Paterson[2]

[1] University of Auckland
[2] Royal Holloway, University of London
s.galbraith@auckland.ac.nz, jake.massimo.2015@rhul.ac.uk,
kenny.paterson@rhul.ac.uk

Abstract. We consider the problem of constructing Diffie-Hellman (DH) parameters which pass standard approaches to parameter validation but for which the Discrete Logarithm Problem (DLP) is relatively easy to solve. We consider both the finite field setting and the elliptic curve setting.

For finite fields, we show how to construct DH parameters (p, q, g) for the safe prime setting in which $p = 2q + 1$ is prime, q is relatively smooth but fools random-base Miller-Rabin primality testing with some reasonable probability, and g is of order q mod p. The construction involves modifying and combining known methods for obtaining Carmichael numbers. Concretely, we provide an example with 1024-bit p which passes OpenSSL's Diffie-Hellman validation procedure with probability 2^{-24} (for versions of OpenSSL prior to 1.1.0i). Here, the largest factor of q has 121 bits, meaning that the DLP can be solved with about 2^{64} effort using the Pohlig-Hellman algorithm. We go on to explain how this parameter set can be used to mount offline dictionary attacks against PAKE protocols.

In the elliptic curve case, we use an algorithm of Bröker and Stevenhagen to construct an elliptic curve E over a finite field $\mathbb{F}_p$ having a specified number of points n. We are able to select n of the form $h \cdot q$ such that h is a small co-factor, q is relatively smooth but fools random-base Miller-Rabin primality testing with some reasonable probability, and E has a point of order q. Concretely, we provide example curves at the 128-bit security level with $h = 1$, where q passes a single random-base Miller-Rabin primality test with probability $1/4$ and where the elliptic curve DLP can be solved with about 2^{44} effort. Alternatively, we can pass the test with probability $1/8$ and solve the elliptic curve DLP with about $2^{35.5}$ effort. These ECDH parameter sets lead to similar attacks on PAKE protocols relying on elliptic curves.

Our work shows the importance of performing proper (EC)DH parameter validation in cryptographic implementations and/or the wisdom of relying on standardised parameter sets of known provenance.

1 Introduction

In a recent paper, Albrecht *et al.* [AMPS18] conducted a systematic study of primality testing "in the wild". They found flaws in primality tests implemented in several cryptographic libraries, for example a reliance on fixed-base Miller-Rabin primality testing, or the use of too few rounds of the Miller-Rabin test when testing numbers of unknown provenance. They studied the implications of their work for Diffie-Hellman (DH) in the finite field case, showing how to generate DH parameter sets of the form (p, q, g) in which $p = kq + 1$ for some k, p is prime, q is composite but passes a Miller-Rabin primality test with some probability, yet q is sufficiently smooth that the Discrete Logarithm Problem (DLP) is relatively easy to solve using the Pohlig-Hellman algorithm in the order q subgroup generated by g. Such a parameter set (p, q, g) might pass DH parameter validation with non-negligible probability in a cryptographic library that performs "naive" primality testing on p and q, e.g. one carrying out just a few iterations of Miller-Rabin on each number. If such a parameter set were used in a cryptographic protocol like TLS, then it would also allow an attacker to recover all the keying material and thence break the protocol's security, cf. [Won16]. Albrecht *et al.* [AMPS18] posited this as a plausible attack scenario when, for example, a malicious developer hard-codes the DH parameters into the protocol.

It is notable that the methods of [AMPS18] for producing malicious DH parameters do not work in the safe prime setting, wherein $p = 2q + 1$. This is because Albrecht *et al.* need flexibility in choosing k to arrange p to be prime. It is also because their methods can only produce q with 2 or 3 prime factors, meaning that q needs to be relatively small so that the Pohlig-Hellman algorithm applies (recall that Pohlig-Hellman runs in time $O(B^{1/2})$ where B is a bound on the largest prime factor of q; if q has only 3 prime factors and we want an algorithm requiring 2^{64} effort, then q can have at most 384 bits). Yet requiring safe primes is quite common for DH in the finite field setting. This is because it helps to avoid known attacks, such as small subgroup attacks [LL97,VAS+17], and because it ostensibly makes parameter validation easier. For example, OpenSSL's Diffie-Hellman validation routine `DH_check`[3] insists on the safe prime setting by default. Indeed, it was left as an open problem in [AMPS18] to find a large, sufficiently smooth, composite q passing a primality test with high probability such that $p = 2q + 1$ is also prime or passes a primality test.

Interestingly, more than a decade ago, Bleichenbacher [Ble05] addressed a closely related problem: the construction of malicious DH parameters (p, q, g) for which p and q pass *fixed-base* Miller-Rabin primality tests. This was motivated by his observation that, at this time, the GNU Crypto library was using such a test, with the bases being the first 13 primes $a = 2, 3, \ldots, 41$. He produced a number q having 1095 bits and 27 prime factors, the largest of which has 53

[3] See `https://www.openssl.org/docs/man1.1.1/man3/DH_check.html` for a description and `https://github.com/openssl/openssl/blob/master/crypto/dh/dh_check.c` for source code.

bits, such that q *always* passed the primality test of GNU Crypto, and such that $p = 2q + 1$ is prime. His q has very special form: it is a Carmichael number obtained using a modified version of the Erdös method [Erd56]. Of course, his DH parameter set (p, q, g) would not stand up to the more commonly used random-base Miller-Rabin testing, but his construction is nevertheless impressive. Bleichenbacher also showed how such a parameter set could be used to break the SRP Password Authenticated Key Exchange (PAKE) protocol: he showed that a client that accepts bad DH parameters in the SRP protocol can be subject to an offline dictionary attack on its password. Here, the attacker impersonates the server in a run of the SRP protocol, sending the client malicious DH parameters, and inducing the client to send a password-dependent protocol message. It is the attacker's ability to solve the DLP that then enables the offline password recovery. Thus Bleichenbacher had already given a realistic and quite standard attack scenario where robust DH (and ECDH) parameter validation is crucial: PAKE protocols in which an attacker impersonating one of the parties can dictate (EC)DH parameters.

1.1 Our Contributions

In this paper, we address the problem left open from [AMPS18] of finding malicious DH parameters in the safe prime setting. We also study the analogous problem in the elliptic curve setting.

Finite Field Setting: As a flavour of the results to come, we exhibit a DH parameter set $(p = 2q+1, q, g)$ in which p has 1024 bits and q is a composite with 9 prime factors, each at most 121 bits in size, which passes a single random-base Miller-Rabin test with probability 2^{-8}. We show that no number with this many factors can achieve a higher passing probability. Because of the 121-bit bound on the factors of q, the DLP in the subgroup of order q generated by g can be solved with about 2^{64} effort using the Pohlig-Hellman algorithm. When OpenSSL's DH validation routine `DH_check` is used in its default configuration, this parameter set is declared valid with probability 2^{-24} for versions of OpenSSL prior to 1.1.0i (released 14th August 2018). This is because OpenSSL uses the size of q to determine how many rounds of Miller-Rabin to apply, and adopts non-adversarial bounds suitable for average case primality testing derived from [DLP93]. These dictate using 3 rounds of testing for 1023-bit q for versions of OpenSSL prior to 1.1.0i, and 5 rounds in later versions (the increase was made in an effort to achieve a 128-bit security level). We also give a DH parameter set $(p = 2q+1, q, g)$ in which p is a 1024 bit prime and q has 11 prime factors, each at most 100 bits in size, which passes a single random-base Miller-Rabin test with probability 2^{-10}. This parameter set is declared valid with a lower probability of 2^{-30} for versions of OpenSSL prior to 1.1.0i, however the DLP in the subgroup of order q generated by g can be solved using the Pohlig-Hellman algorithm with less effort, in about 2^{54} operations.

The probability of 2^{-24} or 2^{-30} for passing DH validation may not seem very large, and indeed can be seen as a vindication of using safe primes for DH. On the

other hand, Bleichenbacher-style attacks against PAKEs can be carried out over many sessions and against multiple users, meaning that the success probability of an overall password recovery attack can be boosted. We exemplify this in the context of J-PAKE, a particular PAKE protocol that was supported in OpenSSL until recently (but we stress that the attack is not specific to J-PAKE).

Obtaining such malicious DH parameter sets in the finite field setting requires some new insights. In particular, we are interested in numbers q that are relatively smooth (having all prime factors less than some pre-determined bound B, say), but which pass random-base Miller-Rabin primality tests with probability as high as possible. We therefore investigate the relationship between the number of prime factors m of a number n and the number of Miller-Rabin non-witnesses $S(n)$ for n, this being the number of bases a for which the Miller-Rabin test fails to declare n composite. We are able to prove that $S(n) \leq \varphi(n)/2^{m-1}$ where $\varphi(\cdot)$ is the Euler function. Since for large n we usually have $\varphi(n) \approx n$, this shows that the highest probability a malicious actor can achieve for passing a single, random-base Miller-Rabin test is (roughly) 2^{1-m}. (This already shows that an adversary can only have limited success, especially if multiple rounds of Miller-Rabin are used.) We are also able to completely characterise those numbers achieving equality in this bound for $m \geq 3$: they are exactly the Carmichael numbers having m prime factors that are all congruent to 3 mod 4.

This characterisation then motivates us to develop constructions for such Carmichael numbers with a controlled number of prime factors. We show how to modify the existing Erdös method [Erd56] and the method of Granville and Pomerance [GP02] for constructing Carmichael numbers, and how to combine them, to obtain cryptographically-sized q with the required properties.

However, this only partly solves our problem, since we also require that $p = 2q + 1$ should pass primality tests (or even be prime). We explore further modifications of our approach so as to avoid trivial arithmetic conditions that prevent p from being prime (the prime 3 is particularly troublesome in this regard). We are also able to show that the probability that p is prime is higher than would be expected for a random choice of p by virtue of properties of the Granville-Pomerance construction: essentially, the construction ensures that p cannot be divisible by certain small primes; we tweak the construction further to enhance this property. Combining all of these steps leads to a detailed procedure by which our example DH parameter set $(p = 2q + 1, q, g)$ described above was obtained. This procedure is amenable to parallelisation. The computation of our particular example required 136 core-days of computation using a server with 3.2GHz processors.

Elliptic Curve Setting: While the main focus of our work is on the finite field setting, we also briefly study the elliptic curve setting. Here ECDH parameters (p, E, P, q, h) consist of a prime p defining a field (we focus on prime fields, $\mathbb{F}_p$), a curve E over that field defined in some standard form (for example, short Weierstrass form), a point P, the (claimed) order q of P, and a co-factor h such that $\#E(\mathbb{F}_p) = h \cdot q$. Parameter validation should verify the primality of p and q,

and check that P does have order q on E by computing $[q]P$ and comparing the result to the point at infinity.

Bröker and Stevenhagen [BS05] gave a reasonably efficient algorithm to construct an elliptic curve E over a prime field $\mathbb{F}_p$ having a specified number of points n, given the factorisation of n as an input. Their algorithm is sensitive to the number of prime factors of n – fewer is better. We use their algorithm with n being one of our specially constructed Carmichael numbers q passing Miller-Rabin primality testing with highest possible probability, or a small multiple of such a q.

Since $p \approx q$ in the elliptic curve setting and we only need these numbers to have, say, 256 bits to achieve a 128-bit security level, the task of constructing q is much easier than in the finite field setting considered above. Indeed, we could employ a Carmichael number q with 3 prime factors to pass Miller-Rabin with probability 1/4 per iteration. At the 128-bit security level, q then has 3 prime factors each of roughly 85 bits, and the Pohlig-Hellman algorithm would solve the ECDLP on the constructed curve in about 2^{44} steps. Using a Carmichael q with 4 prime factors each of exactly 64 bits, we would pass Miller-Rabin with probability 1/8 per iteration and solve the ECDLP with only 2^{34} effort. We give concrete examples of curves having such properties.

