IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 146–166. DOI:10.46586/tches.v2023.i4.146-166

Dlog is Practically as Hard (or Easy) as DH – Solving Dlogs via DH Oracles on EC Standards

Alexander May[1] and Carl Richard Theodor Schneider[2]

[1] Ruhr University Bochum, Bochum, Germany, alex.may@rub.de
[2] Ruhr University Bochum, Bochum, Germany, carl.schneider@rub.de

Abstract. Assume that we have a group G of known order q, in which we want to solve discrete logarithms (dlogs). In 1994, Maurer showed how to compute dlogs in G in poly time given a Diffie-Hellman (DH) oracle in G, and an auxiliary elliptic curve $\hat{E}(\mathbb{F}_q)$ of smooth order. The problem of Maurer's reduction of solving dlogs via DH oracles is that no efficient algorithm for constructing such a smooth auxiliary curve is known. Thus, the implications of Maurer's approach to real-world applications remained widely unclear.
In this work, we explicitly construct smooth auxiliary curves for 13 commonly used, standardized elliptic curves of bit-sizes in the range $[204, 256]$, including e.g., NIST P-256, Curve25519, SM2 and GOST R34.10. For all these curves we construct a corresponding cyclic auxiliary curve $\hat{E}(\mathbb{F}_q)$, whose order is 39-bit smooth, i.e., its largest factor is of bit-length at most 39 bits.
This in turn allows us to compute for all divisors of the order of $\hat{E}(\mathbb{F}_q)$ exhaustively a codebook for all discrete logarithms. As a consequence, dlogs on $\hat{E}(\mathbb{F}_q)$ can efficiently be computed in a matter of seconds. Our resulting codebook sizes for each auxiliary curve are less than 29 TByte individually, and fit on our hard disk.
We also construct auxiliary curves for NIST P-384 and NIST P-521 with a 65-bit and 110-bit smooth order.
Further, we provide an efficient implementation of Maurer's reduction from the dlog computation in G with order q to the dlog computation on its auxiliary curve $\hat{E}(\mathbb{F}_q)$. Let us provide a flavor of our results, e.g., when G is the NIST P-256 group, the results for other curves are similar. With the help of our codebook for the auxiliary curve $\hat{E}(\mathbb{F}_q)$, and less than 24,000 calls to a DH oracle in G (that we simulate), we can solve discrete logarithms on NIST P-256 in around 30 secs.
From a security perspective, our results show that for current elliptic curve standards the difficulty of solving DH is practically tightly related to the difficulty of computing dlogs. Namely, unless dlogs are easy to compute on these curves G, we provide a very concrete security guarantee that DH in G must also be hard. From a cryptanalytic perspective, our results show a way to efficiently solve discrete logarithms in the presence of a DH oracle.

Keywords: Public-Key Cryptography · Discrete Logarithm · Elliptic Curve · Diffie-Hellman · Oracles · Implementation

1 Introduction

While we would like to base cryptographic security on the fundamental discrete logarithm (dlog) problem in a group G, most cryptographic schemes like ElGamal encryption [ElG84] and the famous (EC-)DSA signatures [Lab94, Lab13] require (at least) the stronger assumption that the Diffie-Hellman (DH) problem is hard. Vulnerabilities of the DH problem in practice can lead to quite dramatic consequences, see the Logjam attack [ABD+15] as an example for the finite field setting.

Received: 2023-04-15 Accepted: 2023-06-15 Published: 2023-08-31

While in theory Maurer and Wolf [Mau94, MW96, MW99] reduced the dlog problem in a group G of order q to the DH problem —assuming the existence of a suitable auxiliary elliptic curve $\hat{E}(\mathbb{F}_q)$ of smooth order— the reduction's practical implications remained widely unclear. This is quite surprising given the ubiquitous use of ECDSA in practice.

Our work brings Maurer's algorithm into practice, answering the *practical* tightness of dlog and DH for the commonly used elliptic curve standards. To this end, let us have a closer look at Maurer's reduction.

High-level description of Maurer's algorithm. Assume we have a base curve $E(\mathbb{F}_p)$ with a generator P of prime order $\text{ord}(P) = q$. Let $Q = kP$ be our dlog problem on $E(\mathbb{F}_p)$. Notice that $Q = kP$ uniquely defines $k \in \mathbb{Z}_q$. Therefore, we call $Q = kP$ an implicit representation of k. For the base curve $E(\mathbb{F}_p)$, we also need access to a Diffie-Hellman oracle $\text{DH} : (x_1P, x_2P) \mapsto x_1x_2P$, where $x_1, x_2 \in \mathbb{Z}_q$, and we can *freely choose both inputs* x_1P, x_2P to the oracle.

Let $\hat{E}(\mathbb{F}_q)$ be an auxiliary curve with a generator $\hat{P}$ of smooth order $\text{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{e_i}$. For any point $\hat{Q} = (x, y) \in \hat{E}(\mathbb{F}_q)$ we denote by $\hat{Q} = [xP, yP]$ its *implicit representation*, where both the x- and y-coordinate are implicitly represented via points on the base curve.

In the presence of such an auxiliary curve and a DH oracle we proceed as follows.

(1) **Auxiliary Curve Construction.** We lift our dlog problem $Q = kP$ on $E(\mathbb{F}_p)$ to an implicitly represented, lifted dlog problem $\hat{Q} = [kP, yP] = u\hat{P}$ on an auxiliary curve $\hat{E}(\mathbb{F}_q)$ [1] with smooth order $\text{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{e_i}$.

(2) **Auxiliary Curve Dlog Codebook Construction.** We run the Silver-Pohlig-Hellman algorithm on the lifted dlog instance to precompute and store all values $v_i \bmod p_i^{e_i}$ for all i, and all $v_i \in \mathbb{Z}_{p_i^{e_i}}$ in implicit representation. This precomputation gives us a codebook for all dlogs on the auxiliary curve $\hat{E}(\mathbb{F}_q)$, which allows us to compute dlogs on $\hat{E}(\mathbb{F}_q)$ via simple table lookups.

(3) **Dlog Computation on the Base Curve.** For our lifted dlog instance $\hat{Q} = u\hat{P}$, we determine all $u_i = u \bmod p_i^{e_i}$, and combine them via Chinese Remaindering. All computations on the lifted dlog instance are performed with implicit representations, using DH oracles for multiplication/division/squaring in $\mathbb{F}_q$ for performing the elliptic curve arithmetic on $\hat{E}(\mathbb{F}_q)$. After we determined u, we compute $u\hat{P} = \hat{Q} = (k, y) \in \hat{E}(\mathbb{F}_q)$ now —as opposed to Step (1)— *explicitly*, from which we directly read off the desired dlog solution k.

The main problem of Maurer's reduction is that it is *non-uniform*, i.e., it simply assumes the existence of a smooth auxiliary curve $\hat{E}(\mathbb{F}_q)$ as part of the input. However, the tightness and practicality of the reduction heavily depends on $\hat{E}(\mathbb{F}_q)$'s smoothness. Therefore, the reductions' practical implications remain unclear: Which auxiliary curves can we construct? How many DH oracles queries do we require? How fast are subsequent dlog computations?

We answer all these questions in the following.

Our Results. We construct for each of the base curves from the following 14 cryptographic standards a corresponding auxiliary curve: Anomalous, ANSSIFRP256v1, BLS12-381, BN(2,254), brainpoolP256t1, Curve25519, $F_p - 256$ from GM/T 0003.2-2012, GOST R 34.10, M-221, NIST P-224, NIST P-256, secp256k1, SM2, NIST P-384, and NIST P-521.

Auxiliary Curve Construction. For every base curve, we randomly sampled auxiliary curves $\hat{E}(\mathbb{F}_q)$, computed their order $|\hat{E}(\mathbb{F}_q)|$ via Schoof's algorithm, and factored the order

[1]Here we assume for simplicity that k is a valid x-coordinate on $\hat{E}(\mathbb{F}_q)$. We later show how to modify k otherwise.

into its prime factors. For each curve with order at most 256 bit, it took us less than 10^6 samples to discover a cyclic, B-smooth auxiliary curve $\hat{E}(\mathbb{F}_q)$ with B smaller than 40 bit. We explicitly provide these curves together with their generator point $\hat{P}$ and the factorization of $\mathrm{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{e_i}$.

Auxiliary Curve Dlog Codebook. For every $i = 1, \ldots, n$ we compute the generator $\hat{P}_i = (\mathrm{ord}(\hat{P})/p_i^{e_i})\hat{P}$ of the subgroup of order $p_i^{e_i}$. For all i and all $0 \leq \ell_i < p_i^{e_i}$ we then compute the values of $\ell_i \hat{P}_i$ (in implicit representation). This gives us a complete codebook for all dlog computations in $\hat{E}(\mathbb{F}_q)$. We do some further optimization to minimize the size of our codebooks, and to minimize the number of DH oracle calls when using Maurer's algorithm. As a result, all our codebooks for the auxiliary curves corresponding to elliptic curve standards with at most 256 bit require less than 29 TByte.

