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

Alexander May and Carl Richard Theodor Schneider

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

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

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 a dozen of mostly 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 are less than 29 TByte, 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 $E(\mathbb{F}_q)$. 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. Thus, if practical implementations unintentionally provide a DH oracle, dlog computations actually become surprisingly easy.
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 DSA signatures [Nat94,Nat13] 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.

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 mostly 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 $k$.

Let $\hat{E}(\mathbb{F}_q)$ be an auxiliary curve with a generator $\hat{P}$ of smooth order $\text{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{\varepsilon_i}$. 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 $\hat{E}(\mathbb{F}_q)$ with smooth order $\text{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{\varepsilon_i}$.

2. **Auxiliary Curve Dlog Codebook Construction.** We run the Silver-Pohlig-Hellman algorithm on the lifted dlog instance to precompute and store all values $v_i \mod p_i^{\varepsilon_i}$ for all $i$, and all $v_i \in \mathbb{Z}_{p_i^{\varepsilon_i}}$ 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 $u_i = u \mod p_i^{\varepsilon_i}$, 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

---

3 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.
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 whole 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 then eventually 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, 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 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 $\text{ord}(\hat{P}) = \prod_{i=1}^{n} p_i^{e_i}$.

Auxiliary Curve Dlog Codebook. For every $i = 1, \ldots, n$ we compute the generator $\hat{P}_i = (\text{ord}(\hat{P})/p_i^{e_i})\hat{P}$ 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 to $\hat{Q} = [kP, kP] = u\hat{P}$ on our auxiliary curve $\hat{E}(\mathbb{F}_q)$. For every $i = 1, \ldots, n$ we compute $[\text{ord}(\hat{P})/p_i^{e_i}]\hat{Q}$ in implicit representation using DH oracles for multiplication/division in $\mathbb{F}_q$. A comparison with our codebook for $\hat{P}_i$ immediately reveals $u_i = u \mod p_i^{e_i}$. We compute from $u_1, \ldots, u_n$ 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, we require for a complete dlog computation on any of the considered standardized elliptic curves with at most 256 bits 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 codebooks, and performing dlog computations is available at [https://github.com/e70847616e1d2c84/discrete-log](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 less 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 achieve to provide 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 $q^{3/4}$. 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 are summarized in Section 4.3. 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 ord$(P) = q$. We sample random auxiliary curves $\tilde{E}(\mathbb{F}_q)$ with curve equation $y^2 = x^3 + Ax + B$ by
randomly sampling $A, B \in \mathbb{F}_q$ with non-zero discriminant $4A^3 + 27B^2 \neq 0 \mod q$. Any elliptic curve $\tilde{E}(\mathbb{F}_q)$ 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 $\tilde{E}(\mathbb{F}_q)$ with Schoof’s algorithm [Sch85], together with a generator $\tilde{P}$, i.e., $\text{ord}(\tilde{P}) = |\tilde{E}(\mathbb{F}_q)|$.

We then factor $\text{ord}(\tilde{P})$ into its prime factors. The whole procedure is repeated, until we find a cyclic auxiliary curve $\tilde{E}(\mathbb{F}_q)$ 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 $\tilde{E}(\mathbb{F}_q) : y^2 = x^3 + Ax + B \mod q$, generator $\tilde{P}$ of $\tilde{E}(\mathbb{F}_q)$

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

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 within at most $10^6$ samples a cyclic auxiliary curve with at most 39-bit smooth group order.

<table>
<thead>
<tr>
<th>Base curve $E(\mathbb{F}_p)$</th>
<th>$q$ [bit]</th>
<th>Samples B [bit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomalous</td>
<td>204</td>
<td>71311</td>
</tr>
<tr>
<td>ANSSIFRP256v1</td>
<td>256</td>
<td>156841</td>
</tr>
<tr>
<td>BN(2,254)</td>
<td>254</td>
<td>7060</td>
</tr>
<tr>
<td>brainpoolP256t1</td>
<td>256</td>
<td>498440</td>
</tr>
<tr>
<td>Curve25519</td>
<td>253</td>
<td>104806</td>
</tr>
<tr>
<td>$F_p - 256$ (GM/T 0003.2-2012)</td>
<td>256</td>
<td>514595</td>
</tr>
<tr>
<td>GOST R 34.10</td>
<td>256</td>
<td>113350</td>
</tr>
<tr>
<td>M-221</td>
<td>219</td>
<td>229513</td>
</tr>
<tr>
<td>NIST P-224</td>
<td>224</td>
<td>76980</td>
</tr>
<tr>
<td>NIST P-256</td>
<td>256</td>
<td>437088</td>
</tr>
<tr>
<td>secp256k1</td>
<td>256</td>
<td>991302</td>
</tr>
<tr>
<td>SM2</td>
<td>256</td>
<td>840273</td>
</tr>
</tbody>
</table>