These malicious ECDH parameters (p, E, P, q, h) lead to attacks on PAKEs running over elliptic curves, as well as more traditional ECDH key exchanges. These attacks are fully analogous to those in the finite field setting. They highlight the importance of careful validation of ECDH parameters that may originate from potentially malicious sources, especially in the case of bespoke parameter sets sent as part of a cryptographic protocol. For example, the specification of the TLS extension for elliptic curve cryptography [BWBG+06] caters for the use of custom elliptic curves, though this option does not seem to be widely supported in implementations at present. Our work shows that robust checking of any such parameters would be highly advisable.

1.2 Further Related Work

In the light of the Snowden revelations, a body of work examining methods by which the security of cryptographic algorithms and protocols can be deliberately undermined has been developed. Our work can be seen as fitting into that theme (though we stress that the application of our work to PAKE protocols shows that there are concerns in the "standard" cryptographic setting too).

Young and Yung laid the foundations of kleptography, that is, cryptography designed with malicious intent, see for example [YY97]. Bellare *et al.* [BPR14] studied the problem of how to subvert symmetric encryption algorithms, and how to protect against such subversions.

Fried *et al.* [FGHT17] followed up on early work of Gordon [Gor93] to examine how to backdoor the DLP in the finite field setting. These works showed how to construct large primes p for which the Special Number Field Sieve makes solving the DLP possible if one is in possession of trapdoor information about how p was generated. This provides another avenue to subverting the security of

DH parameters. It appears that the 1024-bit example in [FGHT17] is not in the safe-prime setting, however.

The NIST DualEC generator was extensively analysed [CNE+14] and found to be used in Juniper's ScreenOS operating system in an exploitable way [CMG+16]. This inspired more theoretical follow-up work on backdoored RNGs [DGG+15] and PRNGs [DPSW16].

Bernstein *et al.* [BCC+15] extensively discuss the problem of certifying that elliptic curve parameter sets are free of manipulation during generation.

The dangers of allowing support for old algorithms and protocol versions, especially those allowing export-grade cryptography, are made manifest by the FREAK [BBD+15], Logjam [ABD+15] and DROWN [ASS+16] attacks on SSL and TLS.

2 Miller-Rabin Primality Testing and Pseudoprimes

Suppose $n > 1$ is an odd integer to be tested for primality. We first write $n = 2^e d + 1$ where d is odd. If n is prime, then for any integer a with $1 \leq a < n$, we have:

$$a^d = 1 \mod n \quad \text{or} \quad a^{2^i d} = -1 \mod n \text{ for some } 0 \leq i < e.$$

The Miller-Rabin test then consists of checking the above conditions for some value a, declaring a number to be composite if both conditions fail and to be (probably) prime if either of the two conditions hold. If one condition holds, then we say n is a *pseudoprime to base* a, or that a is a *non-witness to the compositeness of* n (since n may be composite, but a does not demonstrate this fact).

We begin by exploring the relationship between a composite number n and the number of non-witnesses this number possesses, denoted $S(n)$. Since in this work we are interested in constructing numbers n that fool the Miller-Rabin test with as high a probability as possible for random bases a, our main interest is in constructing n for which $S(n)$ is as large as possible. However, since we are also interested in solving discrete logarithm problems in subgroups of order n, we will also want n to be relatively smooth.

The following theorem can be used to calculate the exact number of non-witnesses that some composite n has:

Theorem 1 ([Mon80], Proposition 1). *Let n be an odd composite integer. Suppose that $n = 2^e \cdot d + 1$ where d is odd. Also suppose that n has prime factorisation $n = \prod_{i=1}^{m} p_i^{q_i}$ where each prime p_i can be expressed as $2^{e_i} \cdot d_i + 1$ with each d_i odd. Then:*

$$S(n) = \left(\frac{2^{\min(e_i) \cdot m} - 1}{2^m - 1} + 1 \right) \prod_{i=1}^{m} \gcd(d, d_i). \tag{1}$$

A general upper-bound on $S(n)$ is given by results of [Mon80,Rab80]:

Theorem 2 (Monier-Rabin Bound). *Let $n \neq 9$ be odd and composite. Then*

$$S(n) \leq \frac{\varphi(n)}{4}$$

where φ denotes the Euler totient function.

It is known from [Mon80] that the bound in Theorem 2 is met with equality for numbers n of the form $n = (2x+1)(4x+1)$ with $2x+1, 4x+1$ prime and x odd. It is also known that the bound is met with equality for numbers n that are Carmichael numbers with three prime factors, $n = p_1 p_2 p_3$, and where each factor p_i is congruent to 3 mod 4.

Definition 1 (Carmichael numbers). *Let n be an odd composite number. Then n is said to be a Carmichael number if $a^{n-1} = 1 \bmod n$ for all a co-prime to n.*

Note that Carmichael numbers are those for which the Fermat primality test fails to identify n as composite for all co-prime bases a.

Theorem 3 (Korselt's Criterion). *Let n be odd and composite. Then n is a Carmichael number if and only if n is square-free and for all prime divisors p of n, we have $p-1 \mid n-1$.*

For a proof of this theorem, see [Mon80]. It is also known that Carmichael numbers must have at least 3 distinct prime factors.

2.1 On the Relationship Between $S(n)$ and m, the Number of Prime Factors of n

The following result is central to our work.

Theorem 4 (Factor Bound on $S(n)$). *Let n be an odd composite integer with prime factorisation $n = \prod_{i=1}^{m} p_i^{q_i}$. Write $n = 2^e d + 1$ where d is odd and $p_i = 2^{e_i} d_i + 1$ where each d_i is odd. Then $S(n) \leq \frac{\varphi(n)}{2^{m-1}}$, where $\varphi(\cdot)$ denotes Euler's function, with equality if and only if n is square-free and, for all i, $e_i = 1$ and $d_i \mid d$.*

Proof. We have:

$$\begin{aligned}
\frac{\frac{2^{\min(e_i)\cdot m}-1}{2^m-1}+1}{2^{\min(e_i)\cdot m}} &= \frac{1}{2^m-1} + \left(\frac{1}{2^{\min(e_i)\cdot m}}\right)\left(1 - \frac{1}{2^m-1}\right) \\
&\leq \frac{1}{2^m-1} + \left(\frac{1}{2^m}\right)\left(1 - \frac{1}{2^m-1}\right) \\
&= \frac{2(2^m-1)}{(2^m)(2^m-1)} \\
&= \frac{1}{2^{m-1}}.
\end{aligned}$$

Therefore, using Theorem 1, we have:

$$S(n) = \left(\frac{2^{\min(e_i)\cdot m} - 1}{2^m - 1} + 1\right) \prod_{i=1}^{m} \gcd(d, d_i) \leq \frac{1}{2^{m-1}} \cdot 2^{\min(e_i)\cdot m} \prod_{i=1}^{m} \gcd(d, d_i) \tag{2}$$

$$\leq \frac{1}{2^{m-1}} \prod_{i=1}^{m} (2^{e_i} \cdot d_i) \tag{3}$$

$$= \frac{1}{2^{m-1}} \prod_{i=1}^{m} (p_i - 1)$$

$$\leq \frac{1}{2^{m-1}} \varphi(n). \tag{4}$$

We obtain equality in equation (2) above when $\min(e_i) = 1$ and in equation (3) when $e_1 = e_2 = \cdots = e_m$ and $\gcd(d, d_i) = d_i$ for all i (which is equivalent to $d_i \mid d$). We obtain equality in equation (4) when $\varphi(n) = \prod_{i=1}^{m}(p_i - 1)$. This occurs if and only n is square free. The result follows.

Remark: For the case $m = 2$, the bound of Theorem 4 can be strengthened to $S(n) \leq \varphi(n)/4$, that is, the Monier-Rabin bound. As mentioned above, Monier [Mon80] remarked that the bound is met in this case for numbers of the form $n = (2x + 1)(4x + 1)$ with $2x + 1, 4x + 1$ prime and x odd, see also [Nar14]. This form was exploited extensively in [AMPS18], but will be less useful in our work because we require numbers n of cryptographic size that satisfy a smaller smoothness bound. For example, we will be interested in constructing 1024-bit n in which each prime factor has at most 128 bits, meaning n will have at least 8 prime factors.

We now go on to show that, when $m \geq 3$, the bound in the above theorem is attained if and only if n is a Carmichael number of special form.

Theorem 5. *Let n be a Carmichael number with $m \geq 3$ prime factors that are all congruent to 3 mod 4. Then $S(n) = \frac{\varphi(n)}{2^{m-1}}$. Conversely, if n has $m \geq 3$ prime factors and $S(n) = \frac{\varphi(n)}{2^{m-1}}$, then n is a Carmichael number whose prime factors are all congruent to 3 mod 4.*

Proof. By Korselt's criterion we know that n is square-free. Write $n = p_1 \cdots p_m$ where the p_i are prime and, by assumption, $p_i = 3 \bmod 4$ for each i. As before, we write $n = 2^e d + 1$ where d is odd and $p_i = 2^{e_i} d_i + 1$ where each d_i is odd. Since $p_i = 3 \bmod 4$ for each i, it is immediate that $e_i = 1$ for each i. Moreover, by Korselt's criterion, we have $2^{e_i} d_i | 2^e d$, and hence $d_i | d$, for each i. The result follows from the converse part of Theorem 4.

For the converse, let $n = \prod_{i=1}^{m} p_i^{q_i}$. Suppose $p_i = 2^{e_i} d_i + 1$ where d_i is odd and $n = 2^e d + 1$ where d is odd. Necessarily, $e \geq 1$. By Theorem 4, since $S(n) = \frac{\varphi(n)}{2^{m-1}}$,

we have that n is square free, $e_i = 1$ for all i and $d_i \mid d$ for all i. Since $e_i = 1$ $\forall i$, we have that $p_i = 3 \bmod 4$ and $2^{e_i} \mid 2^e$ for all i. Also, since $d_i \mid d$ for all i, it follows that $2_i^e d_i \mid 2^e d$ for all i, and thus $p_i - 1 \mid n - 1$ for all i. Hence, n satisfies Korselt's criterion, and n is a Carmichael number.

Example 1. Table 1 gives, for each $3 \leq m \leq 10$, the smallest number with m prime factors achieving the bound of Theorem 4. In the light of Theorem 5, these are all Carmichael numbers whose prime factors are all congruent to 3 mod 4. These are obtained from data made available by Pinch and reported in [Pin08]. Of course, these examples are all much too small for cryptographic use.

m	C_m	$S(C_m)$
3	$7 \cdot 19 \cdot 67$	$\varphi(C_m)/4$
4	$7 \cdot 19 \cdot 67 \cdot 199$	$\varphi(C_m)/8$
5	$7 \cdot 11 \cdot 19 \cdot 103 \cdot 9419$	$\varphi(C_m)/16$
6	$7 \cdot 11 \cdot 31 \cdot 47 \cdot 163 \cdot 223$	$\varphi(C_m)/32$
7	$19 \cdot 23 \cdot 31 \cdot 67 \cdot 71 \cdot 199 \cdot 271$	$\varphi(C_m)/64$
8	$11 \cdot 31 \cdot 43 \cdot 47 \cdot 71 \cdot 139 \cdot 239 \cdot 271$	$\varphi(C_m)/128$
9	$19 \cdot 31 \cdot 43 \cdot 67 \cdot 71 \cdot 103 \cdot 239 \cdot 307 \cdot 631$	$\varphi(C_m)/256$
10	$7 \cdot 11 \cdot 19 \cdot 31 \cdot 47 \cdot 79 \cdot 139 \cdot 163 \cdot 271 \cdot 2347$	$\varphi(C_m)/512$

Table 1. The smallest number C_m with m prime factors that meets the upper bound of $\varphi(C_m)/2^{m-1}$ on $S(C_m)$.