Dlog Computation on the Base Curve. Let $Q = kP$ be our dlog problem on the base curve. We lift it to $\hat{Q} = [kP, yP] = u\hat{P}$ on our auxiliary curve $\hat{E}(\mathbb{F}_q)$. For every $i = 1, \ldots, n$ we compute $[\mathrm{ord}(\hat{P})/p_i^{e_i}]\hat{Q}$ in implicit representation using DH oracles for multiplication/division in $\mathbb{F}_q$. A comparison with our codebook for $\hat{P}_i$ immediately reveals $u_i = u \bmod p_i^{e_i}$. From $u_1, \ldots, u_n$ we compute, via Chinese Remaindering, the dlog u on our auxiliary curve $\hat{E}(\mathbb{F}_q)$. The computation $\hat{Q} = u\hat{P} = (k, y)$ eventually reveals the dlog k on our base curve $E(\mathbb{F}_p)$. With our optimizations and the help of our codebook, a complete dlog computation on any of the considered standardized elliptic curves with at most 256 bits requires less than 24,000 DH oracle calls, and a total running time of around 30 secs.

Source code. Our code for finding auxiliary curves, computing the codebooks, and performing dlog computations is available at `https://github.com/e70847616e1d2c84/discrete-log`.

Comparison with Previous Work. In a seminal work, Muzereau, Smart, and Vercauteren [MSV04] provided auxiliary curves for some elliptic curve standards back in 2004. However, most of the curves they considered almost 20 years ago had group orders smaller than 200 bits, which is by now considered too small. For secp256k1, Muzereau-Smart-Vercauteren provide an auxiliary curve with 56 bit smoothness, whereas we succeed to construct an auxiliary curve with 37 bit smoothness.

This significant improvement allows us for the first time to compute a codebook for the auxiliary curve, and to perform dlog computations on secp256k1 with the help of a DH-oracle, in practice! This would not be possible with the 56 bit smooth auxiliary curve of [MSV04]. As another comparison, [MSV04] provide an auxiliary curve for secp384r1 with 83 bit smoothness, whereas we achieved to find an auxiliary curve for NIST P-384 with 65 bit smoothness.

Follow-up works by Bentahar [Ben05] and Kushwaha [Kus18] focussed on the theoretical aspects of tightness of dlog and DH with respect to the number of required DH-oracle calls. Namely, the number of DH-oracle calls is minimized for groups of order q when we constructed an auxiliary curve, whose order splits into 3 co-prime factors of size roughly $q^{\frac{1}{3}}$. While such a tightness analysis is interesting from a theoretical perspective, it does not lead to practical attacks. For 256-bit standards this would imply auxiliary curves with 3 factors around 85 bits. For these factors we would not be able to determine dlogs efficiently.

Our approach instead focuses on practicality, i.e., on auxiliary curves as smooth as possible. We then also try to minimize the number of DH-oracle calls, but without sacrificing practicality.

For group actions, the quantum equivalence of dlog and DH has been established by Galbraith, Panny, Smith, and Vercauteren [GPSV18], which can be considered a quantum analogue of Maurer's reduction. Interestingly, as opposed to the classical setting, in the quantum setting the construction of an auxiliary group is not required.

A nice overview of results in this area can be found in Galbraith [Gal12] and in Galbraith, Gaudry [GG16].

Organization of the paper. In Section 2, we show how to construct auxiliary curves for current elliptic curve standards. In Section 3, we compute the codebooks for our auxiliary curves. In Section 4, we detail how to compute dlogs on our standardized base curves with the help of our constructed auxiliary curves, their codebooks and a DH-oracle. Our results for curves up to 256-bit order are summarized in Subsection 4.3, and Subsection 4.4 discusses the required strength of our DH oracle. In Section 5 we show to which extent our results generalize to groups of larger order like 384 and 521 bits.

Our auxiliary curves for standards with at most 256 bits can be found in Appendix A. Appendix B contains our auxiliary curves for NIST P-384 and NIST P-521.

2 Auxiliary Curve Construction

Let $E(\mathbb{F}_p)$ be our base curve with a generator P of order $\mathrm{ord}(P) = q$. We sample random auxiliary curves $\hat{E}(\mathbb{F}_q)$ with curve equation $y^2 = x^3 + Ax + B$ by randomly sampling $A, B \in \mathbb{F}_q$ with non-zero discriminant $4A^3 + 27B^2 \neq 0 \bmod q$. Any elliptic curve $\hat{E}(\mathbb{F}_q)$ is either cyclic (i.e., generated by a single point), or the product of two cyclic groups. We rejected non-cyclic groups.

For cyclic groups, we compute the order of $\hat{E}(\mathbb{F}_q)$ with Schoof's algorithm [Sch85], together with a generator $\hat{P}$, i.e., $\mathrm{ord}(\hat{P}) = |\hat{E}(\mathbb{F}_q)|$.

We then factor $\mathrm{ord}(\hat{P})$ into its prime factors. The whole procedure is repeated, until we find a cyclic auxiliary curve $\hat{E}(\mathbb{F}_q)$ with B-smooth order for some $B < 40$ bit. The details of the resulting algorithm are provided in Algorithm 1.

Algorithm 1 Auxiliary Curve Construction with Smooth Order

Input: q, prime order of generator P of our base curve $E(\mathbb{F}_p)$

Output: $A, B \in \mathbb{F}_q$, defining auxiliary $\hat{E}(\mathbb{F}_q) : y^2 = x^3 + Ax + B \bmod q$, generator $\hat{P}$ of $\hat{E}(\mathbb{F}_q)$

```
1: repeat
2:     repeat
3:         Sample $A, B \in_R \mathbb{F}_q$
4:     until ($4A^3 + 27B^2 \neq 0 \mod q$) and $\hat{E}(\mathbb{F}_q) : y^2 = x^3 + Ax + B$ is cyclic
5:     Compute $|\hat{E}(\mathbb{F}_q)|$ with Schoof's algorithms, together with a generator $\hat{P}$.
6:     Factor ord $\hat{P} = \prod_i {p_i}^{e_i}$
7: until $\max_i \{p_i^{e_i}\}$ is sufficiently small
8: return $A, B, \hat{P}$
```

We provide the results of running Algorithm 1 in Table 1. For all elliptic curve standard groups with maximal group size 256 bit we found a cyclic auxiliary curve with at most 39-bit smooth group order within at most 10^6 samples.

In Appendix A, we provide a complete list of all auxiliary curves for the considered elliptic curve standards with at most 256 bits, together with their specification A, B, their generator point $\hat{P} = (x(\hat{P}), y(\hat{P}))$, and the factorization of their group order.

Table 1: B-smoothness achieved for the constructed auxiliary curves.

Base curve $E(\mathbb{F}_p)$	q [bit]	Samples	B [bit]
Anomalous	204	71311	33
ANSSIFRP256v1	256	156841	39
BLS12-381	255	3829640	36
BN(2,254)	254	7060	39
brainpoolP256t1	256	498440	39
Curve25519	253	104806	37
$F_p - 256$ (GM/T 0003.2-2012)	256	514595	39
GOST R 34.10	256	113350	37
M-221	219	229513	37
NIST P-224	224	76980	38
NIST P-256	256	437088	37
secp256k1	256	991302	37
SM2	256	840273	39

2.1 Runtime Analysis of Our Auxiliary Curve Construction.

The runtime of our auxiliary curve construction in Algorithm 1 is dominated by the expected number of samples that we have to process until the order $N := \left|\hat{E}(\mathbb{F}_q)\right|$ has a B-smooth factorization.

By Hasse's theorem [Sil09], we have $q + 1 - 2\sqrt{q} \leq N \leq q + 1 + 2\sqrt{q}$. Thus, N lies in a so-called *Hasse interval* of length $4\sqrt{q}$ centered at $q + 1$. A theorem of Lenstra [LJ87] shows that the distribution of N within the Hasse interval is almost uniform.

Let $\Psi(q, B)$ denote the total number of B-smooth integers up to q. Then a random integer picked uniformly from the interval $[1, q]$ is B-smooth with probability

$$p_{\leq B}(q) := \Pr\Big[\text{integer from } [1, q] \text{ is } B\text{-smooth}\Big] = \frac{\Psi(q, B)}{q}. \tag{1}$$

For simplicity, we make the standard number-theoretic assumption that any elliptic curve group order N taken uniformly from the Hasse interval (instead of $[1, q]$) is B-smooth with the same probability $p_{\leq B}(q)$.

We are also interested in the probability that a number is *exactly B-smooth*, meaning that it is B-smooth but not $(B - 1)$-smooth. A random number picked uniformly from $[1, q]$ is exactly B-smooth with probability

$$p_{=B}(q) := \Pr\Big[\text{integer from } [1, q] \text{ is exactly } B\text{-smooth}\Big] = \frac{\Psi(q, B) - \Psi(q, B - 1)}{q}. \tag{2}$$

We approximate $\Psi(q, B) \approx q \cdot \rho\left(\frac{\log q}{B}\right)$ [Gra08], where log is base 2 and ρ is the Dickman-de Bruijn ρ-function that we evaluate with SageMath[2]. Thus, we estimate that on expectation we obtain a B-smooth, respectively an exactly B-smooth, curve order after sampling

$$p_{\leq B}^{-1}(q) \approx \rho\left(\frac{\log q}{B}\right)^{-1}, \text{ respectively } p_{=B}^{-1}(q) \approx \left(\rho\left(\frac{\log q}{B}\right) - \rho\left(\frac{\log q}{B - 1}\right)\right)^{-1}, \tag{3}$$

many curves in Algorithm 1.