**Table 1.** $B$-smoothness achieved for the constructed auxiliary curves.
We detail the distribution of the largest prime factor and the number of tested auxiliary curves in Figure 1 for our NIST P-256 computations. As one would probably expect, most curve orders have one prime factors with approximately 128 bit. The probability of achieving \( B \)-smooth curves with \( B \ll 128 \) drops quite quickly. We only found very few curves with \( B < 50 \), until we eventually discovered our 37-bit smooth auxiliary curve after \( 4.4 \times 10^5 \) trials.

![Relative frequency of \( B \)-smooth auxiliary curves for NIST P-256.](image)

**Fig. 1.** Relative frequency of \( B \)-smooth auxiliary curves for NIST P-256.

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.

### 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

\[
\hat{P}_i := \left( \frac{\text{ord}(\hat{P})}{p_i^{e_i}} \cdot \hat{P} \right)
\]
generates the subgroup of order $p^e_i$. The idea of the Silver-Pohlig-Hellman 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

$$v_i \cdot \hat{P}_i \text{ for all values } 1 \leq i \leq n \text{ and } 0 \leq v_i \leq \frac{p^e_i}{2}. \quad (1)$$

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

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

Thus, if $v_i \cdot \hat{P}_i = (x, y)$ then $(p^e_i - v_i) \hat{P}_i = -v_i \cdot \hat{P}_i = (x, -y)$.

Let $\hat{Q} = u \hat{P}$ be a dlog instance on our auxiliary curve. We compute

$$\left(\frac{\text{ord}(\hat{P})}{p^e_i}\right) \hat{Q} = u \hat{P}_i = (u \mod p^e_i) \cdot \hat{P}_i. \quad (2)$$

Assume that we store all values of $(v_i \cdot \hat{P}_i, v_i)$ from Eq. (1) in a codebook $C_i$. Then we simply compute the point $\left(\frac{\text{ord}(\hat{P}_i)}{p^e_i}\right) \hat{Q}$, 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 $u_i := v_i = u \mod p^e_i$ in the subgroup of order $p^e_i$.

**Lifting to Implicit Representation.** Recall that in Maurer’s algorithm, we obtain the dlog instance $\hat{Q} = u \hat{P} = [xP, yP]$ 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*

$$\ell : \hat{E}(\mathbb{F}_q) \to E(\mathbb{F}_p), \ (x, y) \to xP = (x_P, y_P) \quad (3)$$

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 log $u$ in the $x$-coordinate of the auxiliary curve, only. Therefore, our implicit embedding ignores the $y$-coordinate.

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

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

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

$$\text{sign} : \mathbb{F}_p \to \{0, 1\}, \ y_P \mapsto \begin{cases} 0 & \text{if } y_P < p/2, \\ 1 & \text{else} \end{cases}. \quad (4)$$
This enables us to compactly store \((x_P, y_P)\) as \((x_P, \text{sign}(y_P))\).

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

**Algorithm 2** Auxiliary Curve Dlog Codebook

**Input:** \(E(F_p)\) with generator \(P\),
\[q = \text{ord}(P),\]
\(\hat{E}(F_q)\) with generator \(\hat{P}\),
factorization of \(\text{ord}(\hat{P}) = \prod p_i^{e_i}\)