3 Generating Large Carmichael Numbers

The results in the previous section motivate the search for cryptographically-sized Carmichael numbers with a chosen number of prime factors, with each factor congruent to 3 mod 4. In this section, we discuss two existing constructions for Carmichael numbers: the Erdös method [Erd56] and the method of Granville and Pomerance [GP02]. We show how to combine these two methods to produce large examples. We also show how to modify the constructions to improve the probability that they will succeed in constructing large examples meeting our additional congruence requirements.

3.1 The Erdös Method

Erdös [Erd56] gave a method to construct Carmichael numbers with many prime factors. The method starts with a highly composite number L and then considers the set $\mathcal{P}(L) = \{p : p \text{ prime}, p - 1 \mid L, p \nmid L\}$. If for some subset $p_1, p_2, \ldots, p_m$ of $\mathcal{P}(L)$, we have $p_1 p_2 \cdots p_m = 1 \bmod L$, then $n = p_1 p_2 \cdots p_m$ is a Carmichael number, by Korselt's criterion. This is easy to see: by construction, $p_i - 1 \mid L$; the condition $n = 1 \bmod L$ implies that $L \mid n - 1$; it follows that $p_i - 1 \mid n - 1$, and n is evidently square-free.

Example 2. If $L = 120 = 2^3 \cdot 3 \cdot 5$, then $\mathcal{P}(L) = \{7, 11, 13, 31, 41, 61\}$. If we examine all the subsets of $\mathcal{P}(L)$, we find that $41040 = 7 \cdot 11 \cdot 13 \cdot 41$, $172081 = 7 \cdot 13 \cdot 31 \cdot 61$ and $852841 = 11 \cdot 31 \cdot 41 \cdot 61$ are all 1 mod 120, and so are all Carmichael numbers.

The Erdös method lends itself to a computational approach to generating Carmichael numbers with a chosen number of prime factors m for moderate values of L. For a given L, the set $\mathcal{P}(L)$ can be quickly generated by considering each factor f of the selected L and testing the primality of $f + 1$. One can then examine all m-products of distinct elements from $\mathcal{P}(L)$ and test the product n against the condition $n = 1 \bmod L$.

Alternatively, as pointed out in [Ble05], one can employ a time-memory trade-off (TMTO): for some k, build a table of all k-products $p_1 \cdots p_k$ from $\mathcal{P}(L)$, and look for collisions in that table with the inverses of $(m - k)$-products $(p_{k+1} \cdots p_m)^{-1} \bmod L$ from $\mathcal{P}(L)$. Such a collision gives an equation

$$p_1 \cdots p_k = (p_{k+1} \cdots p_m)^{-1} \bmod L$$

and hence

$$p_1 \cdots p_k p_{k+1} \cdots p_m = 1 \bmod L.$$

Of course, one needs to take care to avoid repeated primes in such an approach. For the L we use later, the direct approach suffices, and so we did not explore this direction further.

3.2 The Selection of L in the Erdös Method

Clearly, L must be even, otherwise the integers p satisfying $p - 1 \mid L$ will all be even. We can ensure that all primes p in $\mathcal{P}(L)$ satisfy $p = 3 \bmod 4$ by setting the maximum power of 2 in L to be 1, i.e. by setting $L = 2 \bmod 4$. For then each factor f of L must be 2 mod 4, and hence $p = f + 1 = 3 \bmod 4$. As we shall see later, other conditions can be imposed on L as needed.

Note that since $2 \mid L$, $p = 3$ is a candidate for inclusion in $\mathcal{P}(L)$. However, if 3 is also a factor of L then it is excluded because of the additional condition $p \nmid L$ on elements of $\mathcal{P}(L)$; this condition is needed in general, since if $p \mid L$, then any product $p_1 p_2 \cdots p_m$ including p as a factor would be 0 mod L instead of the required 1 mod L.

For the Erdös method to be successful in producing a Carmichael number with m prime factors, we need to find a product p_i such that $p_1 p_2 \cdots p_m = 1 \bmod L$. One can see that the number of possible products that can be considered is $\binom{|\mathcal{P}(L)|}{m}$. Let us make the heuristic assumption that the values of $p_1 p_2 \cdots p_m$ are uniformly distributed amongst the odd numbers modulo the even integer L. Then we need to ensure that:

$$\binom{|\mathcal{P}(L)|}{m} \gtrapprox L/2$$

for the method to have a reasonable chance of success.

Thus it is desirable to find L such that $|\mathcal{P}(L)|$ is as large as possible. In turn, this heuristically depends on L being as smooth as possible, since such an L

L_{bound}	L_{best}	$\lvert\mathcal{P}(L_{best})\rvert$
2^{20}	$810810 = 2 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	39
2^{21}	$2088450 = 2 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 13 \cdot 17$	50
2^{22}	$4054050 = 2 \cdot 3^4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	58
2^{23}	$7657650 = 2 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	65
2^{24}	$13783770 = 2 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	73
2^{25}	$22972950 = 2 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	89
2^{26}	$53603550 = 2 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17$	93

Table 2. For a given L_{bound} (column 1), the value L_{best} (column 2) gives the value of $L \leq L_{bound}$ resulting in the largest set of primes $\mathcal{P}(L)$, subject to the additional restriction that $p = 3 \bmod 4$ for all $p \in \mathcal{P}(L)$.

has many factors f and therefore many possible candidates $p = f + 1$ that, if prime, can be included in $\mathcal{P}(L)$. This analysis of course depends on the primality of the different values $f + 1$ being in some sense independent for the different factors f of L; this is a reasonable assumption given standard heuristics on the distribution of primes.

For various bounds L_{bound}, we have computed the value of $L \leq L_{bound}$ giving the largest set $\mathcal{P}(L)$, where we impose the restriction $L = 2 \bmod 4$ to ensure the primes in $\mathcal{P}(L)$ are all 3 mod 4. The results are shown in Table 2, and bear out our heuristic analysis suggesting that smooth L make the best choices.

Example 3. Suppose $L = 53603550$. Then $|\mathcal{P}(L)| = 93$ with:

$$\begin{aligned}\mathcal{P}(L) = \{&19, 23, 31, 43, 67, 71, 79, 103, 127, 131, 151, 199, 211, 239, 307, 331, 443,\\ &463, 491, 547, 631, 859, 883, 911, 991, 1051, 1123, 1171, 1327, 1471, 1531,\\ &1667, 1871, 1951, 2003, 2143, 2311, 2551, 2731, 3571, 3823, 3851, 4951,\\ &4999, 5851, 6007, 7151, 7351, 8191, 9283, 10711, 11467, 11551, 16831,\\ &17851, 19891, 22051, 23563, 26951, 27847, 28051, 33151, 34651, 41651,\\ &42043, 43759, 46411, 50051, 53551, 54979, 57331, 72931, 77351, 91631\\ &102103, 117811, 124951, 126127, 150151, 232051, 242551, 286651,\\ &324871, 350351, 450451, 824671, 1051051, 1093951, 1191191, 1624351,\\ &2144143, 4873051, 10720711\}.\end{aligned}$$

As representative examples, the following Carmichael numbers with, respectively 8 and 16 prime factors, can then be obtained by running a simple search algorithm over subsets of $\mathcal{P}(L)$ to find subsets whose products are 1 mod L:

$$\begin{aligned}C_8 =& 19 \cdot 23 \cdot 43 \cdot 239 \cdot 859 \cdot 9283 \cdot 11467 \cdot 242551\\ C_{16} =& 19 \cdot 23 \cdot 31 \cdot 43 \cdot 67 \cdot 71 \cdot 79 \cdot 103 \cdot 127 \cdot\\ & 131 \cdot 491 \cdot 1531 \cdot 3851 \cdot 7151 \cdot 11467 \cdot 33151\end{aligned}$$

Here

$$C_8 = 99605240811373000403701$$

and

$$C_{16} = 29520757403834736752314039155478508748 01.$$

Our SAGE [S+18] implementation of the Erdös method running on a 3.3GHz processor took 4.83 seconds to find C_8 and 1.78 seconds to find C_{16}. The code used to generate these examples can be found in Appendix A.

It would be tempting to think that this method could easily be scaled-up to numbers of cryptographic size. However, this is not so easy. To illustrate, suppose we wanted to construct a 1024-bit n with, say, $m = 8$ prime factors, all having about 128 bits. This would necessitate using an L substantially larger than 2^{128}, which would make the direct approach of finding a product $p_1 \cdots p_8 = 1 \bmod L$ infeasible; even the TMTO version would require prohibitive time and memory, on the order of 2^{64} of each.

3.3 The Method of Granville and Pomerance

The second method of generating Carmichael numbers that we consider is due to Granville and Pomerance [GP02]. This takes a small Carmichael number with m (known) factors and produces from it a larger Carmichael number, also with m factors. It is based on the following theorem.

Theorem 6 (Granville and Pomerance [GP02]). *Let $n = p_1 p_2 \cdots p_m$ be a Carmichael Number. Let $L = \operatorname{lcm}(p_i - 1)$ and let M be any integer with $M \equiv 1 \bmod L$. Set $q_i = 1 + M(p_i - 1)$. Then $N = q_1 \cdots q_m$ is a Carmichael number whenever each q_i is prime.*

Recall that we are interested in Carmichael numbers N in which all prime factors are congruent to 3 mod 4. Fortunately, as the following lemma shows, the method of Granville and Pomerance 'preserves' this property.

Lemma 1. *With notation as in Theorem 6, suppose $p_i \equiv 3 \pmod 4$. Then $q_i = 3 \pmod 4$.*

Proof. The integer L is even as it is the least common multiple of even integers $p_i - 1$. But $M \equiv 1 \pmod L$ implies that M is odd; write $M = 2s + 1$. Moreover, since $p_i = 3 \bmod 4$, we have $p_i - 1 = 2d_i$ with d_i odd; write $d_i = 2t_i + 1$. Then $q_i = 1 + M(p_i - 1) = 1 + (2s+1)(4t_i + 2) = 3 + 4(2st_i + s + t_i)$, which is evidently 3 mod 4.

There are two important choices of variable in this method: M and the starting Carmichael number n.

Clearly, the properties of the resulting Carmichael number N are dependent on n, for example the value of each prime factor mod 4 (as seen in Lemma 1) and the number m of these factors.

The effects of M are more subtle. In particular, we need to select an M such that all the resulting $q_i = 1 + M(p_i - 1)$ are prime. Using the heuristic that the values q_i are as likely to be prime as random choices of odd q_i of the same size,

the probability that a random choice of M yields m primes is approximately $(2/\ln(B))^m$ where B is a bound on the q_i. This probability drops very quickly for N of cryptographic size and even moderate m. For example, with B of 128 bits and $m = 8$ (so that the target N has 1024 bits), we obtain $(2/\ln(B))^m \approx 2^{-43.77}$. Clearly then, simply making random choices of M is unlikely to yield candidates of cryptographically interesting sizes in a reasonable amount of time. We therefore turn to investigating methods for improving the probability that the q_i are all prime by careful choice of M.

3.4 The Selection of M in the Method of Granville and Pomerance

The only requirement on M coming from Theorem 6 is that $M \equiv 1 \pmod{L}$, where $L = \text{lcm}(p_i - 1)$. However, by a careful choice of M we can both ensure that this is true, and that the resulting values $q_i = 1 + M(p_i - 1)$ are more likely to be prime than if M was chosen at random.

Our approach is inspired by techniques originally introduced in [JPV00,JP06] for generating primes on low-end processors. There, one considers numbers of the form $p = kH + \delta$ where H is smooth (say, H is the product of the first h primes, $H = \prod_{i=1}^{h} s_i$), δ is chosen to be co-prime to H, and k is a free parameter. Then p is guaranteed to be divisible by each of $s_1, \ldots, s_h$, since $p = \delta \neq 0 \bmod s_i$. By choosing different values of k, one can generate different candidates for p, and test them for primality. Numbers p generated in this way have a higher probability of being prime than uniformly random candidates, since they are effectively guaranteed to pass trial divisions by each of the small primes dividing H. We refer to this process as 'sieving' by the primes $s_1, s_2, \ldots, s_h$. An analysis using the inclusion-exclusion principle can be used to evaluate the increase in probability that can be achieved by this means; a factor of 5 increase is typical even for moderate values of h, since many small divisors can be eliminated.