In Figure 1 we plotted for 437,088 sampled auxiliary curves for NIST P-256 the observed relative amount of B bit-smooth group orders N (yellow dots), as well as our estimate for

[2] https://doc.sagemath.org/html/en/reference/functions/sage/functions/transcendental.html#sage.functions.transcendental.DickmanRho

$p_{=B}(q)$ from Eq. (2) (blue line). We see that our experimental observation very accurately matches our estimate.

From Figure 1 we observe that 128-bit smooth orders appear most frequently, and $p_{=B}(q)$ drops quite quickly for smaller B. We only found 97 curves with $B < 50$, until we eventually discovered our 37-bit smooth auxiliary curve after $4.4 \cdot 10^5$ trials. This experimentally observed number of trials is in line with the expected number $p_{\leq 37}^{-1}(q) \approx 8.8 \cdot 10^5$ trials.

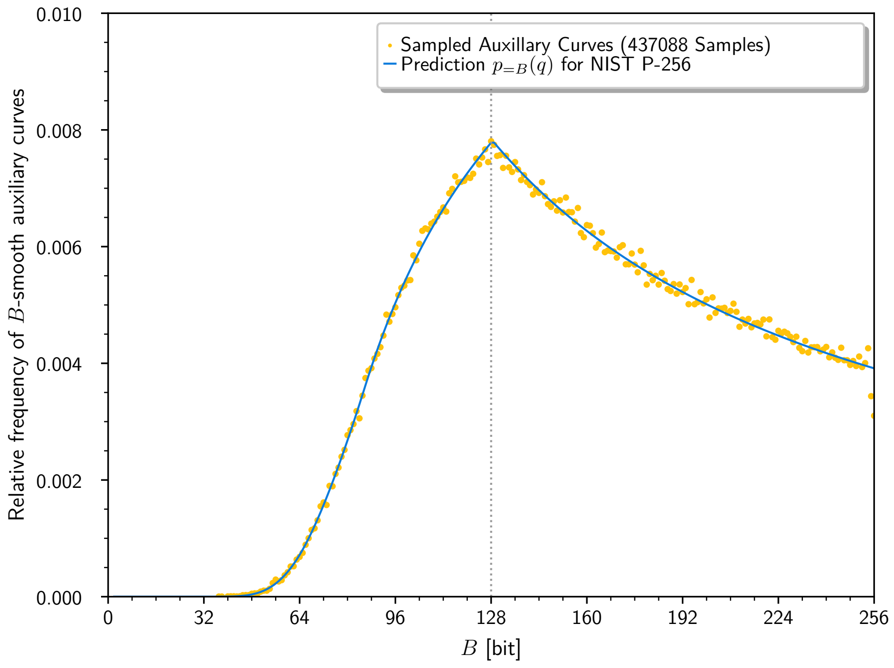

Figure 1: Observed and estimated relative frequency of B-smooth auxiliary curves for NIST P-256.

3 Auxiliary Curve Dlog Codebook

In Section 2, we constructed cyclic auxiliary curves $\hat{E}(\mathbb{F}_q)$ with a generator $\hat{P}$ of smooth order $\prod_{i=1}^{n} p_i^{e_i}$. Since $\hat{P}$ generates $\hat{E}(\mathbb{F}_q)$, the point

$$\hat{P}_i := \left(\frac{\operatorname{ord}(\hat{P})}{p_i^{e_i}} \right) \hat{P}$$

generates the subgroup of order $p_i^{e_i}$. The idea of the Silver-Pohlig-Hellman algorithm is to determine the dlog in all these subgroups, and then to combine the results via Chinese Remaindering.

In order to quickly determine dlogs on $\hat{E}(\mathbb{F}_q)$ in the desired subgroups, we precompute

$$v_i \cdot \hat{P}_i \text{ for all values } 1 \leq i \leq n \text{ and } 0 \leq v_i \leq \frac{p_i^{e_i}}{2}. \tag{4}$$

Notice that on elliptic curves it suffices to compute $v_i \hat{P}_i$ for $v_i \leq \frac{p_i^{e_i}}{2}$. Assume that $v_i \geq \frac{p_i^{e_i}}{2}$. Then

$$v_i\hat{P}_i = -\left(p_i^{e_i} - v_i\right) \cdot \hat{P}_i \text{ with } p_i^{e_i} - v_i \leq \frac{p_i^{e_i}}{2}.$$

Thus, if $v_i\hat{P}_i = (x, y)$ then $(p_i^{e_i} - v_i)\hat{P}_i = -v_i\hat{P}_i = (x, -y)$.
Let $\hat{Q} = u\hat{P}$ be a dlog instance on our auxiliary curve. We compute

$$\left(\frac{\operatorname{ord}(\hat{P})}{p_i^{e_i}}\right)\hat{Q} = u\hat{P}_i = \left(u \bmod p_i^{e_i}\right)\hat{P}_i. \tag{5}$$

Assume that we store all values of $(v_i \cdot \hat{P}_i, v_i)$ from Eq. (4) in a codebook C_i. Then we simply compute the point $\left(\frac{\operatorname{ord}(\hat{P})}{p_i^{e_i}}\right)\hat{Q}$, and search for the corresponding point in the first entry $(v_i \cdot \hat{P}_i, v_i)$ of C_i. This reveals $\hat{Q}$'s dlog $u_i := v_i = u \bmod p_i^{e_i}$ in the subgroup of order $p_i^{e_i}$.

Lifting to Implicit Representation. Recall that in Maurer's algorithm, we obtain the dlog instance $\hat{Q} = u\hat{P} = [kP, yP]$ in *implicit representation* only. Therefore, we should also lift our codebook to implicit representation to allow for a simple dlog search, as before. To this end, we define the *implicit embedding*

$$\ell : \hat{E}(\mathbb{F}_q) \to E(\mathbb{F}_p),\ (x, y) \mapsto xP = (x_P, y_P) \tag{6}$$

that takes a point (x, y) on the auxiliary curve and computes the implicit representation $xP = (x_P, y_P)$ of x on the base curve. Recall that Maurer's reduction embeds the dlog k in the x-coordinate of $\hat{Q}$, only. Therefore, our implicit embedding ignores the y-coordinate.

Thus, instead of storing all *explicit* points $v_i\hat{P}_i = (x, y)$ in codebook C_i, we store their implicit embedding $\ell(v_i\hat{P}_i) = (x_P, y_P) \in E(\mathbb{F}_p)$. By the elliptic curve equation we have

$$y_P = \pm\sqrt{x_P^3 + Ax_P + b} \bmod p.$$

Since exactly one of the two values $\pm\sqrt{x_P^3 + Ax_P + b} \bmod p$ is smaller than $p/2$, we define the function

$$\operatorname{sign} : \mathbb{F}_p \to \{0, 1\},\ y_P \mapsto \begin{cases} 0 & \text{if } y_P < p/2 \\ 1 & \text{else} \end{cases}. \tag{7}$$

This enables us to compactly store (x_P, y_P) as $(x_P, \operatorname{sign}(y_P))$.

The resulting auxiliary curve dlog codebook generation algorithm is summarized in Algorithm 2.

Remark 1 (Affine vs projective coordinates). Throughout this work, for ease of exposition and for improved readability we use *affine coordinates* for all high-level descriptions of our algorithms, such as in Algorithm 2. Our implementation of these algorithms however uses *projective coordinates.* The reason for projective coordinates becomes clear in Subsection 4.1, when we show that the inversion-free elliptic curve projective coordinate doubling and addition formulas of Cohen, Miyaji, Ono [CMO98] provide benefits for calculating with implicit representations.

Using Algorithm 2, we explicitly computed the codebooks C_i for all prime factors of our 37-bit smooth auxiliary curve $\hat{E}(\mathbb{F}_q)$ for NIST P-256. The results are depicted in Table 2. In total, we obtain a memory requirement of 3.0 TByte, easily fitting on our hard disk. This computation took roughly one week on a single machine with two AMD EPYC 7742 (2.25GHz), but the algorithm is trivially distributable over multiple machines.

We slightly deviate from the description of Algorithm 2 by grouping the three smallest factors 2, 3, and 626663 into a single codebook. We elaborate in Subsection 4.2 why and how the combination of prime factors is favorable.

Algorithm 2 Auxiliary Curve Dlog Codebook

```
Input:  $E(\mathbb{F}_p)$ with generator $P$,
        $q = \mathrm{ord}\,(P)$,
        $\hat{E}(\mathbb{F}_q)$ with generator $\hat{P}$,
        factorization of $\mathrm{ord}(\hat{P}) = \prod_i^n p_i^{e_i}$
Output: Codebooks $C_i$, $i = 1, \ldots, n$ for subgroups of order $p_i^{e_i}$
 1: for $i = 1$ to $n$ do
 2:     $C_i = \emptyset$
 3:     $\hat{P}_i \leftarrow \left(\frac{\mathrm{ord}\,\hat{P}}{p_i^{e_i}}\right)\hat{P}$    ▷ $\mathrm{ord}(\hat{P}_i) = p_i^{e_i}$
 4:     $\hat{R} \leftarrow \mathcal{O}$    ▷ Initialize $\hat{R} = 0\hat{P}_i$, invariant: $\hat{R} = v_i\hat{P}_i$.
 5:     for $v_i = 0$ to $\lfloor \frac{p_i^{e_i}}{2} \rfloor$ do
 6:         $(x_P, y_P) \leftarrow \ell(\hat{R})$    ▷ implicit embedding, see Eq. (6)
 7:         $C_i \leftarrow C_i \cup \{(x_P, \mathrm{sign}(y_P), v_i)\}$    ▷ store implicit representation/dlog
 8:         $\hat{R} \leftarrow \hat{R} + \hat{P}_i$
 9:     end for
10:     Return $C_i$, sorted by first entry $(x_P, \mathrm{sign}(y_P))$.
11: end for
```

Table 2: The factors p_i, their binary length and the resulting codebook sizes $|C_i|$ for NIST P-256's auxiliary curve.