**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: \(\hat{P}_i \leftarrow \left(\frac{\text{ord}(\hat{P})}{p_i^{e_i}}\right) \hat{P}\) \(\triangleright \text{ord}(\hat{P}_i) = p_i^{e_i}\)
4: \(\hat{R} \leftarrow 0\) \(\triangleright \text{Initialize } \hat{R} = 0\hat{P}_i, \text{ invariant: } \hat{R} = v_i\hat{P}_i\)
5: for \(v_i = 0\) to \(\lfloor \frac{p_i^{e_i}}{2} \rfloor\) do
6: \((x_P, y_P) \leftarrow \ell(\hat{R})\) \(\triangleright \text{implicit embedding, see Eq. } (3)\)
7: \(C_i \leftarrow C_i \cup \{(x_P, \text{sign}(y_P), v_i)\}\) \(\triangleright \text{store implicit representation/dlog}\)
8: \(\hat{R} \leftarrow \hat{R} + \hat{P}_i\)
9: end for
10: Return \(C_i\), sorted by first entry \((x_P, \text{sign}(y_P))\).
11: end for

**Remark 1 (Affine vs projective coordinates).** Throughout this work, we use for ease of exposition and for improved readability **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 Section 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 \(\hat{E}(F_q)\) 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.

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 the Section 4.2 why and how the combination of prime factors is favorable.
Dlog is Practically as Hard (or Easy) as DH

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 |
| 474781593 | 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 $\tilde{Q} = [kP, yP]$ is an implicit representation of a point $(k, y)$ on the auxiliary curve $\tilde{E}(\mathbb{F}_q)$. To this end, we have to ensure that $k$ is a valid $x$-coordinate on $\tilde{E}(\mathbb{F}_q)$. 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 $\beta$ is a square iff its Legendre symbol satisfies

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

Two problems remain. First, what happens if $\beta$ 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

$$xP := Q + rP = (k + r)P \quad (5)$$

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 $\tilde{E}(\mathbb{F}_q)$. To this end, we compute with the help of our DH oracle

$$\beta P := DH(xP, DH(xP, xP)) + A \cdot xP + B = (x^3 + Ax + B)P. \quad (6)$$

Next, we check the Legendre symbol (implicitly) via

$$\beta^{\frac{q-1}{2}} P \equiv P. \quad (7)$$

The left-hand side is computed using $O(\log q)$ DH oracle calls. If this identity holds, we know that $\beta$ is a quadratic residue, i.e., there exists some square root
\( y \in \mathbb{F}_q \) satisfying the auxiliary elliptic curve equation \( y^2 = \beta = x^3 + Ax + B \mod q \). Otherwise, we rerandomize \( k \) again using Eq. (5).

Let \( \beta \) be a quadratic residue in \( \mathbb{F}_q \). In the case \( q = 3 \mod 4 \), we have \( y = \pm \beta^{\frac{q+1}{4}} \mod q \). Notice that both square roots work for our purpose, so we choose \( y = \beta^{\frac{q+1}{4}} \), and compute \( y \) implicitly as
\[
yP = \beta^{\frac{q+1}{4}} P, \tag{8}\]
again using \( \mathcal{O}(\log q) \) DH oracle calls. Eqs. (5) and (8) together define our point \( \hat{Q} = [xP, yP] \) on the auxiliary curve in implicit representation.

In the case \( q = 1 \mod 4 \), 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. \( \hat{P}_i = \frac{\text{ord}(P)}{p_i^{e_i}} = \hat{P} \) is a generator of the subgroup of order \( p_i^{e_i} \). We recap from Eq. (2) that
\[
\left( \frac{\text{ord}(P)}{p_i^{e_i}} \right) \hat{Q} = (u \mod p_i^{e_i}) \hat{P}_i. \tag{9}\]

A computation of the left-hand side of Eq. (9) in implicit representation, and a comparison with the codebook \( C_i \) from 2 reveals the value \( u_i := u \mod p_i^{e_i} \).

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

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

In Section 4.2 we then show how to minimize the required number of DH oracle calls in line 9.
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: $xP \leftarrow Q + rP \quad \triangleright \quad xP = (k+r)P$
4: $\beta P \leftarrow (x^3 + Ax + B)P \quad \triangleright \quad \text{Eq. (7), DH oracle}$
5: until $\beta^{q-1} P = P \quad \triangleright \quad \text{2 times DH oracle}$
6: $yP \leftarrow \sqrt{\beta P} \quad \triangleright \quad \text{Eq. (8) or Cipolla, DH oracle}$
7: $\hat{Q} \leftarrow [xP, yP]$ \hspace{1cm} $\triangleright \quad \text{Implicit Representation of } \hat{Q} = u\hat{P}$
8: for $i = 1$ to $n$
9: $[x_i P, y_i P] \leftarrow \left(\frac{\text{ord } \hat{P}}{p_i^{e_i}}\right) \hat{Q}$ \hspace{1cm} $\triangleright \quad \text{DH oracle}$
10: Denote $x_i P = (x_i P, y_i P) \in \mathbb{F}_p \times \mathbb{F}_p$.
11: Search entry $(x_i P, \text{sign}(y_i P)), u_i)$ in codebook $C_i$. \hspace{1cm} $\triangleright \quad u_i = u \mod p_i^{e_i}$
12: end for
13: $u \leftarrow \text{CRT}(u_1, \ldots, u_n) \in \mathbb{Z}_{\text{ord}(\hat{P})}$
14: $\hat{Q} \leftarrow u\hat{P}$
15: Let $\hat{Q} = (x, y) \in \mathbb{F}_q \times \mathbb{F}_q$.
16: return $k = x - r$