We present an adaptation of this method to generate candidates for M in the method of Granville and Pomerance, such that the resulting q_i are guaranteed to be indivisible by many small primes.

Since $M = 1 \pmod{L}$, we can write $M = kL + 1$, where k now becomes the free parameter in the construction method. Then

$$q_i - 1 = M(p_i - 1) = (kL + 1)(p_i - 1) = kLp_i + p_i - kL - 1.$$

Rearranging, we get:

$$q_i = kLp_i + p_i - kL = kL(p_i - 1) + p_i.$$

Note that, typically, many small primes will divide L because L is the least common multiple of the $p_i - 1$. This is especially so if we use the Erdös method to generate the starting Carmichael number n, since it starts with a smooth number which all the $p_i - 1$ will divide.

Now none of the primes dividing L can be a p_i (again, because L is the least common multiple of the $p_i - 1$). For each such prime p, we have:

$$q_i = p_i \neq 0 \bmod p.$$

Hence, we are assured that q_i is not divisible by any of the prime divisors of L: we achieve 'free' sieving on q_i for every such divisor.

Now we consider other primes (not equal to any of the p_i, and not dividing L). Let s denote such a prime, and suppose we choose k such that s divides k. Recalling that $M = kL + 1$, then we get:

$$q_i = kL(p_i - 1) + p_i = p_i \neq 0 \bmod s.$$

Hence, by choosing k so that it is divisible by a product of primes s_j that do not equal any of the p_i nor any of the divisors of L, we also obtain sieving on all the s_j. Of course, we can include an additional factor when building k to ensure that the resulting q_i are of any desired bit-size and that there are sufficiently many choices for k (and thence M). In what follows, we write $k = k' \prod_j s_j$ for some collection of primes s_j subject to the above constraints; k' now replaces k as the free parameter in the construction.

The overall sieving effectiveness will be determined by the collection of prime factors present in L and the s_j. Let us denote the complete set of primes from these two sources as $\{s_1, \ldots, s_h\}$. Then the fraction of non-prime candidates for each q_i that are removed by the sieving can be calculated using the formula:

$$\sigma = 1 - \prod_{i=1}^{h} \left(1 - \frac{1}{s_i}\right). \tag{5}$$

This means that the prime values of q_i are now concentrated in a fraction $1 - \sigma$ of the initial set of candidates, so that a random selection from this reduced set is $1/(1 - \sigma)$ times more likely to result in a prime. Notice that the effect here is multiplicative across all m of the q_i – they all benefit from the sieving on the s_i. Note too how powerful the prime $s = 3$ is in sieving, contributing a factor $2/3$ to the product term determining σ.

The overall effect is to improve the success probability for each trial of the modified Granville-Pomerance construction (involving a choice of k') from $(2/\ln(B))^m$ to $(2/(1 - \sigma)\ln(B))^m$.

Example 4. Using a C implementation of the modified Granville-Pomerance construction, with the Carmichael number C_8 of Example 3 as the starting value n and $L = 53603550$, we found that choosing

$$k = 7891867750444302551322686487$$

produces the 8-factor, 1024-bit Carmichael number $N = q_1 \cdots q_8$ where:

$$
\begin{aligned}
q_1 &= 76145782959779164924491574423241193 19\\
q_2 &= 93067068061952312685489702072850347 23\\
q_3 &= 17767349357281805149048034032089611743\\
q_4 &= 10068164635793022917793885951517446 6539\\
q_5 &= 36296156544161401947340983808411635 4159\\
q_6 &= 39265842079592789379396155210918041 94983\\
q_7 &= 48504863745379328056901132907604640 05567\\
q_8 &= 10260644253830242473575239653531750 7810051.
\end{aligned}
$$

Here, q_8, the largest prime factor, has 137 bits.

As pointed out in Section 3.3, with B of 128 bits and $m = 8$ (so that the target N has 1024 bits), we estimate the standard Granville-Pomerance construction to have a success rate of $(2/\ln(B))^m \approx 2^{-43.8}$ per trial, so that the expected number of trials would be about $2^{43.8}$. With our modified version of the Granville-Pomerance construction we obtain sieving on each of the q_i by the primes $3, 5, 7, 11, 13, 17$ that divide L (in this case, we did not add any more primes to k to improve the sieving further). This gives us $\sigma = 0.6393$ and therefore reduces the expected number of trials by a factor of about $1/(1-\sigma)^m \approx 2^{11.8}$ to roughly 2^{32} trials. Finding the above N using our 'C' implementation actually took $2^{31.51}$ trials and less than one core-hour running on 3.3GHz CPUs.

The above example illustrates that we can generate numbers that are of cryptographically interesting size, have a controlled number of prime factors (and therefore achieve a given smoothness bound), achieve the upper bound of Theorem 4 on the number of Miller-Rabin non-witnesses, and hence maximise the probability of passing random-base Miller-Rabin primality tests.

4 Fooling Diffie-Hellman Parameter Validation in the Safe-Prime Setting

In this section, we target the problem of producing Diffie-Hellman parameters for the prime order setting, where the parameters are able to pass validity tests on the parameters but where the relevant Discrete Logarithm Problem (DLP) is relatively easy.

A Diffie-Hellman (DH) parameter set (p, q, g) in the prime order setting is formed of a prime p with $g \in \mathbb{Z}_p$ generating a group of prime order q, where $q \mid p - 1$. As explained in the introduction, validating the correctness of DH parameters is vital in ensuring the subsequent security of the DH key exchange. As also explained there, Bleichenbacher [Ble05] provided an extreme example of this in the context of Password Authenticated Key Exchange (PAKE): he showed that a client that accepts bad DH parameters in the SRP protocol can be subject to an offline dictionary attack on its password. Here, the attacker

impersonates the server in a run of the SRP protocol, and induces the client to send a password-dependent protocol message; the attacker's ability to solve the DLP is what enables the offline password recovery.

DH validation checks should consist of primality tests on both p and q as well as a verification that $p = kq + 1$ for some integer k. The checks should also ensure that the given generator g generates the subgroup of order q. The security is based in part on size of q: it must still be large enough to thwart the Pohlig-Hellman algorithm for solving the DLP. For prime q, this algorithm runs in time $O(\sqrt{q})$.

Albrecht *et al.* [AMPS18] already showed how to subvert DH parameters in the case where k is permitted to be large and where a weak primality test based on Miller-Rabin with a small number of rounds is permitted. For example, they selected q to be of the form $(2x + 1)(4x + 1)$ with both factors prime, and then tried k of a suitable size until $kq + 1$ was prime. This gives an $O(q^{1/4})$ algorithm using the Pohlig-Hellman algorithm in the subgroups of orders $2x + 1$ and $4x + 1$, with q passing t rounds of random-base Miller-Rabin testing with the best possible probability 4^{-t} (this coming from the Monier-Rabin bound).

However, many implementations insist on using DH parameters in which p is a safe prime; that is, they require $p = 2q + 1$, in which case g must have order q or $2q$ if it is not equal to ± 1. OpenSSL in its default setting is a good example of such a library. Insisting on safe primes to a large extent eliminates small subgroup attacks. It is also a good option in the context of protocols like SSL/TLS in which a server following the specification only provides p and g but not q.[4] As noted in the introduction, the techniques of [AMPS18] do not extend to the safe-prime setting, since they need the flexibility in k to force $p = kq + 1$ to be prime. The resulting q would also be too large and have too few prime factors to make the Pohlig-Hellman algorithm effective.

This leaves open the problem of fooling DH parameter validation when random-base Miller-Rabin tests are used for checking p and q (as should be the case in practice, in light of the work of [Arn95] and [Ble05]).

4.1 Generating Carmichael Numbers q such that $p = 2q + 1$ is Prime

To summarise the above discussion, we wish to construct a number q such that q and $p = 2q + 1$ both pass random-base Miller-Rabin primality testing, and such that q is sufficiently smooth that the Pohlig-Hellman algorithm can be used to solve the DLP in some subgroup mod p.

Our approach parallels that of [Ble05]: we construct q as a large Carmichael number with m prime factors that are all 3 mod 4 using the techniques from the previous section. Then q will pass random-base Miller-Rabin primality tests with

[4] For if p is not a safe prime, then the client is forced to blindly accept the parameters or to do an expensive computation to factorise $p - 1$ and then test g for different possible orders arising as factors of $p - 1$. We know of no cryptographic library that does the latter.

the highest possible probability amongst all integers with m prime factors. After constructing a candidate q, we test $2q+1$ for primality (using a robust primality test), rejecting q if this test fails, and stopping if it passes. If $2q+1$ is prime, then the DLP in the subgroup of order q can be solved with $O(mB^{1/2})$ effort where B is an upper bound on the prime factors of q.

The approach just described will fail in practice. The first reason is that it is unlikely that $2q+1$ will happen to be prime by chance (the probability is about $1/\ln q$ by standard density estimates for primes). The second reason is that there may be arithmetic reasons why $2q+1$ can *never* be prime. We investigate and resolve these issues next.

Sieving for $2q+1$: We begin by examining the method of Granville and Pomerance and its consequences for the values of $2q+1$ modulo small primes.

Assume we have some starting Carmichael number $n = p_1 \cdots p_m$, and we apply the method of Granville and Pomerance, setting $q_i = M(p_i - 1) + 1$ where $M = 1 + kL$ and $L = \operatorname{lcm}(p_i - 1)$. We assume k is such that the q_i are all prime, and we write $q = q_1 \cdots q_m$ for the resulting Carmichael number.

Lemma 2. *With notation as above, for all primes s dividing kL, we have that $2q+1 = 2n+1 \pmod{s}$.*

Proof. Since $q_i = M(p_i-1)+1 = (1+kL)(p_i-1)+1$, it follows that for any prime s with $s \mid kL$ we have $q_i = p_i \pmod{s}$, therefore $2q+1 \equiv 2n+1 \pmod{s}$.

The importance of the above lemma is that we can determine at the outset, based only on the small starting Carmichael number n, whether $2q+1$ will be divisible by each of the primes s or not. In particular, we should just ignore any n for which $2n+1 \equiv 0 \pmod{s}$ for any of the primes s dividing L or k, since then $2q+1$ can never be prime. Typically, there are many such primes s, since L is usually rather smooth, arising as the least common multiple of the $p_i - 1$. This is particularly so when the Erdös method is used to construct n.

The Prime 3: The prime 3 plays a particularly important role when applying our sieving trick in the method of Granville and Pomerance: it contributes a factor $2/3$ to the product term $\prod_{i=1}^{h}\left(1-\frac{1}{s_i}\right)$ when computing σ. It is therefore desirable to keep 3 as a factor of kL in the construction. On the other hand, the above lemma then imposes the necessary condition $2n+1 \neq 0 \bmod 3$ for $2q+1$ to be prime; this in turn requires $n = 0 \bmod 3$ or $n = 2 \bmod 3$.

We consider the two cases $n = 0 \bmod 3$ and $n = 2 \bmod 3$.

The case $n = 0 \bmod 3$: In this case, we have $3 \mid n$, and so we can set $p_1 = 3$. Recall that, in our approach, $n = p_1 \cdots p_m$ will be obtained using the Erdös method, in which case $p_1 = 3$ is contained in the set $\mathcal{P}(L^*)$ (henceforth L^* denotes the smooth number used in the Erdös method; we use L^* to distinguish it from $L = \operatorname{lcm}(p_i - 1)$ in the method of Granville and Pomerance – they are often

equal but need not be so). From the conditions on $\mathcal{P}(L^*)$, we deduce that $3 \nmid L^*$. Since each prime in $\mathcal{P}(L^*)$ is constructed by adding 1 to a factor of L^*, we deduce that $p = 2 \bmod 3$ for every $p \in \mathcal{P}(L^*) \setminus \{3\}$. Since we will also have $p = 3 \bmod 4$ by choice of L^*, we deduce that $p = 11 \bmod 12$ for every $p \in \mathcal{P}(L^*) \setminus \{3\}$.