Factor p_i	$\lceil \log_2 p_i \rceil$	$|C_i|$ (GB)
$2 \cdot 3 \cdot 626663$	21	0.07
6487813	22	0.12
17752487	24	0.33
30034813	24	0.56
620378903	29	11.48
1316356273	30	24.35
4747815593	32	90.21
17399156003	34	330.58
131964961211	36	2507.33

4 Dlog Computation on the Base Curve

Let $Q = kP$ with $k \in \mathbb{Z}_q$ be the dlog instance on the base curve. For simplicity, we assumed so far that $\hat{Q} = [kP, yP]$ is an implicit representation of a point (k, y) on the auxiliary curve $\hat{E}(\mathbb{F}_q)$. To this end, we have to ensure that k is a valid x-coordinate on $\hat{E}(\mathbb{F}_q)$. In other words, let $y^2 = x^3 + Ax + B$ be our auxiliary curve equation, then $\beta := k^3 + Ak + B$ has to be a square (of some y) in $\mathbb{F}_q$. The term β is a square iff its Legendre symbol satisfies

$$\left(\frac{\beta}{q}\right) := \beta^{\frac{q-1}{2}} = 1.$$

Two problems remain. First, what happens if β is not a square, which occurs with probability roughly $\frac{1}{2}$. Second, since we do not know k explicitly, we have to show that all computations can be performed with implicit representations. We address both problems in the following.

Implicit Embedding of k into Our Auxiliary Curve. Choose some uniformly random $r \in \mathbb{Z}_q$. Then

$$xP := Q + rP = (k + r)P \tag{8}$$

has rerandomized dlog $k + r$, from which we easily derive our desired k.

We now have to check that x can indeed serve (implicitly) as an x-coordinate on $\hat{E}(\mathbb{F}_q)$. To this end, with the help of our DH oracle we compute

$$\beta P := DH(xP, DH(xP, xP)) + A \cdot xP + B = (x^3 + Ax + B)P. \tag{9}$$

Next, we check the Legendre symbol (implicitly) via

$$\beta^{\frac{q-1}{2}} P \overset{?}{=} P. \tag{10}$$

The left-hand side is computed using $\mathcal{O}(\log q)$ DH oracle calls. If this identity holds, we know that β is a quadratic residue, i.e., there exists some square root $y \in \mathbb{F}_q$ satisfying the auxiliary elliptic curve equation $y^2 = \beta = x^3 + Ax + B \bmod q$. Otherwise, we rerandomize k again using Eq. (8).

Let β be a quadratic residue in $\mathbb{F}_q$. In the case $q = 3 \bmod 4$, we have $y = \pm\beta^{\frac{q+1}{4}} \bmod q$. Notice that both square roots work for our purpose, so we choose $y = \beta^{\frac{q+1}{4}}$, and compute y implicitly as

$$yP = \beta^{\frac{q+1}{4}} P, \tag{11}$$

again using $\mathcal{O}(\log q)$ DH oracle calls. Eq. (8) and Eq. (11) together define our point $\hat{Q} = [xP, yP]$ on the auxiliary curve in implicit representation.

In the case $q = 1 \bmod 4$, we compute a square root with Cipolla's algorithm [Cip03], which also allows for implicit computation of y as yP.

Dlog extraction of k. Let $\hat{Q} = [xP, yP] = u\hat{P}_i$ be our lifted dlog instance in implicit representation. $\hat{P}_i = \left(\frac{\text{ord}(\hat{P})}{p_i^{e_i}}\right)\hat{P}$ is a generator of the subgroup of order $p_i^{e_i}$. We recap from Eq. (5) that

$$\left(\frac{\text{ord}(\hat{P})}{p_i^{e_i}}\right)\hat{Q} = (u \bmod p_i^{e_i})\,\hat{P}_i. \tag{12}$$

A computation of the left-hand side of Eq. (12) in implicit representation, and a comparison with the codebook C_i from Algorithm 2 reveals the value $u_i := u \bmod p_i^{e_i}$.

The resulting dlog computation of u, and therefore also the dlog computation of k, is summarized in Algorithm 3.

In the subsequent Subsection 4.1 we detail how to perform the computation of $\left(\frac{\text{ord}\,\hat{P}}{p_i^{e_i}}\right)\hat{Q}$ in line 9 of Algorithm 3, and to which extent this computation requires DH oracle calls.

In Subsection 4.2 we then show how to minimize the required number of DH oracle calls in line 9.

4.1 How to Compute with Implicit Representations

We have to compute $\left(\frac{\text{ord}\,\hat{P}}{p_i^{e_i}}\right)\hat{Q}$ for all $i = 1, \ldots, n$, where $\hat{Q} = [kP, yP]$ is in implicit representation.

Let $c_i = \left(\frac{\text{ord}\,\hat{P}}{p_i^{e_i}}\right)$ with binary representation $c_i = \sum_{j=1}^{m} c_{i,j}2^j$, $c_{i,j} \in \{0, 1\}$. Then

$$c_i\hat{Q} = \sum_{j=1}^{m} c_{i,j}(2^j\hat{Q}) = \sum_{\substack{1 \le j \le m,\\ c_{i,j} \neq 0}} 2^j\hat{Q}. \tag{13}$$

Thus, we compute $2^j\hat{Q}$ for all j with $2^j \le \frac{\text{ord}\,\hat{P}}{\min_i\{p_i^{e_i}\}}$. These values are precomputed once, and commonly used for the computation of all $c_i\hat{Q}$. This requires a *point doubling*

Algorithm 3 Dlog Computation on the Base Curve

```
Input:  Base curve $E(\mathbb{F}_p)$ with generator $P$ of order $q$,
        dlog instance $Q = kP$,
        DH oracle for $E(\mathbb{F}_p)$,
        Auxiliary curve $\hat{E}(\mathbb{F}_q)$ with equation $y^2 = x^3 + Ax + b$ and
        Generator $\hat{P}$ of order $\prod_{i=1}^{n} p_i^{e_i}$,
        Codebooks $C_i$ for dlogs in order-$p_i^{e_i}$ subgroups of $\hat{E}(\mathbb{F}_q)$.
Output: $k \in \mathbb{Z}_q$
 1: repeat
 2:     Choose $r \in \mathbb{Z}_q$ uniformly at random
 3:     $xP \leftarrow Q + rP$                                  ▷ $xP = (k + r)P$
 4:     $\beta P \leftarrow (x^3 + Ax + B)P$                    ▷ Eq. (10), DH oracle
 5: until $\beta^{\frac{q-1}{2}} P = P$                        ▷ $\beta$ square in $\mathbb{F}_q$? DH oracle
 6: $yP \leftarrow \sqrt{\beta}P$                              ▷ Eq. (11) or Cipolla, DH oracle
 7: $\hat{Q} \leftarrow [xP, yP]$                              ▷ Implicit Representation of $\hat{Q} = u\hat{P}$
 8: for $i = 1$ to $n$ do
 9:     $[x_iP, y_iP] \leftarrow \left(\frac{\operatorname{ord} \hat{P}}{p_i^{e_i}}\right) \hat{Q}$   ▷ DH oracle
10:     Denote $x_iP = (x_P, y_P) \in \mathbb{F}_p \times \mathbb{F}_p$.
11:     Search entry $\{(x_P, \operatorname{sign}(y_P)), u_i\}$ in codebook $C_i$.   ▷ $u_i = u \bmod p_i^{e_i}$
12: end for
13: $u \leftarrow \operatorname{CRT}(u_1, \ldots, u_n) \in \mathbb{Z}_{\operatorname{ord}(\hat{P})}$
14: $\hat{Q} \leftarrow u\hat{P}$
15: Let $\hat{Q} = (x, y) \in \mathbb{F}_q \times \mathbb{F}_q$.
16: return $k = x - r$
```

procedure. Subsequently, we show how to realize the sum computation in Eq. (13) via some *point addition* procedure.

At this point we change to projective coordinates for $\hat{E}(\mathbb{F}_q)$, since projective coordinates allow defining point doubling and addition without costly inversions in $\mathbb{F}_q$.

Let $\hat{Q} = [xP, yP, zP]$. Before we turn our attention to point doubling and addition, let us first show to work with the individual coordinates of $\hat{Q}$. Let us take xP as an example.

Elementary operations on xP. Let us start with *scalar multiplication*, i.e., we want to transform $x \mapsto \alpha x$ for some constant $\alpha \in \mathbb{F}_q$, which is the same as a scalar multiplication in $E(\mathbb{F}_q)$. To this end, we multiply the implicit representation xP on $E(\mathbb{F}_p)$ with α, resulting in the desired

$$\alpha(xP) = \alpha xP = (\alpha x)P.$$

Negation $x \mapsto -x$ is the special case $\alpha = -1$.
Addition $(x_1, x_2) \mapsto x_1 + x_2$ is realized from x_1P, x_2P by elliptic curve point addition on $E(\mathbb{F}_p)$ as

$$x_1P + x_2P = (x_1 + x_2)P.$$

Multiplication $(x_1, x_2) \mapsto x_1x_2$ however is nothing but the application of a DH-oracle on $E(\mathbb{F}_p)$, since

$$\operatorname{DH}(x_1P, x_2P) = (x_1x_2)P.$$

Squaring is the special case $x_1 = x_2$.