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

$$c_i \hat{Q} = \sum_{j=1}^{m} c_{i,j} (2^j \hat{Q}) = \sum_{1 \leq j \leq m, \; c_{i,j} \neq 0} 2^j \hat{Q}. \quad (10)$$

Thus, we compute $2^j \hat{Q}$ for all $j$ with $2^j \leq \frac{\text{ord } \hat{P}}{\min_i \{p_i^{e_i}\}}$. These values are precomputed once, and commonly used for the computation of all $c_i \hat{Q}$. This requires a point doubling procedure. Subsequently, we show to realize the sum computation in Eq. (10) 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 manipulate each coordinate 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 $\alpha \in \mathbb{F}_q$. To this end, we multiply the implicit representation $xP$ on $E(\mathbb{F}_p)$ with $\alpha$, resulting in the desired

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

Inversion $x \mapsto -x$ 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

$$DH(x_1P, x_2P) = (x_1x_2)P.$$ 

Squaring is the special case $x_1 = x_2$.

All operations are summarized in Table 3.

<table>
<thead>
<tr>
<th>$\mathbb{F}_q$ operation</th>
<th>$E(\mathbb{F}_p)$ operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha x_1, \alpha$ constant</td>
<td>$\alpha(x_1P) = \alpha x_1P = (\alpha x_1)P$</td>
</tr>
<tr>
<td>$x_1 + x_2$</td>
<td>$x_1P + x_2P = (x_1 + x_2)P$</td>
</tr>
<tr>
<td>$x_1x_2$</td>
<td>$DH(x_1P, x_2P) = x_1x_2P$</td>
</tr>
</tbody>
</table>

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

Point Doubling. Let $y^2 = x^3 + Ax + B$ be the curve equation for our auxiliary curve $E(\mathbb{F}_q)$. 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

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

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

$$s := y_1z_1, t := y_1s, b := x_1t, w := A_2^2 + 3x_1^2, h := w^2 - 8b.$$ 

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 of (11) to $Q = [x_1P, y_1P, z_1P]$ using the arithmetic from Table 3 in order to compute $2Q = [x_2P, y_2P, z_2P]$ requires a total of 11 DH-oracle applications.
Point Addition. Again, let us first for simplicity assume 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

$$x_3 = va, y_3 = u(h_5 - a) - h_3h_0, \text{ and } z_3 = h_3h_4,$$

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

$$h_0 = y_1z_2, u := y_2z_1 - h_0, h_1 := x_1z_2, v := x_2z_1 - h_1, h_2 := v^2, h_3 := vh_2, h_4 := z_1z_2, h_5 := h_2h_1, a := u^2h_4 - h_3 - 2h_5.$$

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 of (12) to $Q_1 = [x_1P, y_1P, z_1P]$ and $Q_2 = [x_2P, y_2P, z_2P]$ using the arithmetic from Table 3 in order to compute $Q_1 + Q_2 = [x_3P, y_3P, z_3P]$ requires a total of 14 DH-oracle applications.

4.2 Optimizing Oracle Calls by Prime Factor Pooling

From Section 4.1 we know that the majority of DH oracle calls of Algorithm 3 is consumed in line 9 where we compute $\left( \frac{\text{ord} \hat{P}}{p_i^e} \right) \hat{Q}$. By Eq. (10) these DH oracle calls can be split into the following two steps.