Hence, in the case where 3 appears as a factor in the starting Carmichael number n, and n is obtained via the Erdös method, then the remaining primes arising as factors of n must all be 11 mod 12. This happens automatically in the Erdös method simply by ensuring $3 \nmid L^*$.

The case $n = 2 \bmod 3$*:* In this case, we can show that $p_i = 2 \bmod 3$ for all primes p_i arising as factors of n. For suppose that $p_i = 1 \bmod 3$ for some i. This implies $3 \mid p_i - 1$. By Korselt's criterion, we deduce that $3 \mid n - 1$, and hence $n = 1 \bmod 3$. This contradicts our starting assumption on n.

Moreover, it is easy to see that we must take m, the number of prime factors of n, to be odd in this case. For $n = \prod_{i=1}^{m} p_i = 2^m \bmod 3$, and so $n = 2 \bmod 3$ if and only if m is odd.

Hence, in the case where $n = 2 \bmod 3$, we are forced to use a starting Carmichael number with m odd in which $p_i = 2 \bmod 3$ for each prime factor p_i (whether or not we use the Erdös method). This may sound overly restrictive. But, fortunately, we have already seen how to arrange this for the Erdös method: we simply need to ensure that $3 \nmid L^*$, where L^* denotes the smooth number used in that construction, and then all but one of the primes $p \in \mathcal{P}(L^*)$ will satisfy this requirement. We then remove $p = 3$ from $\mathcal{P}(L^*)$ when running the last step in the Erdös method.

Other Primes: Of course, Lemma 2 imposes a single condition on n for every other prime s dividing kL, but these conditions are much less restrictive than that in the case $s = 3$, and so we do not investigate the implications for the p_i any further here.

Completing the Construction: We have now assembled all the tools necessary to produce a suitable Carmichael number n such that when the method of Granville and Pomerance is applied to produce q from n, then $2q + 1 \neq 0 \bmod 3$; moreover q will attain the bound of Theorem 4 on $S(q)$, the number of Miller-Rabin non-witnesses for q, namely $S(q) = \varphi(q)/2^{m-1}$. Our procedure is as follows:

1. We use the first step of the Erdös method with an L^* such that $2 \mid L^*$, $4 \nmid L^*$, $3 \nmid L^*$. This ensures that the resulting set $\mathcal{P}(L^*)$ contains the prime 3, and a collection of other primes that are all 11 mod 12.[5]

[5] Of course, one could choose not to restrict L^* in this way and just filter the resulting set $\mathcal{P}(L^*)$ for primes that are 11 mod 12, but this involves wasted computation and the use of larger L^* than is necessary.

2. We remove 3 from $\mathcal{P}(L^*)$ and run the second step of the Erdös method with an odd m to find a subset of primes $p_1, \ldots, p_m$ such that $n := p_1 \cdots p_m = 1 \pmod{L}$; n is then a Carmichael number with m prime factors that are all 11 mod 12 and therefore both 3 mod 4 and 2 mod 3.
3. We set $L = \mathrm{lcm}(p_i - 1)$ and test the condition $2n + 1 \neq 0 \bmod s$ for each prime factor s of L (cf. Lemma 2). If any test fails, we go back to the previous step and generate another n.
4. Integer n is then used in the method of Granville and Pomerance to produce candidates for q (in which the q_i are all prime). By construction of the p_i, we will have $3 \nmid L$ in the Granville-Pomerance method, but we desire $3 \mid kL$ in view of the power of sieving by 3 in that method. We therefore set $k = 3k'$ for k' of suitable size when running this step, introducing the prime 3 in k.
5. Finally, we test $2q + 1$ for primality. By choice of n, we are guaranteed that $2q + 1 \neq 0 \bmod 3$ and $2q + 1 \neq 0 \bmod s$ for each prime divisor s of L, so we are assured that $2q + 1$ will not be divisible by certain (small) primes.

Note that the procedure as described focusses on the case $n = 2 \bmod 3$. An alternative procedure could be developed for the case $n = 0 \bmod 3$. The procedure can be enhanced by setting k at step 4 to contain additional prime factors s beyond 3 not already found in L, to increase the effect of sieving. Of course, in view of Lemma 2, certain bad choices of s should be avoided at this stage.

4.2 Examples of Cryptographic Size

Using the method described above, we now give two examples of Carmichael numbers q such that $p = 2q + 1$ is a 1024-bit prime. In the first example q is the product of 9 prime factors, which by construction will pass a random-base Miller-Rabin primality test with probability approximately $1/2^8$. Since the largest factor of q is 121 bits in size, the DLP in the subgroup of order q mod p for this parameter set can be solved in approximately $9 \cdot 2^{60.5} \approx 2^{64}$ operations. In the second example, q is the product of 11 prime factors, which by construction will pass a random-base Miller-Rabin primality test with probability approximately $1/2^{10}$. However, because the q with 11 factors is smoother, with largest factor 100 bits in size, the DLP in the subgroup of order q mod p for this parameter set can be solved in approximately $11 \cdot 2^{50} \approx 2^{54}$ operations. We give both these examples to illustrate the trade off between the probability of a parameter set being accepted and the work required to solve the DLP for that parameter set.

Example 5. Using SAGE [S+18] we examined all $L^* < 2^{30}$ such that $2 \mid L^*$, $4 \nmid L^*$, $3 \nmid L^*$. We found the largest set of primes $\mathcal{P}(L^*)$ was produced when $L = 565815250 = 2 \cdot 5^3 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$. Here, $|\mathcal{P}(L^*)| = 53$ (including the prime 3).

Then, using the Erdös method with $L^* = 565815250$ we generated the 9-factor Carmichael number

$$\begin{aligned} n &= 171296939496088794253492158757225 1 \\ &= 71 \cdot 131 \cdot 647 \cdot 1871 \cdot 4523 \cdot 4751 \cdot 46751 \cdot 350351 \cdot 432251. \end{aligned}$$

Using the procedure described above, we found $k = 3k'$ with

$$k' = 1844674409176776955124$$

produced a 9-factor, 1023-bit Carmichael number q such that $n = 2q + 1$ is a 1024-bit prime.

To generate a target q with 1023 bits, with $m = 9$ factors each around 114 bits in size, we estimate the standard Granville-Pomerance construction to have a success rate of $(2/\ln(B))^m \approx 2^{-47.73}$ per trial, so that the expected number of trials would be about $2^{47.7}$. With our modified version of the Granville-Pomerance construction we obtain sieving on each of the q_i by the primes $5, 7, 11, 13, 17, 19$ that divide L and the prime 3 since it divides k. This gives us $\sigma = 0.658$ and therefore reduces the expected number of trials by about $1/(1-\sigma)^m \approx 2^{13.9}$ to roughly $2^{33.8}$ trials. We then need to consider the probability that the q produced is such that $p = 2q + 1$ is also prime. By Lemma 2 we know that we obtain sieving on $2q + 1$ from all primes $s \mid kL$, hence a success rate of $(2/(1-\sigma)\ln(2^{1024})) \approx 2^{-6.9}$. Therefore we expect to require $2^{33.8+6.9} = 2^{40.7}$ total trials. Finding the above q such that $p = 2q+1$ is prime actually took $2^{38.15}$ trials, so we were somewhat lucky. Our implementation is in 'C' and ran for 136 core-days on 3.2GHz CPUs.

The factors of this q are:

$$\begin{aligned} q_1 &= 219186431519361672882122216610071 \\ q_2 &= 407060515678814535352512687990131 \\ q_3 &= 202277639450109152597870741858647 \\ q_4 &= 585540895630294754699383635801187 1 \\ q_5 &= 141594434761507640681850951930105 23 \\ q_6 &= 148733649959566849455725789842547 51 \\ q_7 &= 146385223907573688674845908950296751 \\ q_8 &= 109702808975440517277502169413340035 1 \\ q_9 &= 135347621463205833004710468756718225 1. \end{aligned}$$

Since $2^q \equiv 1 \pmod{p}$ we can set a generator $g = 2$ to obtain a complete set of DH parameters (p, q, g). By construction q will pass a random-base Miller-Rabin primality test with probability approximately $1/2^8$. Since q_9, the largest factor of q, is 121 bits in size, the DLP in the subgroup of order q mod p for this parameter set can be solved in approximately $9 \cdot 2^{60.5} \approx 2^{64}$ operations. The C code used to generate this example can be found in Appendix B.

Example 6. Again, using the Erdös method with $L^* = 565815250$ we generated the 11-factor Carmichael number

$$\begin{aligned} n &= 966475945911454012761317536092647 51 \\ &= 23 \cdot 71 \cdot 191 \cdot 419 \cdot 491 \cdot 3851 \cdot 4523 \cdot 4751 \cdot 9311 \cdot 17291 \cdot 113051. \end{aligned}$$

Using the procedure described above, we found $k = 3k'$ with

$$k' = 3994916512074331$$

produced a 11-factor, 1023-bit Carmichael number q such that $p = 2q + 1$ is a 1024-bit prime.

To generate a target q with 1023 bits, with $m = 11$ factors each around 93 bits in size, we estimate the standard Granville-Pomerance construction to have a success rate of $(2/\ln(B))^m \approx 2^{-55.11}$ per trial, so that the expected number of trials would be about $2^{55.1}$. Again, using our modified version of the Granville-Pomerance construction we sieve as in the previous example to reduce the expected number of trials by about $1/(1 - 0.658)^m \approx 2^{17}$ to roughly $2^{38.1}$ trials. Then again by considering the probability that the q produced is such that $2q + 1$ is also prime we expect to require $2^{38.1+6.9} = 2^{45}$ total trials. Finding the above q such that $2q + 1$ was prime took $2^{44.83}$ trials. The computation using our 'C' implementation ran for 1680 core-days on 3.3GHz CPUs.

The factors of this q are:

$$\begin{aligned} q_1 &= 14918538921055873048095 1523 \\ q_2 &= 47468078385177777880302 7571 \\ q_3 &= 12884192704548253996082 17691 \\ q_4 &= 28345223950006158791380 78919 \\ q_5 &= 33227654869624444516211 92991 \\ q_6 &= 26107443111847777834166 516351 \\ q_7 &= 30664378636824844510675 581023 \\ q_8 &= 32210481761370634990205 442251 \\ q_9 &= 63132544252286444580802 666811 \\ q_{10} &= 11724615361138911136434 7809791 \\ q_{11} &= 76660946592062111276688 9525551. \end{aligned}$$

Since $2^q \equiv 1 \pmod{p}$ we can set a generator $g = 2$ to obtain a complete set of DH parameters (p, q, g). By construction q will pass a random-base Miller-Rabin primality test with probability approximately $1/2^{10}$. Since q_{11}, the largest factor of q, is 100 bits in size, the DLP in the subgroup of order q mod p for this parameter set can be solved in approximately $11 \cdot 2^{50} \approx 2^{54}$ operations.

4.3 Application to OpenSSL and PAKE protocols

OpenSSL provides the DH parameter verification function `DH_check` in `dh_check-.c`. This function takes a DH parameter set (p, q, g) and performs primality

testing on both p and q. A safe-prime setting is enforced by default, and if q is not provided then it is calculated from p via $q = (p-1)/2$. For this reason, Albrecht *et al.* [AMPS18] were not able to create malicious DH parameter sets passing OpenSSL's testing.

The primality test that OpenSSL uses is `BN_is_prime_ex`; this performs t rounds of random-base Miller-Rabin testing, where t is determined by the bit-size of p and q. Since p and q are 1024 and 1023 bits respectively, $t = 3$ rounds of Miller-Rabin are performed, at least in versions prior to OpenSSL 1.1.0i (released 14th August 2018). From version 1.1.0i onwards, t was increased to 5, with the aim of achieving 128 bits of security instead of 80 bits.[6] This change was made independently of our work and does not appear to have been influenced by the results of [AMPS18]: the numbers 3 and 5 were selected based on estimates for the average case performance of Miller-Rabin primality testing, with the OpenSSL developers implicitly assuming that p and q are generated randomly rather than maliciously.