All operations are summarized in Table 3.

Table 3: Arithmetic operations in $\mathbb{F}_q$ can be represented by operations on their implicit representation as elements on $E(\mathbb{F}_p)$.

$\mathbb{F}_q$ operation	$E(\mathbb{F}_p)$ operation
αx_1, α constant	$\alpha(x_1 P) = \alpha x_1 P = (\alpha x_1)P$
$x_1 + x_2$	$x_1 P + x_2 P = (x_1 + x_2)P$
$x_1 x_2$	$\mathrm{DH}(x_1 P, x_2 P) = x_1 x_2 P$

Point Doubling. Let $y^2 = x^3 + Ax + B$ be the curve equation for our auxiliary curve $\hat{E}(\mathbb{F}_q)$. Assume for simplicity first that we want to double $Q = (x_1, y_1, z_1)$ given in explicit, projective coordinates. We use the doubling formula of Cohen, Miyaji, Ono [CMO98] that computes $2Q = (x_2, y_2, z_2)$ as

$$x_2 = 2hs, y_2 = w(4b - h) - 8t^2, \text{ and } z_2 = 8s^3, \tag{14}$$

where s, t, b, w, h are defined as

$$s := y_1 z_1, t = y_1 s, b := x_1 t, w := A z_1^2 + 3x_1^2, h := w^2 - 8b.$$

Ignoring scalar multiplications, the computations of s, t, b, h, x_2 require a single multiplication in $\mathbb{F}_q$, whereas the computations of w, y_2, z_2 require two multiplications, each. Thus, point doubling can be realized with a total of $5 \cdot 1 + 3 \cdot 2 = 11$ multiplications.

As a consequence, an application of the doubling formula (14) to $Q = [x_1 P, y_1 P, z_1 P]$ using the arithmetic from Table 3 in order to compute $2Q = [x_2 P, y_2 P, z_2 P]$ requires a total of 11 DH-oracle applications.

Point Addition. Again, let us first assume for simplicity that we want to add $Q_1 = (x_1, y_1, z_1)$ and $Q_2 = (x_2, y_2, z_2)$. The addition formula of Cohen, Miyaji, Ono [CMO98] computes $Q_1 + Q_2 = (x_3, y_3, z_3)$ as

$$x_3 = va, y_3 = u(h_5 - a) - h_3 h_0, \text{ and } z_3 = h_3 h_4, \tag{15}$$

where $h_0, u, h_1, v, h_2, h_3, h_4, h_5, a$ are defined as

$$\begin{aligned} &h_0 = y_1 z_2, u := y_2 z_1 - h_0, h_1 := x_1 z_2, v := x_2 z_1 - h_1, h_2 := v^2, h_3 := v h_2, \\ &h_4 := z_1 z_2, h_5 := h_2 h_1, a := u^2 h_4 - h_3 - 2h_5. \end{aligned}$$

Ignoring scalar multiplications, the computations of $x_3, z_3, h_0, u, h_1, v, h_2, h_3, h_4, h_5$ require a single multiplication in $\mathbb{F}_q$, whereas the computations of y_3, a require two multiplications, each. Thus, point addition can be realized with a total of $10 \cdot 1 + 2 \cdot 2 = 14$ multiplications.

As a consequence, an application of the addition formula (15) to $Q_1 = [x_1 P, y_1 P, z_1 P]$ and $Q_2 = [x_2 P, y_2 P, z_2 P]$ using the arithmetic from Table 3 in order to compute $Q_1 + Q_2 = [x_3 P, y_3 P, z_3 P]$ requires a total of 14 DH-oracle applications.

4.2 Optimizing Oracle Calls by Prime Factor Pooling

From Subsection 4.1 we know that the majority of DH oracle calls of Algorithm 3 is consumed in line 9, where we compute $\left(\frac{\operatorname{ord} \hat{P}}{p_i^{e_i}}\right) \hat{Q}$. By Eq. (13) these DH oracle calls can be split into the following two steps.

(1) Precomputation of $2^j \hat{Q}$ for all 2^j with $2^j \leq \frac{\operatorname{ord} \hat{P}}{\min_i \{p_i^{e_i}\}}$.

(2) Computation of all $\left(\frac{\operatorname{ord} \hat{P}}{p_i^{e_i}}\right) \hat{Q}$ via Eq. (13) for all prime powers.

While step (1) is performed only once, step (2) is carried out for all prime powers. However, for the Pohlig-Silver-Hellman algorithm it is not strictly necessary to compute modulo all prime powers of the order. Chinese Remaindering only requires that all divisors of the order are coprime. Thus, we can freely pool prime powers into larger divisors in order to save on DH-oracle calls. We performed such a prime power pooling to minimize the number of oracle calls with the following three constraints, sorted by descending priority.

Memory preserving. Our prime power pooling should not produce a divisor larger than $\max_i\{p_i^{e_i}\}$. Notice that the size of our codebook computed in Section 3 mainly depends on the parameter $\max_i\{p_i^{e_i}\}$. Thus, our pooling should not come at the cost of a significant memory increase.

Minimize number of divisors. We pool prime powers such that the number of pools, i.e., the number of divisors, becomes minimal. This minimizes the number of iterations of line 9 in Algorithm 3.

Maximize smallest divisor. Lastly, we maximize the smallest divisor, which minimizes the number of DH oracle calls required during precomputation.

Let us first provide as an especially simple example the pooling of the prime factors of NIST P-256's auxiliary curve, all other optimizations can be found in Appendix A, where we put the pooled prime factors into parentheses.

For NIST P-256 we only pooled the primes 2, 3 and 626663 into a single divisor $2 \cdot 3 \cdot 626663$. This simple optimization already saves 220 oracle calls during precomputation, and another $4,868$ oracle calls in line 9, thereby reducing the total number of DH-oracles calls from $28,524$ down to $23,656$. For other auxiliary curves with larger pools we achieved even more significant savings.

For instance, for secp256k1 we pooled the prime factors $\left(2^2 \cdot 2683 \cdot 81197\right)$, $(7 \cdot 189270023)$, $(3 \cdot 59 \cdot 8313647)$, $\left(5^2 \cdot 4787 \cdot 16451\right)$, and $(41 \cdot 4937 \cdot 12577)$, resulting in 8 instead of 17 divisors. This saves 308 oracle calls during precomputation, and another $20,377$ oracle calls in line 9, thereby reducing the total number of DH-oracles calls from $42,554$ down to $21,868$.

4.3 Results: Dlog Computations

The results of our implementation are summarized in Table 4.

Column B denotes the smoothness in bits of the auxiliary curves' orders that we computed in Section 2. For the (small) 204-bit Anomalous curve we computed an auxiliary curves with 33 bit smoothness. For all curves with group sizes of maximal 256 bits we achieved to compute auxiliary curves with smoothness between 36 and 39 bits.

Notice that the smoothness directly affects the required codebook size that we computed in Section 3. For the Anomalous curve we only need a codebook of size less than 0.2 TByte. With 37-bit smoothness we obtain codebook sizes in the range of 3.0–4.1 TByte, with 38-bit smoothness we require 7.3 TByte, whereas 39-bit smoothness implies codebook sizes of up to 28.2 TByte.

More precisely, the codebook sizes depend on the full prime power factorization of our auxiliary curves' group orders. Notice that one can calculate the required codebook size directly from the factorization. However, to experimentally verify our calculations and to demonstrate the practicality of our achievements, we explicitly constructed the 3.0 TByte Codebook for NIST P-256 (therefore marked **bold** in Table 4).

Column DH-calls provides the number of required DH-oracle calls for a single dlog computation in Algorithm 3. This number heavily depends on whether our pooling strategy from Subsection 4.2 succeeds in balancing the size of the divisors of the group order of our

Table 4: Summary of our results. All data can be computed from our auxiliary curves only, but we experimentally verified the data for NIST P-256, including the 3.0 TByte codebook construction, and concrete dlog computations.

Curve	**q**	**B**	**Codebook**	**DH-calls**	**Time**
	[bit]	**[bit]**	**[TB]**	**[no.]**	**[s]**
ANSSIFRP256v1	256	39	7.4	20227	27
Anomalous	204	33	0.2	12308	17
BLS12-381	255	36	1.9	20397	28
BN(2,254)	254	39	6.0	22036	30
Curve25519	253	37	4.1	19091	26
$F_p - 256$ (GM/T 0003.2-2012)	256	39	19.4	16389	22
GOST R 34.10	256	37	3.0	17519	24
M-221	219	37	3.7	13648	18
NIST P-224	224	38	7.3	16859	23
NIST P-256	**256**	**37**	**3.0**	**23656**	**32**
SM2	256	39	10.0	19592	27
brainpoolP256t1	256	39	28.2	16542	22
secp256k1	256	37	3.1	21868	30

auxiliary curve. Assume that we achieved 37-bit smoothness for a 256-bit curve. Then ideally pooling would result in a minimal number of 7 divisors, all of roughly the same size of 37 bits. The pooling of our NIST P-256 auxiliary curve leads to 9 divisors of unbalanced sizes, resulting in the maximal amount of 23.656 DH oracles. In contrast, the M-221's auxiliary curve allows for a pooling into only 7 divisors of balanced sizes, resulting in the minimal amount of only 13.648 DH oracle calls.