For the DH parameter set given in Example 5, we know that q has $\varphi(q)/2^8$ Miller-Rabin non-witnesses, and thus a probability of approximately $1/2^8$ of being declared prime by a single round of Miller-Rabin testing. Hence this DH parameter set will be accepted by `DH_check` as being valid with probability approximately 2^{-24} (and the lower probability of 2^{-40} since version 1.1.0i of OpenSSL).

This may seem like a small probability, and indeed it is in a scenario where, say, malicious DH parameters are hard-coded into a server by a developer with the hope of later compromising honestly established TLS sessions between a client and a server: only 1 in 2^{24} sessions would be successfully established, and the malicious DH parameters would be quickly spotted if ever careful validation were to be carried out.

Consider instead a PAKE scenario like that envisaged by Bleichenbacher [Ble05]. Here, a client and server use some hypothetical PAKE protocol which relies on DH parameters as part of the protocol, with the server supplying the DH parameters. Assume OpenSSL's DH parameter validation is used by the client. Then an attacker impersonating the server to the client has a 1 in 2^{24} chance of fooling the client into using a weak set of DH parameters. For specific PAKE protocols, this may allow the client's password to be recovered thereafter. For example, this is the case for SRP [Wu00,TWMP07], as seen in [Ble05]. It is also true of J-PAKE [Hao17]: in this protocol, the client in a first flow sends values $g_1 = g^{x_1}, g_2 = g^{x_2}$, while the server sends $g_3 = g^{x_3}, g_4 = g^{x_4}$ (along with proofs of knowledge of the exponents). In the second flow in J-PAKE, the client sends $(g_1 g_3 g_4)^{x_2 s}$ where s is the password or a derivative of it. At this point, the attacker aborts the protocol, and uses its ability to solve the DLP to recover x_2 from the first flow and then again to recover $x_2 s$ and thence s from the second flow.

[6] Interestingly, the last time these iteration counts were changed was in February 2000 (OpenSSL version 0.9.5), before which they were all 2, independent of the bit-size of the number being tested.

We pick SRP and J-PAKE here only as illustrative examples; many other protocols would be similarly affected. We also note that the specification for using SRP in TLS [TWMP07] makes careful mention of the need to use trusted DH parameters, and gives examples of suitable parameter sets. However, [TWMP07] states that *clients SHOULD only accept group parameters that come from a trusted source*, leaving open the possibility for implementations to use parameters from untrusted sources (to remove that possibility the IETF reserved term "MUST" should have been used). Meanwhile J-PAKE [Hao17] just assumes that the DH parameters are agreed in advance and suggests some methods and sources for obtaining parameters. This does not remove the possibility of the parties using bad parameters and side-steps the important problem of parameter verification.

The power of the attack in the PAKE scenario is that the client has a secret that an attacker would like to learn; the attacker then gains an advantage by impersonating the server in a standard attack scenario. This is different from a protocol like TLS where there is no such static secret and the server is usually authenticated and therefore hard to impersonate; there we require a "malicious developer" attack scenario.

The attack can be carried out repeatedly to boost its success probability, and it can be done across a large population of users in a stealthy manner. Thus even a small per-attempt success probability of 2^{-24} may represent a significant weakness in practice.

4.4 OpenSSL Disclosure and Mitigations

As remediation to attacks of this form, we recommend that OpenSSL and other cryptographic libraries modify their DH parameter testing code to carry out stronger primality tests – as our analysis shows, 3 rounds of random-base Miller-Rabin testing is insufficient; 5 rounds are better in that it reduces the success probability of our attack to 2^{-40}, but this is still far from the 128-bit security level that the OpenSSL developers have targeted.

As part of ongoing work with the developers of OpenSSL to improve security within primality testing and prime parameter validation, we disclosed the findings of this work to OpenSSL. This resulted in a contribution to the OpenSSL codebase by a pull request[7] to increase the number of rounds of Miller-Rabin performed during the primality test on Diffie-Hellman parameters p and q during the check found in `DH_check`. This update modified the primality tests within `DH_check` to perform at the 128-bit security level, by replacing the call to set the number of Miller-Rabin rounds on the bit-size of the parameters with an enforced 64 rounds. This request was accepted by reviewers and merged into OpenSSL in March 2019 and was utilised as part of OpenSSL 1.1.1c in May 2019.

[7] see `https://github.com/openssl/openssl/commit/2500c093aa1e9c90c11c415053c0a27a00661d0d`.

5 The Elliptic Curve Setting

An elliptic curve over a prime field $\mathbb{F}_p$ in short Weierstrass form is the set of solutions $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying an equation of the type $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$, together with the point at infinity $\mathcal{O}$. When using a scheme such as Elliptic Curve Diffie-Hellman (ECDH), one typically transmits a description of the used curve via a set of domain parameters as part of the protocol, uses hard-coded parameters, or uses a standardised 'named' curve. An ECDH parameter set is typically composed of (p, E, P, q, h), where E is a description of the elliptic curve equation (typically represented by a and b), P is a base point that generates a subgroup of order q on the curve and h is the cofactor of this subgroup.

Analogously to our attacks on the parameter sets on finite field DH, we can create malicious ECDH parameter sets. The idea is to first construct a composite number q that is designed to be declared 'probably prime' by a target implementation of a probabilistic primality test but which is actually reasonably smooth, then retroactively construct a curve of suitable order $n = h \cdot q$. This can be done using the algorithm of Bröker and Stevenhagen [BS05].

Depending on the specific structure of n, a composite order will expose ECDH to attacks like Lim-Lee style small subgroup attacks as in [LL97], or may aid in solving the Elliptic Curve Discrete Logarithm Problem (ECDLP) in the order q subgroup. For this we would use the Pohlig-Hellman algorithm to solve ECDLP in time $O(B^{1/2})$ where B is an upper bound on the largest prime factor of q. For example, we could produce a 256-bit q with 4 prime factors, and hope to use the algorithm of Bröker and Stevenhagen to find a suitable curve over a 256-bit prime p of order $n = h \cdot q$ possibly even with $h = 1$. During parameter validation, q would pass a single round of the Miller-Rabin test with probability 1/8. And the ECDLP could be solved with effort approximately $4 \cdot 2^{32} = 2^{34}$ group operations.

5.1 The algorithm of Bröker and Stevenhagen

For completeness, we give a short exposition of the algorithm of Bröker and Stevenhagen [BS05].

An elliptic curve E over $\mathbb{F}_p$ has $\#E(\mathbb{F}_p) = p + 1 - t$ points where $|t| < 2\sqrt{p}$. The endomorphism ring of E contains $\mathbb{Z}[\sqrt{t^2 - 4p}]$, which is a subring of the imaginary quadratic field $K = \mathbb{Q}(\sqrt{t^2 - 4p})$. Conversely, if E is an elliptic curve over a number field whose endomorphism ring is the ring of integers of K, then (by the Complex Multiplication theory of elliptic curves) the reduction modulo p of E is an elliptic curve over $\mathbb{F}_p$ and, by taking a suitable isomorphism (a twist), we may ensure that the reduced curve has $p + 1 - t$ points.

The algorithm of Bröker and Stevenhagen exploits these ideas. Given an integer n, the first step is to construct a prime p and an integer t such that $p + 1 - t = n$ and such that $\mathbb{Q}(\sqrt{t^2 - 4p})$ has small discriminant D. Once this is done, the curve E is constructed using standard tools in Complex Multiplication (namely the Hilbert class polynomial).

We now briefly sketch the first step of the algorithm. The input is an integer n, and we wish to construct an elliptic curve with n points.

Let $D < 0$ be a discriminant of an imaginary quadratic field. We will try to find (p, t) such that $t^2 - 4p = f^2 D$ for some $f \in \mathbb{N}$. We also need $p + 1 - t = n$ and so $p = n + t - 1$. If $t^2 - 4p = f^2 D$ then

$$(t-2)^2 - f^2 D = t^2 - f^2 D - 4t + 4 = 4(p - t + 1) = 4n.$$

Hence, to construct a curve with n points it suffices to choose a discriminant D, solve the equation $w^2 - f^2 D = 4n$, and then check whether $n+(w+2)-1 = n+w+1$ is prime. Note that if $\ell \mid n$ then $w^2 - f^2 D \equiv 0 \pmod{\ell}$ and so $(\frac{D}{\ell}) \neq -1$.

An important ingredient is Cornacchia's algorithm, which solves the equation $w^2 - f^2 D = 4n$ (note that $D < 0$, so the left hand side is positive definite and the equation only has finitely many solutions). Cornacchia's algorithm starts by taking as input an integer x_0 such that $x_0^2 \equiv D \pmod{4n}$.

Putting everything together, the algorithm is as follows (we refer to [BS05] for the full details). Let $n = \ell_1 \cdots \ell_k$ be the target group order. Search over all $D < 0$ such that $D \equiv 0, 1 \pmod 4$, up to some bound $|D| < D_{\text{bound}}$. Ensure that $(\frac{D}{\ell_i}) \geq 0$ for all $\ell_i \mid n$. Determine all solutions $x_0 \in \mathbb{Z}/4n\mathbb{Z}$ such that $x_0^2 \equiv D \pmod{4n}$ and run Cornacchia's algorithm for each. Whenever we find an integer solution $w^2 - f^2 D = 4n$ check whether $p = n + w + 1$ is prime. If so, output (p, t).

Note that the algorithm is not guaranteed to succeed for a given integer n, because we are restricting to $|D| < D_{\text{bound}}$. In our application this is not a serious problem, because we are able to generate many viable choices for n.

In practice one usually desires elliptic curves of order q (supposed to be prime) or whose group order is $4q$ (Edwards and Montgomery curves have group order divisible by 4). We make one remark about the case when $n = 4q$ is even. If D is odd then any solution (w, f) to $w^2 - f^2 D = 4n$ has w odd, and so t is odd. If n is odd then this means $p = n + w + 1$ is odd, which is all good, whereas if n is even then p cannot be prime when D is odd, so when n is odd we must use odd discriminants D. On the other hand, when n is even then we can take D even (so that w and t will be even and so $p = n + w + 1$ will be odd).

5.2 Examples

We implemented the algorithm of Bröker and Stevenhagen [BS05] in SAGE, and ran it with q that are 256-bit Carmichael numbers with 3 and 4 prime factors, all congruent to 3 mod 4. These were generated using methods described in Section 3. By design, these values of q pass random-base Miller-Rabin primality testing with probability 1/4 and 1/8 per iteration, respectively. We used an early abort strategy for each q and estimate a success probability of roughly 1/4 for each q we tried. When successful, the computations took less than a minute on a laptop. The SAGE code for the first stage (finding p, t) of the 3-prime case can be found in the Appendix C.

Example 7. Set $q = q_1 q_2 q_3$ where:

$$q_1 = 120969320416809549586937 71$$
$$q_2 = 362907961250428648760813 11$$
$$q_3 = 133066252458490504545631471$$

Then q is a Carmichael number with 3 prime factors that are all congruent to 3 mod 4, so q passes random-base Miller-Rabin primality testing with probability 1/4 per iteration. Using the algorithm of Bröker and Stevenhagen, we obtain the elliptic curve $E(\mathbb{F}_p)$ defined by $y^2 = x^3 + 5$, where

$$p = 584170554761513436280134435700062590071846222494668956566359474640363466559 53$$

such that $\#E(\mathbb{F}_p) = q$ and p has 256 bits. Every point P on this curve satisfies $[q]P = \mathcal{O}$, the point at infinity, so any point can be used as a generator (of course such points may not have order q, but if q is accepted as being prime then this will not matter). The Pohlig-Hellman algorithm can be used to solve the ECDLP on this curve using about $3 \cdot 2^{42.5}$ group operations, since the largest prime factor of q has 85 bits.