We used our 3.0 TByte codebook to experimentally compute dlogs for NIST P-256. To this end, the DH oracle was simulated (which can be done, since we know the dlog). The running time for a dlog search in the codebook was negligible, thus the running time basically scaled linearly with the number of DH-calls.

In our experiments, we achieved to compute dlogs in a maximum of 32 seconds (for NIST P-256) on an AMD EPYC 7742 (2.25 GHz). If we could replace our DH-oracle simulation by a real-world DH-oracle, our results imply that the dlog computation time T_{dlog} is roughly the number **DH-calls** multiplied by the time cost T_{DH} for the DH-oracle:

$$T_{dlog} \approx \textbf{DH-Calls} \cdot T_{DH}.$$

Thus, our results tightly connect the difficulty of computing discrete logarithms to the difficulty of computing DH in practice via our auxiliary curves from Appendix A, for the most commonly used elliptic curve standards of at most 256 bit.

4.4 Required Strength of DH-Oracle

Our analysis so far assumed that we have a *full* DH-oracle $\mathrm{DH}(x_1P, x_2P) = (x_1x_2)P$ for freely chosen inputs x_1P and x_2P. We are able to simulate such an oracle, because for every value x_1P that we compute in our algorithm we keep track of its discrete logarithm x_1, and by construction we also know the discrete logarithm k of $Q = kP$.

Insufficiency of Static DH. Some real-world instantiations, e.g., [MBA+21], provide a *static* DH-oracle $DH(kP, \cdot)$ that provides $DH(kP, xP)$ for a fixed, static value kP and a *single* freely chosen input xP. Such a static oracle is however not sufficient for efficiently realizing our algorithm.

As discussed in Subsection 4.2, our Algorithm 3 requires computing $c_i \cdot \hat{Q}$, where $\hat{Q} = [kP, yP]$ is in implicit representation. Since c_i can be of size up to q, we make use of a fast square-and-multiply procedure by precomputing $2^j \hat{Q}$ for all $2^j \leq c_i$, and then using Eq. (13).

As a consequence, Algorithm 3 can be realized with $\mathcal{O}(\log q)$ calls to a full DH-oracle, but requires $\mathcal{O}(q)$ calls when using a static DH-oracle, rendering our algorithm completely impractical in the static DH-oracle case.

Real World Instantiation of a DH-oracle. Our construction provides a tight reduction of the discrete logarithm problem on elliptic curves with 256-bit order to the DH problem. Such a reduction implies that DH must be hard on these curves, unless the discrete logarithm appears to be easy. This gives us a strong security guarantee for our standards, under the dlog assumption.

However, such a tight reduction always also has a cryptanalytic facet. Namely, the reduction also provides us with a very practical discrete logarithm algorithm for our elliptic curve standards in the presence of an efficient DH oracle.

We are not aware of any real-world instantiation that provides such an oracle. However, the situation might be comparable with the *Hidden Number Problem* that was originally introduced by Boneh and Venkatesan [BV96] to show via a reduction the security of certain bits of a Diffie-Hellman key. As cryptanalytic facet, the Hidden Number Problem was widely applied to attack nonce leakage in (EC-)DSA signatures [HS01, DHMP13], and many real-world side channels provided such nonce leakage [BvSY14, MSEH20, MBA+21].

Our work may stimulate the search for real-world instantiations of DH oracles.

5 Challenges for Curves with Larger Group Order

It is natural to ask whether our results also extend to elliptic curve standards with group orders significantly larger than 256 bit. To this end, we constructed an auxiliary curve for NIST P-384 with 65-bit smoothness (using 16,436 $\approx 2^{14}$ sampled curves in the construction of Section 2), and for NIST P-521 with 110-bit smoothness (using 1,860 $\approx 2^{10.9}$ samples). These auxiliary curves are provided in Appendix B, the results are summarized in Table 5.

Table 5: Results for large group orders.

Curve	q	B	Codebook	DH-calls
	[bit]	[bit]	[TB]	[no.]
NIST P-384	384	65	613, 705, 033	26129
NIST P-521	521	110	29, 923, 937, 044, 117, 456, 000, 000	34909

Several aspects are interesting when comparing Table 5 to Table 4. Naturally, we cannot achieve comparably small smoothness bounds. This is because a single auxiliary curve sample costs significantly more time to process (group order computation and full factorization), and larger group orders are less likely to split smoothly.

Performance of our large smoothness auxiliary curves. As a consequence of the larger smoothness B, the resulting codebooks would consume an excessive, today not realizable amount of memory. More concretely, the codebooks for NIST P-384 would require 614 Exabyte, and for NIST P-521 even 29924 Quettabyte.

Let us assume for a moment that we have a quickly accessible storage with 614 Exabyte capacity for storing NIST P-384's auxiliary curve codebook. Since we can split its group order into 7 divisors of balanced size (see Appendix B), we could realize dlog computations with only 26,129 DH-oracle calls. This compares well to the numbers obtained in Table 5.

Thus, with memory comparably fast as the one used in our experiments for NIST P-256's codebook we could realize dlog computations for NIST P-384 in slightly more than 30s.

Small memory variants. In order to avoid excessive memory requirements, one could think of constructing a small memory dlog algorithm like Pollard Rho that works with implicit representations. However, even if we realize such an algorithm then this would come at the cost of a largely increased number of DH-oracle calls. Asymptotically speaking, our approach has codebook memory size of roughly 2^B, but its DH-oracle calls are linear in $\log q$ and in the number of divisors. In contrast, a memory-less Pollard-type algorithm would require DH-calls in the order of $2^{B/2}$. As a consequence, such an algorithm would not tightly relate dlog to DH complexity.

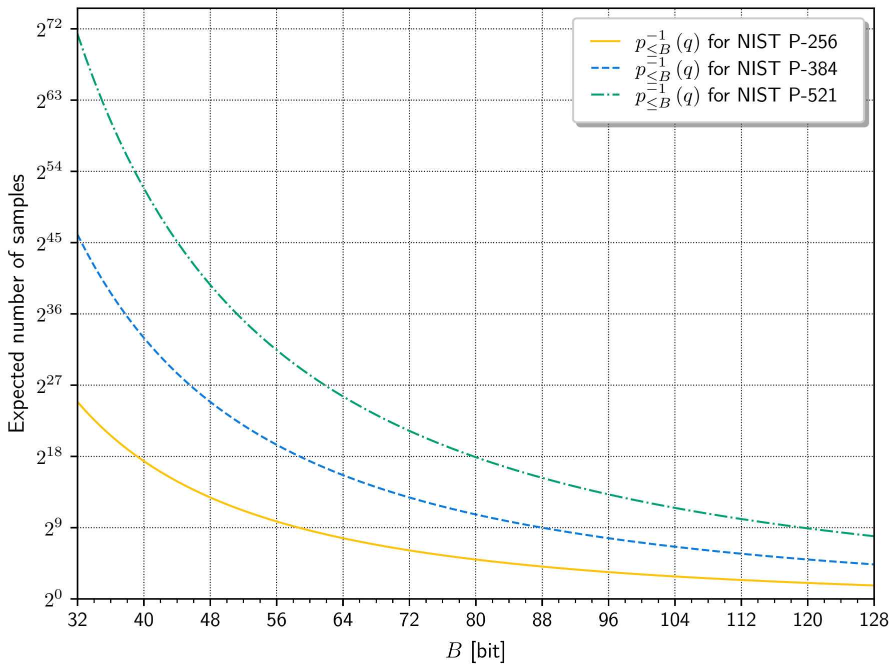

Figure 2: Expected number of samples to find an auxiliary curve with B-smooth order, for different sizes of the base curve.

5.1 Estimations for Achieving Better Smoothness

Assume that we want to run our auxiliary curve construction from Algorithm 1 until we obtain a smoothness bound $B \leq 40$, comparable to the results for 256-bit curves, thereby also establishing a tight relation between the discrete logarithm and the Diffie-Hellman problem for curves with 384-bit and 521-bit order.

We take the estimate from Subsection 2.1 for the probability $p_{\leq B}(q)$ that an auxiliary curve $\hat{E}(\mathbb{F}_q)$ has an order that is at most B-smooth. This gives an expected amount of $p_{\leq B}^{-1}(q)$ trials, until we find an at most B-smooth auxiliary curve.

Figure 2 provides the estimates of $p_{\leq B}^{-1}(q)$ for the order q of NIST P-256, NIST P-384, and NIST P-521. The estimates from Figure 2 are again in line with our results from Table 5, where we found a 65-bit smooth auxiliary curve for NIST P-384 within 2^{14} trials, and a 110-bit smooth auxiliary curve for NIST P-521 within $2^{10.9}$ trials. We see that the expected number of trials significantly increases with decreasing B, but it still appears to be feasible for actors with high-performance computing capabilities.

Table 6: Expected number of trials to reach a given smoothness bound B.

Curve	$B \leq 2^{40}$	$B \leq 2^{65}$	$B \leq 2^{80}$	$B \leq 2^{110}$
NIST P-256	$2^{17.4}$	$2^{7.5}$	$2^{5.0}$	$2^{2.5}$
NIST P-384	$2^{33.0}$	$2^{15.2}$	$2^{10.7}$	$2^{5.9}$
NIST P-521	$2^{51.8}$	$2^{25.0}$	$2^{17.9}$	$2^{10.4}$

In Table 6, we provide estimates for the expected number $p_{\leq B}^{-1}(q)$ of trials that is necessary for achieving certain smoothness levels B. For instance, for NIST P-384 we would expect to hit an auxiliary curve with at most 40-bit smoothness after 2^{33} trials. For NIST P-521 hitting a 40-bit smooth auxiliary curve would require an estimated number of $2^{51.8}$ trials. These are certainly ambitious computational efforts, but also not completely out of reach.

A List of Auxiliary Curves

Anomalous

$$A = \text{0x45ddec04e4ed7b779ac1e2864a23b561d6fbad726d249323723}$$

$$B = \text{0xa32381cbd50fc8cb48201e84f600bf85cb0536adb34bd3f5f80}$$

$$x(\hat{P}) = \text{0x9bd8400e49480fe9f22b7b6e22bf5fcd9868cf05dca5bd7ae95}$$

$$y(\hat{P}) = \text{0x91b318c596349f5ae7fe5fd5ca3d5b8637784b9e8b00ff50cc0}$$

$$\operatorname{ord} \hat{P} = (31 \cdot 1557019) \cdot (53 \cdot 1136459) \cdot \left(2^2 \cdot 3 \cdot 7 \cdot 764593\right) \cdot 1266653719 \cdot (2903 \cdot 747743) \cdot 5683625323 \cdot 6057790241$$

ANSSIFRP256v1

$$A = \text{0x7c836c3107f9f9c7fb55773b5e389f347fcecf65a5065cbc6480de1e3038a028}$$

$$B = \text{0x96c848939e57d61da7cec3c6b48db5f4c51167cb24cb60174c534888af7c2494}$$

$$x(\hat{P}) = \text{0x91e1316c2a1a3f1fc38906ce6b637c20c749b08b2d42f9fa6102968db3f0b56}$$

$$y(\hat{P}) = \text{0xdad16d846dc68b761fe8dd51aa971d49e130016c7e318953e87c938d08529bac}$$

$$\operatorname{ord} \hat{P} = \left(2^2 \cdot 23 \cdot 2339893\right) \cdot (503 \cdot 461239) \cdot (5 \cdot 52470317) \cdot 4210883441 \cdot 5780236507 \cdot 19660693177 \cdot 58041243599 \cdot 300758573363$$

BLS12-381

$$A = \text{0x41ae7764404433acbd3cbab0ca700ff2cc00f97f7f5f7251978f52618e3bd886}$$

$$B = \text{0x28a1527292dd8a91c610e416acdf282794cc817a755a8f4980ef22b324e31d2a}$$

$$x(\hat{P}) = \text{0x69e12c9f2a3f4a4ec8dd0ca29964a2ede6eccdfeb1a4b52869f6551f314bb819}$$

$$y(\hat{P}) = \text{0x5dc3d5c62045b2b1b977ee64ba96d90e75dad87ae88fd759f97b1c242fa461a}$$

$$\operatorname{ord} \hat{P} = (2 \cdot 265338611) \cdot 1158518213 \cdot \left(3^3 \cdot 277 \cdot 216761\right) \cdot (7 \cdot 7451 \cdot 34519) \cdot 3826533983 \cdot 13897244563 \cdot 15561919889 \cdot 35310370103$$

BN(2,254)

$$\begin{aligned}
A &= \mathrm{0x23080b5443a53e447cee5a56e1a93a638ae09b1918a3519545fbafa9a1f34eba}\\
B &= \mathrm{0x1a4ef8aad7675bd5594f06beac1e8a334ac71ff6b57f5f66eb43ec1c2940a0a2}\\
x(\hat{P}) &= \mathrm{0x21a844cb11b93a3a46eed06dacbfa2b1ccc293afadc5887fe5d7d48a9bf44532}\\
y(\hat{P}) &= \mathrm{0x1847b560ed162d5293a49b667e98ec03c265b0c24153b66af548430962714e79}\\
\operatorname{ord}\hat{P} &= (7\cdot 2898739)\cdot(3\cdot 10977103)\cdot\left(2^2\cdot 8383013\right)\cdot 45422513\cdot(19\cdot 2878037)\cdot(23\cdot 2590279)\cdot\\
&\quad (29917\cdot 51407)\cdot 10561842803\cdot 311895131749
\end{aligned}$$

brainpoolP256t1

$$\begin{aligned}
A &= \mathrm{0x83dce5e1fce1e7500b4830eb5ee0e8089ead4280a861a286a2f48cc2823e06b4}\\
B &= \mathrm{0x4af4bfe1ab842bb4454875290fbb897c10516cec9bfb653ab9c1e3f7d833070e}\\
x(\hat{P}) &= \mathrm{0x3ac17faa67673cf8b888816c464a5312f92eb20f8cc37ef277e8424b65ec992e}\\
y(\hat{P}) &= \mathrm{0x555134db1215b0b63e4a5c530bbd211044ef63fe1d7330c0e97907455f1e9366}\\
\operatorname{ord}\hat{P} &= (2\cdot 734197267)\cdot(197\cdot 5813\cdot 38917)\cdot 163919008373\cdot 168007838681\cdot 262438726679\cdot\\
&\quad 296370932339\cdot 548492026207
\end{aligned}$$

Curve25519

$$\begin{aligned}
A &= \mathrm{0x7c6924b558914bbfa3661a2a2a1687de21ed7b0b20b11ca4f69da9c7d797e20}\\
B &= \mathrm{0x8396db4aa76eb4405c99e5d5d5e978232c0222df0ec8b0c08a8887ddf7c55cd}\\
x(\hat{P}) &= \mathrm{0x943069e813cca7ae6e0d920ea8be9b679a64af600d8791537887e2c5173e99b}\\
y(\hat{P}) &= \mathrm{0x7d2fafc0dc0869719f6f2c9f2c65fe0dd110db31ef833bfa13282f28c11b537}\\
\operatorname{ord}\hat{P} &= 255833749\cdot\left(2^2\cdot 53\cdot 1563739\right)\cdot(3\cdot 7\cdot 1013\cdot 26339)\cdot(23\cdot 25395859)\cdot 1073269973\cdot\\
&\quad 36776837081\cdot 49009622279\cdot 134777522111
\end{aligned}$$

$F_p - 256$ (GM/T 0003.2-2012)

$$\begin{aligned}
A &= \mathrm{0x539f1a674e56855db0fbaf00cb505a7e155b2e8e3fee4c15998752eb31bf7050}\\
B &= \mathrm{0x4ff601fb2c38236ad69db8219a9a410ee6cdbbf0aa9210cdddb891fe6fddeb0b}\\
x(\hat{P}) &= \mathrm{0x24f198eba9e4efc64f6f56a30a0c194381493c18a6ba44bec3af504a07ecd959}\\
y(\hat{P}) &= \mathrm{0x300eebc808a4ae372e022d029120d0b5258191c72fd1205efeb87c4ecfb8e9bf}\\
\operatorname{ord}\hat{P} &= \left(3^2\cdot 4051573727\right)\cdot(2\cdot 5\cdot 6311\cdot 634441)\cdot 51412214251\cdot(106019\cdot 633053)\cdot\\
&\quad 93252768551\cdot 288767400343\cdot 444310543783
\end{aligned}$$

GOST R 34.10

$$\begin{aligned}
A &= \mathrm{0x11dc26d570c4f5328e738a6bf64968511d5d2356a7eec97adbebfc545f58cb89}\\
B &= \mathrm{0x7ea42914cc45b3b7391b09dca5cf29ea96ec2e166b23e76e1a1645dd56871015}\\
x(\hat{P}) &= \mathrm{0x1c7c23c623f7bfc4b587d16a6f8095f41cca51ab452e651d117a60f8d809b90f}\\
y(\hat{P}) &= \mathrm{0x5f0c5eaf878bef7ed3895b853041c2d39c0e20911b1741100d9a8fa59bb136ed}\\
\operatorname{ord}\hat{P} &= (13\cdot 29681521)\cdot 393794411\cdot\left(2^3\cdot 73276447\right)\cdot(5\cdot 233492191)\cdot 9210725213\cdot\\
&\quad 14479106177\cdot 59192041087\cdot 70526802109
\end{aligned}$$

M-221

$$\begin{aligned}
A &= \mathrm{0x11ab83a2b7b651fbd8ff1cc37e05111fd94e7874bee1c818b14fa82}\\
B &= \mathrm{0x3681d5a10403d463bdc67504242200018032c05a3c51c9b4020b5c1}\\
x(\hat{P}) &= \mathrm{0x93f693abe651831bc9c06d2374a780d1b79f1355836f6fabdb1073}\\
y(\hat{P}) &= \mathrm{0x3049587cf7356670f417a594fe429ff95f7f41a044bcadebef8e53e}\\
\operatorname{ord}\hat{P} &= \left(2^2 \cdot 163 \cdot 202309\right) \cdot 191299609 \cdot (31 \cdot 19436929) \cdot (89 \cdot 15714541) \cdot 1781291429 \cdot\\
&\quad 82270262003 \cdot 135177143687
\end{aligned}$$