Example 8. Set $q = q_1 q_2 q_3 q_4$ where:

$$q_1 = 2758736250382478263$$
$$q_2 = 8276208751147434787$$
$$q_3 = 30346098754207260883$$
$$q_4 = 91038296262621782647$$

Then q is a Carmichael number with 4 prime factors that are all congruent to 3 mod 4, so q passes random-base Miller-Rabin primality testing with probability 1/8 per iteration. Using the algorithm of Bröker and Stevenhagen, we obtain the elliptic curve $E(\mathbb{F}_p)$ defined by $y^2 = x^3 + 2$, where

$$p = 630766480273645340284659517403254049576129731687884275351051601579812429521 39$$

such that $q = \#E(\mathbb{F}_p)$ and p has 256 bits. Every point P on this curve satisfies $[q]P = \mathcal{O}$, the point at infinity, so any point can be used as a generator. The Pohlig-Hellman algorithm can be used to solve the ECDLP on this curve using about $4 \cdot 2^{33.5}$ group operations, since the largest prime factor of q has 67 bits.

The two examples above both construct examples of order q. We were also able to construct examples of order $4q$, compatible with applications that use Montgomery or Edwards curves, see for example [BL07,BCLN16].

We have not attempted to do it, but we see no reason why similar examples could not be constructed where q passes fixed-base Miller-Rabin primality tests with probability 1, as per [Ble05].

These examples illustrate the necessity for careful parameter validation, in particular robust primality testing of q, when accepting bespoke curves in cryptographic applications.

6 Conclusion and Recommendations

The best countermeasure to malicious DH and ECDH parameter sets is for protocols and systems to use only widely vetted sets of parameters, and to eliminate any options for using bespoke parameters. This is already widely done in the elliptic curve setting, not necessarily because parameter validation is hard, but because suitable parameter generation is non-trivial in the first place, and because safe and efficient implementation is much easier with a limited and well-understood set of curves. Nevertheless, issues can still arise with the provenance of parameter sets. In short, it is difficult to eliminate suspicion that a curve may have a hidden backdoor unless the generation process is fully explained and has demonstrably little opportunity for manipulation; see [BCC+15] for an extensive treatment. Similar concerns apply in the finite field setting, in the light of [Gor93,FGHT17].

On the flip-side is the argument that, in the finite field setting, using a common set of DH parameters may be inadvisable because, with the best known algorithms for finding discrete logarithms, the cost of solving many logarithms can be amortised over the cost of a large pre-computation, making commonly used DH parameter an even more attractive target. This was a crucial factor in assessing the impact of the Logjam attack on 512-bit DH arising in export cipher suites in TLS [ABD+15].

Our work adds to the weight of argument in favour of using only limited sets of carefully vetted DH parameters even in the finite field setting. This approach was recently adopted in TLS 1.3, for example, which in contrast to earlier versions of the protocol only supports a small set of DH and ECDH parameter sets, with the allowed DH parameters being specified in [Gil16].

If bespoke parameters must be used, then implementations should employ robust primality testing as part of parameter validation, using, for example, at least 64 rounds of Miller-Rabin tests, or the Baillie-PSW primality test for which there are no known pseudoprimes, cf. [AMPS18].

Acknowledgements

Massimo was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). Paterson was supported by EPSRC grants EP/M013472/1, EP/K035584/1, and EP/P009301/1.

We thank Matilda Backendal for comments on the paper and Richard G.E. Pinch for providing the data on Carmichael numbers used in Table 1.

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 15*, pages 5–17. ACM Press, October 2015.

AMPS18. Martin R. Albrecht, Jake Massimo, Kenneth G. Paterson, and Juraj Somorovsky. Prime and prejudice: Primality testing under adversarial conditions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, October 15-19, 2018*, 2018.

Arn95. François Arnault. Constructing Carmichael numbers which are strong pseudoprimes to several bases. *Journal of Symbolic Computation*, 20(2):151–161, 1995.

ASS+16. Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: breaking TLS using SSLv2. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 689–706. USENIX Association, 2016.

BBD+15. Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society Press, May 2015.

BCC+15. Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: A white paper for the black hat http: //bada55.cr.yp.to. In Liqun Chen and Shin'ichiro Matsuo, editors, *Security Standardisation Research - Second International Conference, SSR 2015, Tokyo, Japan, December 15-16, 2015, Proceedings*, volume 9497 of *Lecture Notes in Computer Science*, pages 109–139. Springer, 2015.

BCLN16. Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *J. Cryptographic Engineering*, 6(4):259–286, 2016.

BL07. Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, Heidelberg, December 2007.

Ble05. Daniel Bleichenbacher. Breaking a cryptographic protocol with pseudoprimes. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 9–15. Springer, Heidelberg, January 2005.

BPR14. Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014.

BS05. Reinier Bröker and Peter Stevenhagen. Constructing elliptic curves in almost polynomial time. arXiv:math/0511729, 2005.

BWBG+06. S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Obsoleted by RFC 8422, updated by RFCs 5246, 7027, 7919.

CMG+16. Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the juniper dual EC incident. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 468–479. ACM Press, October 2016.

CNE+14. Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335. USENIX Association, 2014.

DGG+15. Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, April 2015.

DLP93. Ivan Damgård, Peter Landrock, and Carl Pomerance. Average case error estimates for the strong probable prime test. *Mathematics of Computation*, 61(203):177–194, 1993.

DPSW16. Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, August 2016.

Erd56. P. Erdös. On pseudoprimes and Carmichael numbers. *Publ. Math. Debrecen*, 4:201–206, 1956.

FGHT17. Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A kilobit hidden SNFS discrete logarithm computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 202–231. Springer, Heidelberg, April / May 2017.

Gil16. D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). RFC 7919 (Proposed Standard), August 2016.

GMP19. Steven Galbraith, Jake Massimo, and Kenneth G. Paterson. Safety in Numbers: On the Need for Robust Diffie-Hellman Parameter Validation. Cryptology ePrint Archive, Report 2019/032, 2019. `https://eprint.iacr.org/2019/032`.

Gor93. Daniel M. Gordon. Designing and detecting trapdoors for discrete log cryptosystems. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 66–75. Springer, Heidelberg, August 1993.

GP02. Andrew Granville and Carl Pomerance. Two contradictory conjectures concerning Carmichael numbers. *Mathematics of Computation*, 71(238):883–908, 2002.

Hao17. F. Hao (Ed.). J-PAKE: Password-Authenticated Key Exchange by Juggling. RFC 8236 (Informational), September 2017.

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.

LL97. Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 249–263. Springer, Heidelberg, August 1997.

Mon80. Louis Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoretical Computer Science*, 12(1):97–108, 1980.

Nar14. Shyam Narayanan. *Improving the Speed and Accuracy of the Miller-Rabin Primality Test*. MIT PRIMES-USA, 2014. `https://math.mit.edu/research/highschool/primes/materials/2014/Narayanan.pdf`.

Pin08. Richard GE Pinch. The Carmichael numbers up to 10^{21}. In *Proceedings Conference on Algorithmic Number Theory*, volume 46 of *Turku Centre for Computer Science General Publications*, 2008.

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

S$^+$18. William Stein et al. *Sage Mathematics Software Version 8.3*. The Sage Development Team, 2018. Available at `http://www.sagemath.org`.

TWMP07. D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054 (Informational), November 2007.

VAS$^+$17. Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, and Nadia Heninger. Measuring small subgroup attacks against Diffie-Hellman. In *NDSS 2017*. The Internet Society, February / March 2017.

Won16. David Wong. How to backdoor Diffie-Hellman. Cryptology ePrint Archive, Report 2016/644, 2016. `https://eprint.iacr.org/2016/644`.

Wu00. T. Wu. The SRP Authentication and Key Exchange System. RFC 2945 (Proposed Standard), September 2000.

YY97. Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997.

A SAGE code of the Erdös Method for Generating Carmichael Numbers

We present below our SAGE code implementation of the Erdös Method for generating Carmichael numbers. This particular code was used to generate the Carmichael numbers with 8 and 16 factors in Example 3.

```
import itertools
from operator import mul
from sage.arith.functions import LCM_list


def all_combinations(any_list):
    """
    Wrapper for itertools to generate all possible combinations of all
    (non trivial) sizes.
```

```
    """
    return itertools.chain.from_iterable(
        itertools.combinations(any_list, i + 1)
        for i in xrange(len(any_list)))


def LCMpim1(n):
    """
    Takes as input n: a list of integers p_i and returns the lcm(p_i-1) for all i
    """
    pim1list = []
    for pi in n:
        pim1 = pi - 1
        pim1list.append(pim1)
    return LCM_list(pim1list)


def listbuild(L):
    """
    Takes as input a (highly composite) number L and returns a list of all primes
    p such that p-1 | L where p does not divide L. We include the additional
    requirement that p = 3 mod 4.
    """
    a = list(factor(L))
    p = []
    for y in a:
        for i in range(0, y[1]):
            p.append(y[0])

    pvals = all_combinations(p)
    ps = []
    for pp in pvals:
        t = reduce(mul, pp, 1)
        tt = t + 1
        if tt.is_prime(proof=False) and L % tt != 0:
            if tt not in ps:
                ps.append(tt)
    pps = []
    ps.sort()
    # we now filter results to only inlude p with p = 3 mod 4
    for p in ps:
        if p % 4 == 3:
            pps.append(p)
    return pps


def erdos_build(factors, L, k):
    """
    This function takes a list of possible factors, a (highly composite) integer L
    and k, and produces a Carmichael number with k factors sampled from "factors"
    such that the LCM of each factor p_i - 1 is equal to L. Output is parsed as
    n,[p_1,p_2,...,p_k] where n = p_1 * p_2 * ... * p_k.
    """
    if k <=2:
        print "Choice of factors must be >=3"
        return 0
    for i in itertools.combinations(factors, k):
        v = reduce(mul, i, 1)
        if v % L == 1:
            fin = list(i)
            fin.sort()
            if LCMpim1(fin) == L:
                return [v,fin]
    print "None found, try increasing size of factor list"


L = 53603550
factors = listbuild(L)
print factors, L, len(factors)

print erdos_build(factors, L, 8)
print erdos_build(factors, L, 16)
```

B C Code of the Modified Granville and Pomerance Method for Generating Carmichael Numbers q such that $p = 2q + 1$ is Prime

We present below our C code used to generate a Carmichael number q with 9 prime factors such that $p = 2q + 1$ is a 1024 bit prime as in Example 5.

```
#define _XOPEN_SOURCE 500
```

```
#include <stdint.h>
#include <stdio.h>                      /* printf() */
#include <stdlib.h>     /* abort() */
#include <unistd.h>     /* getopt() */
#include <gmp.h>

/* Command Line Parsing  */
#define DEFAULT_COUNT 37
#define DEFAULT_OFFSET 0

struct _cmdline_params_struct {
  uint32_t count;  // we use this for parallelisation
  uint32_t offset; //< how much we want to offset the starting value of k by
};

typedef struct _cmdline_params_struct cmdline_params_t[1];

static inline void print_help_and_exit() {
  printf("-c    log2 of number of trials (default: %d)\n", DEFAULT_COUNT);
  printf("-o    offset on starting k value, where offset*c (default: %d)\n", DEFAULT_OFFSET);
  abort();
}

static inline void parse_cmdline(cmdline_params_t params, int argc, char *argv[]) {
  params->count = DEFAULT_COUNT;
  params->offset = DEFAULT_OFFSET;

  int c;
  while ((c = getopt(argc, argv, "c:o:")) != -1) {
    switch(c) {
    case 'c':
      params->count = (uint32_t)strtoul(optarg, NULL, 10);
      break;
    case 'o':
      params->offset = (int32_t)strtoul(optarg,NULL, 10);
      break;
    case ':':  /* without operand */
      print_help_and_exit();
    case '?':
      print_help_and_exit();
    }
  }
  printf("-c %d -o %d\n",
       params->count, params->offset);
}

/* Logging */
void logit(mpz_t q, mpz_t q1, mpz_t q2, mpz_t q3, mpz_t q4, mpz_t q5, mpz_t q6, mpz_t q7,mpz_t q8, mpz_t q9) {

  char tmp[2000];
  snprintf(tmp, 2000, "0x%s:0x%s:0x%s:0x%s:0x%s:0x%s:0x%s:0x%s:0x%s:0x%s",
   mpz_get_str(NULL, 16, q), mpz_get_str(NULL, 16, q1), mpz_get_str(NULL, 16, q2),
   mpz_get_str(NULL, 16, q3), mpz_get_str(NULL, 16, q4), mpz_get_str(NULL, 16, q5),
   mpz_get_str(NULL, 16, q6), mpz_get_str(NULL, 16, q7), mpz_get_str(NULL, 16, q8),
   mpz_get_str(NULL, 16, q9));
  FILE *fh = fopen("CARM-9.log", "a");
  fprintf(fh, "%s\n", tmp);
  fclose(fh);
}

/*
 * Function:  main
 * --------------------
 * This function uses the modified Granville Pomerance method to generate a
 * Carmichael number q of cryptographic size, such that N = 2q+1 is prime.
 *
 * This function is currently not set up for generality, and does not perform
 * sanity checks. We specifically set up an instance of this code to search for
 * a single valid example. This is the 9 factored example that is given a starting
 * Carmichael number p = p_1 *...* p_9 generated previously by the Erdos method.
 *
 * The function iterates through kprime (k') values to construct:
 *    m = kL + 1, where k = k' * s
 *    then q_i = M(p_i-1)+1 for all i
 * such that q = q_1 * ... * q_9 is approx 1023 bits.
 *
 * We then test each q_i for primality, iterating to the next k' value if composite.
 * Finally, if all q_i are prime, we construct q = q_1 * ... * q_9 and test if
 * N = 2q+1 is prime. If true, we log q, and its factors.
 *
 */
int main(int argc, char *argv[])
{
  mpz_t s,p1,p2,p3,p4,p5,p6,p7,p8,p9,q,q1,q2,q3,q4,q5,q6,q7,q8,q9,kprime,fudge2,fudge3,fudge4,fudge5,k,m,off,L,N;
  mpz_init(q);
  mpz_init(q1);
  mpz_init(q2);
  mpz_init(q3);
  mpz_init(q4);
  mpz_init(q5);
```

```
mpz_init(q6);
mpz_init(q7);
mpz_init(q8);
mpz_init(q9);
mpz_init(k);
mpz_init(m);
mpz_init(off);
mpz_init(N);
int res;

cmdline_params_t params;
parse_cmdline(params, argc, argv);
// here we set up our specific starting Carmichael number p and other parameters
mpz_init_set_str(s, "3", 10);
mpz_init_set_str(kprime, "1", 10);
mpz_init_set_str(fudge2, "1", 10);
mpz_init_set_str(fudge3, "1", 10);
mpz_init_set_str(fudge4, "1", 10);
mpz_init_set_str(fudge5, "1", 10);
mpz_init_set_str(p1, "70", 10);
mpz_init_set_str(p2, "130", 10);
mpz_init_set_str(p3, "646", 10);
mpz_init_set_str(p4, "1870", 10);
mpz_init_set_str(p5, "4522", 10);
mpz_init_set_str(p6, "4750", 10);
mpz_init_set_str(p7, "46750", 10);
mpz_init_set_str(p8, "432250", 10);
mpz_init_set_str(p9, "350350", 10);
mpz_init_set_str(L, "565815250", 10);
uint64_t Lbits = 30;

mpz_init_set_ui(off,params->offset);
mpz_mul_2exp(off,off,params->count);

size_t p1bits = mpz_sizeinbase (p1, 2);
size_t sbits = mpz_sizeinbase (s, 2);

// we now make some speicific alterations to ensure the final N is 1024 bits
uint64_t t = 9;
uint64_t fudgefactor = 1;
uint64_t power = 113 - (t/2 -1) -p1bits - Lbits - sbits + fudgefactor;

mpz_mul_2exp(kprime,kprime,power);
mpz_mul_2exp(fudge2,fudge2,power-1);
mpz_mul_2exp(fudge3,fudge3,power-4);
mpz_mul_2exp(fudge4,fudge4,power-5);
mpz_mul_2exp(fudge5,fudge5,power-6);
mpz_sub(kprime,kprime,fudge2);
mpz_sub(kprime,kprime,fudge3);
mpz_sub(kprime,kprime,fudge4);

if (params->offset != 0) {
  mpz_add(kprime,kprime,off);
}
// The following for loop accounts for the bulk of the time to run
for (uint64_t i = 0; i <= (1ULL)<<params->count; i++){
  mpz_add_ui(kprime,kprime,1);
  mpz_mul(k,kprime,s);
  mpz_mul(m,k,L);
  mpz_add_ui(m,m,1);

  //q1
  mpz_mul(q1,m,p1);
  mpz_add_ui(q1,q1,1);
  res= mpz_probab_prime_p (q1, 2);
  if (!res) {
      continue;
  }
  //q2
  mpz_mul(q2,m,p2);
  mpz_add_ui(q2,q2,1);
  res= mpz_probab_prime_p (q2, 2);
  if (!res) {
      continue;
  }
  //q3
  mpz_mul(q3,m,p3);
  mpz_add_ui(q3,q3,1);
  res= mpz_probab_prime_p (q3, 2);
  if (!res) {
      continue;
  }
  //q4
  mpz_mul(q4,m,p4);
  mpz_add_ui(q4,q4,1);
  res= mpz_probab_prime_p (q4, 2);
  if (!res) {
      continue;
  }
  //q5
```

```
    mpz_mul(q5,m,p5);
    mpz_add_ui(q5,q5,1);
    res= mpz_probab_prime_p (q5, 2);
    if (!res) {
        continue;
    }
    //q6
    mpz_mul(q6,m,p6);
    mpz_add_ui(q6,q6,1);
    res= mpz_probab_prime_p (q6, 2);
    if (!res) {
        continue;
    }
    //q7
    mpz_mul(q7,m,p7);
    mpz_add_ui(q7,q7,1);
    res= mpz_probab_prime_p (q7, 2);
    if (!res) {
        continue;
    }
    //q8
    mpz_mul(q8,m,p8);
    mpz_add_ui(q8,q8,1);
    res= mpz_probab_prime_p (q8, 2);
    if (!res) {
        continue;
    }
    //q9
    mpz_mul(q9,m,p9);
    mpz_add_ui(q9,q9,1);
    res= mpz_probab_prime_p (q9, 2);
    if (!res) {
        continue;
    }

    mpz_mul(q,q1,q2);
    mpz_mul(q,q,q3);
    mpz_mul(q,q,q4);
    mpz_mul(q,q,q5);
    mpz_mul(q,q,q6);
    mpz_mul(q,q,q7);
    mpz_mul(q,q,q8);
    mpz_mul(q,q,q9);

    mpz_mul_2exp(N,q,1);
    mpz_add_ui(N,N,1);
    res= mpz_probab_prime_p (N, 2);
    if (!res) {
      continue;
    }
    printf("PRIME!\n" );
    logit(q,q1,q2,q3,q4,q5,q6,q7,q8,q9);
  }

  mpz_clear(s);
  mpz_clear(p1);
  mpz_clear(p2);
  mpz_clear(p3);
  mpz_clear(p4);
  mpz_clear(p5);
  mpz_clear(p6);
  mpz_clear(p7);
  mpz_clear(p8);
  mpz_clear(p9);
  mpz_clear(q1);
  mpz_clear(q2);
  mpz_clear(q3);
  mpz_clear(q4);
  mpz_clear(q5);
  mpz_clear(q6);
  mpz_clear(q7);
  mpz_clear(q8);
  mpz_clear(q9);
  mpz_clear(kprime);
  mpz_clear(fudge2);
  mpz_clear(fudge3);
  mpz_clear(fudge4);
  mpz_clear(fudge5);
  mpz_clear(k);
  mpz_clear(m);
  mpz_clear(off);
  mpz_clear(L);
  mpz_clear(N);
  return 0;
}
```

C SAGE code for Algorithm of Bröker and Stevenhagen

We present below our SAGE code for the first step of the algorithm of Bröker and Stevenhagen in the case where N, the target group order, has 3 prime factors.

```
# Generate elliptic curve using CM with group order divisible by product p*q*r that is a fake prime.


# Cornacchia algorithm
def Cornacchia( A, B, D ):
   a = A
   b = B
   while (b^2 > A):
     rrem = int( Mod(a,b) )
     a = b
     b = rrem
   x = b
   f2 = (A - x^2) / -D
   f = int( sqrt( f2 ))
   return x, f



# [584170554761513436280134435700062590076357016263612392265089290457585365018 51,
p = 120969320416809549586937 71
q = 362907961250428648760813 11
r = 133066252458490504545631471
N = p*q*r


DBOUND = -2000;

# First try to construct a curve with N points
D = -3
while (D > DBOUND):
  if (1 == legendre_symbol( D, p )) and (1 == legendre_symbol( D, q )) and (1 == legendre_symbol( D, r )):
    F = GF( p )
    x01 = int( sqrt( F( D ) ))
    F = GF( q )
    x02 = int( sqrt( F( D ) ))
    F = GF( r )
    x03 = int( sqrt( F( D ) ))
# There are 8 possible choices for x0 coming from the 2^3 choices of sign +/- x01, +/- x02, +/- x03
    ct = 0
    while (ct < 8):
      x0 = crt( crt( x01, x02, p, q ), x03, p*q, r )
      while (0 != Mod(x0^2-D,4*N)):
        x0 = x0+N
      x, f = Cornacchia( 4*N, x0, D )
      if ( 0 == (x^2 - D*f^2 - 4*N)):
         pp = int( N + x + 1 )
         if is_prime(pp):
            print "Success (D,x,f) = ", D, x, f
            print "And get a prime p = ", pp
         pp = int( N - x + 1 )
         if is_prime(pp):
            print "Success with other sign (D,x,f) = ", D, x, f
            print "And get a prime p = ", pp
      x01 = p - x01
      if (0 == (ct % 2)):
        x02 = q - x02
      if (0 == (ct % 4)):
        x03 = r - x03
      ct = ct + 1
  D = D - 4


# Now consider curves whose number of points is a multiple of 2*N
# Algorithm is basically the same except D now must be even
c = 1
while (c < 5):
  NN = 2*c*N
  c = c + 1
  D = -4
  while (D > DBOUND):
    D = D - 4
    DD = D
    if (1 == legendre_symbol( D, p )) and (1 == legendre_symbol( D, q )) and (1 == legendre_symbol( D, r )):
      F = GF( p )
      x01 = int( sqrt( F( DD ) ))
      F = GF( q )
      x02 = int( sqrt( F( DD ) ))
      F = GF( r )
      x03 = int( sqrt( F( DD ) ))
      ct = 0
      while (ct < 8):
        x0 = crt( crt( x01, x02, p, q ), x03, p*q, r )
```

```
chk=0
while (0 != Mod(x0^2-DD,4*NN)) and (chk < 100):
  chk = chk+1
  x0 = x0+N
x, f = Cornacchia( 4*NN, x0, D )
if ( 0 == (x^2 - DD*f^2 - 4*NN)):
   pp = int( NN + x + 1 )
   if is_prime(pp):
      print "Success (D,x,f) = ", DD, x, f
      print "And get a prime p = ", pp
   pp = int( NN - x + 1 )
   if is_prime(pp):
      print "Success with other sign (D,x,f) = ", DD, x, f
      print "And get a prime p = ", pp
x01 = p - x01
if (0 == (ct % 2)):
  x02 = q - x02
if (0 == (ct % 4)):
  x03 = r - x03
ct = ct + 1
```