NIST P-224

$$\begin{aligned}
A &= \mathrm{0x5c01766a74b8c2d862b22285b00ac178f999026de39f63c7ecdfe6a4}\\
B &= \mathrm{0x18b5ff0f9b632632db98fce082baa04f06e0f5a94cd98c8b19141fcc}\\
x(\hat{P}) &= \mathrm{0xda6600c2e2d292ac129cc267407b28721ff305987fe7903a20d32d9}\\
y(\hat{P}) &= \mathrm{0x33310ad7a6635ee685276e17bf07a6b83765d137bb45053f2497c010}\\
\operatorname{ord}\hat{P} &= (2 \cdot 67 \cdot 697201) \cdot (281 \cdot 469207) \cdot \left(7^2 \cdot 42767789\right) \cdot (8677 \cdot 408341) \cdot 6707397163 \cdot\\
&\quad 205822236209 \cdot 213517584151
\end{aligned}$$

NIST P-256

$$\begin{aligned}
A &= \mathrm{0x11c877b751dcab93a3dc546a7af6f26a4a7506a0f648d54b143b9cdbb100025a}\\
B &= \mathrm{0xee378847ae23546d5c23ab9585090d957271f40cb0cec939df7e2de74b6322f7}\\
x(\hat{P}) &= \mathrm{0xcd7ca16b05e3dd64c99da4a31cef71bdf7d48798d213a40ba4ec3a4d137bab30}\\
y(\hat{P}) &= \mathrm{0x445c8a8c21843080bf651958a5c26df5f9ad5bd73f4684d1ecb1026ec59c161f}\\
\operatorname{ord}\hat{P} &= (2 \cdot 3 \cdot 626663) \cdot 6487813 \cdot 17752487 \cdot 30034813 \cdot 620378903 \cdot 1316356273 \cdot 4747815593 \cdot\\
&\quad 17399156003 \cdot 131964961211
\end{aligned}$$

secp256k1

$$\begin{aligned}
A &= \mathrm{0xd7d03e8f857c179d91cd6a6778d0a4dc267f8f09a90c945e71e8253e9dccf81f}\\
B &= \mathrm{0x34510d92130b79e1e7abf72e664bb0f458a34b9eb4d6fad4e62a165391732348}\\
x(\hat{P}) &= \mathrm{0x301bc7b2f9e3618092ddd09909bf7088c386370d284142e454fce8c0f8188962}\\
y(\hat{P}) &= \mathrm{0x19adf7dae54fb47e3ca69efa348d2193e8b61e66dbaed5ca21b9765c0dfc6a23}\\
\operatorname{ord}\hat{P} &= \left(2^2 \cdot 2683 \cdot 81197\right) \cdot (7 \cdot 189270023) \cdot (3 \cdot 59 \cdot 8313647) \cdot \left(5^2 \cdot 4787 \cdot 16451\right)\\
&\quad \cdot (41 \cdot 4937 \cdot 12577) \cdot 2991313439 \cdot 40403184727 \cdot 112516500491
\end{aligned}$$

SM2

$$\begin{aligned}
A &= \mathrm{0x603daf21c7be756f00b06fccfac7c2de9d6a4fe8839ddc3b036f914ac669526b}\\
B &= \mathrm{0x82b5f187fe7a21c5b3956353223feb3464f7174c635a077e8958a6f38dc447b}\\
x(\hat{P}) &= \mathrm{0xe9a2490ac62b388b50fb3c69610dea0aee9b580e909b82a41b261ad6a9fe7383}\\
y(\hat{P}) &= \mathrm{0x40e4456b979905b7afca6e34b8fb84af318e31a74338ffc2394547d452ccc336}\\
\operatorname{ord}\hat{P} &= 29148461 \cdot \left(2^2 \cdot 53 \cdot 182821\right) \cdot (2099 \cdot 55717) \cdot 11753285897 \cdot 26589277361 \cdot\\
&\quad 74605790993 \cdot 133832588101 \cdot 280867123013
\end{aligned}$$

B List of Auxiliary Curves for Larger Groups

NIST P-384

$$\begin{aligned}
A &= \texttt{0x3771167d1314bed32995fad5a54b3036e08377f4eb206ff082f6f2310ab7e773b47}\\
&\quad \texttt{d52c9bde56468615d4ed779550514}\\
B &= \texttt{0x1acf5f8d531c7d9d5eac68be4ba7f9373a55d6709d0e89252dc2224bb26da44d6}\\
&\quad \texttt{779ca65afeaaf56126970684514abd5}\\
x(\hat{P}) &= \texttt{0xbf0033a63fc9b49c4c5bfdd3f49a6df96784959dba38f79ad00815dc6d01e6490d}\\
&\quad \texttt{62bfae169465f8588899da542701a2}\\
y(\hat{P}) &= \texttt{0xcd385505741daaeeb9becdf3d9d0ae088a3e0faccbcfa1c3a649cf6fb31fbf135f30}\\
&\quad \texttt{8df0ed08bb01611c0ba9d19ab29f}\\
\operatorname{ord}\hat{P} &= 3554867932881493 \cdot (17 \cdot 4431839 \cdot 49164959) \cdot (13 \cdot 199 \cdot 1759247842367)\\
&\quad \cdot (11 \cdot 151 \cdot 4369951069619) \cdot (7 \cdot 2099 \cdot 1015347659219) \cdot 291305128317199177 \cdot\\
&\quad 20843144683794886337
\end{aligned}$$

NIST P-521

$$\begin{aligned}
A &= \texttt{0x19aa7d0ff1fae51b05966f45b38bc87402ac74b9a3fc10fb24788669815efa941c1}\\
&\quad \texttt{75ea9de8fd469924197e880b27c77f4a73af58085daeb23070878cd603617afe}\\
B &= \texttt{0xf22dc68bdb0178729f4ce859afcf17391cf83c42724d9b653452f32c09985bc495e}\\
&\quad \texttt{8edfab0e75abaf2062a1e9ba14d23785552988aa1ed18c3e6e576552f231093}\\
x(\hat{P}) &= \texttt{0x1d0f9662dec76a512d9e2c4354136af2e627a5be6751c68e4a87284d9a30ccadde}\\
&\quad \texttt{276835e28d8b5c3230668a18da69dc7d4d770fdf7a0bc19bb1eefb26e8c8cb6c8}\\
y(\hat{P}) &= \texttt{0x1e9a561d48250a5e54caa73ab33a108d4e5d6ffb15676521be529a48da225ad10}\\
&\quad \texttt{445e92372f395527be317fe7add751c6fd878b9d306a485d6828167f8821c94f37}\\
\operatorname{ord}\hat{P} &= (3 \cdot 5 \cdot 541 \cdot 1738387339027138321) \cdot 11757557626983443626690433\\
&\quad \cdot 20433390575861429207316277\\
&\quad \cdot (4435900135201 \cdot 5611391852501)\\
&\quad \cdot 108773751715661588194866439 \cdot 748098266452871187238818871695923
\end{aligned}$$

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, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 5–17. ACM, 2015.

[Ben05] K. Bentahar. The equivalence between the DHP and DLP for elliptic curves used in practical applications, revisited. In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 376–391. Springer, Heidelberg, December 2005.

[BV96] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 129–142. Springer, Heidelberg, August 1996.

[BvSY14] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. "ooh aah... just a little bit": A small amount of side channel can go a long way. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 75–92. Springer, Heidelberg, September 2014.

[Cip03] Michele Cipolla. Un metodo per la risolutione della congruenza di secondo grado. *Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche, Napoli*, 9:154–163, 1903.

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer, Heidelberg, October 1998.

[DHMP13] Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 435–452. Springer, Heidelberg, August 2013.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 2012.

[GG16] Steven D. Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Des. Codes Cryptogr.*, 78(1):51–72, 2016.

[GPSV18] Steven D. Galbraith, Lorenz Panny, Benjamin Smith, and Frederik Vercauteren. Quantum equivalence of the DLP and CDHP for group actions. *CoRR*, abs/1812.09116, 2018.

[Gra08] Andrew Granville. Smooth numbers: computational number theory and beyond. *Algorithmic number theory: lattices, number fields, curves and cryptography*, 44:267–323, 2008.

[HS01] N. A. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. *Designs, Codes and Cryptography*, 23(3):283–290, August 2001.

[Kus18] Prabhat Kushwaha. Improved lower bound for diffie-hellman problem using multiplicative group of a finite field as auxiliary group. *J. Math. Cryptol.*, 12(2):101–118, 2018.

[Lab94] Computer Systems Laboratory. Federal information processing standards publication: digital signature standard (DSS). Technical report, National Institute of Standards and Technology, 1994.

[Lab13] Information Technology Laboratory. Digital signature standard (DSS). Technical report, National Institute of Standards and Technology, July 2013.

[LJ87] Hendrik W Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[Mau94] Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 271–281. Springer, Heidelberg, August 1994.

[MBA+21] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting Most-Significant-Bit-Oracles in TLS-DH(E). In *30th USENIX Security Symposium (USENIX Security 21)*, pages 213–230. USENIX Association, August 2021.

[MSEH20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2057–2073. USENIX Association, August 2020.

[MSV04] A. Muzereau, N. P. Smart, and F. Vercauteren. The Equivalence between the DHP and DLP for Elliptic Curves Used in Practical Applications. *LMS Journal of Computation and Mathematics*, 7:50–72, 2004. Publisher: Cambridge University Press.

[MW96] Ueli M. Maurer and Stefan Wolf. Diffie-Hellman oracles. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 268–282. Springer, Heidelberg, August 1996.

[MW99] Ueli M. Maurer and Stefan Wolf. The relationship between breaking the diffie–hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, 44(170):483–494, 1985.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, New York, NY, 2009.