1. Precomputation of $2^i \hat{Q}$ for all $2^i$ with $2^i \leq \text{ord} \hat{P}$.
2. Computation of all $\left( \frac{\text{ord} \hat{P}}{p_i^e} \right) \hat{Q}$ via Eq. (10) 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 $2^2 \cdot 2683 \cdot 81197$, $(7 \cdot 189270023)$, $(3 \cdot 59 \cdot 8313647)$, $(5^2 \cdot 4787 \cdot 16451)$, 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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSSIFRP256v1</td>
<td>256</td>
<td>39</td>
<td>7.4</td>
<td>20227</td>
<td>27</td>
</tr>
<tr>
<td>Anomalous</td>
<td>204</td>
<td>33</td>
<td>0.2</td>
<td>12308</td>
<td>17</td>
</tr>
<tr>
<td>BN(2,254)</td>
<td>254</td>
<td>39</td>
<td>6.0</td>
<td>22036</td>
<td>30</td>
</tr>
<tr>
<td>Curve25519</td>
<td>253</td>
<td>37</td>
<td>4.1</td>
<td>19091</td>
<td>26</td>
</tr>
<tr>
<td>$F_p - 256$ (GM/T 0003.2-2012)</td>
<td>256</td>
<td>39</td>
<td>19.4</td>
<td>16389</td>
<td>22</td>
</tr>
<tr>
<td>GOST R 34.10</td>
<td>256</td>
<td>37</td>
<td>3.0</td>
<td>17519</td>
<td>24</td>
</tr>
<tr>
<td>M-221</td>
<td>219</td>
<td>37</td>
<td>3.7</td>
<td>13648</td>
<td>18</td>
</tr>
<tr>
<td>NIST P-224</td>
<td>224</td>
<td>38</td>
<td>7.3</td>
<td>16859</td>
<td>23</td>
</tr>
<tr>
<td>NIST P-256</td>
<td>256</td>
<td>37</td>
<td><strong>3.0</strong></td>
<td><strong>23656</strong></td>
<td><strong>32</strong></td>
</tr>
<tr>
<td>SM2</td>
<td>256</td>
<td>39</td>
<td>10.0</td>
<td>19592</td>
<td>27</td>
</tr>
<tr>
<td>brainpoolP256t1</td>
<td>256</td>
<td>37</td>
<td>28.2</td>
<td>16542</td>
<td>22</td>
</tr>
<tr>
<td>secp256k1</td>
<td>256</td>
<td>37</td>
<td>3.1</td>
<td>21868</td>
<td>30</td>
</tr>
</tbody>
</table>

**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.

*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 37 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 Figure 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 Section 4.2 succeeds in balancing the size of the divisors of the group order of our 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-calls were 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). 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 $\text{DH-calls}$ multiplied by the time cost $T_{DH}$ for the DH-oracle:

$$T_{dlog} \approx \text{DH-calls} \cdot T_{DH}.$$  

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 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 sampled curves in the construction of Section 2), and for NIST P-521 with 110-bit smoothness (using 1,860 samples). These auxiliary curves are provided in section B the results are summarized in Table 5.

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.

As a consequence of the larger smoothness $B$, the resulting codebooks would consume an excessive, today not realizable amount of memory. More concretely, the codebook 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 amount of 614 Exabyte for storing NIST P-384’s auxiliary curve codebook. Since we can split its group order into 7 divisors of balanced size (see section 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 a comparably fast memory — as 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.

We leave it as an open problem whether the $dlog$ problem can be tightly related to DH also for elliptic curve standards with order 384 bit and beyond, practically realizable.

### A List of Auxiliary Curves

**Anomalous**

$$A = 0x45ddec04e4ed7b779ac1e2864a23b561d6fbad726d249323723$$

$$B = 0xa32381cbd50fc8cb48201e84f600bf85cb0536adb34bd3f580$$

$$x(P) = 0x9bd8400e49480fe9f22b7b6e22b5fcd9868cf05dca5bd7ae95$$

$$y(P) = 0x91b318c596349f5ae7fe5f5d5ca3d5b8637784b9e8b00ff50cc0$$

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

<table>
<thead>
<tr>
<th>Curve</th>
<th>$q$ [bit]</th>
<th>$B$ [bit]</th>
<th>Codebook DH-calls [TB] [no.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIST P-384</td>
<td>384</td>
<td>65</td>
<td>613,705,033.1 26129</td>
</tr>
<tr>
<td>NIST P-521</td>
<td>521</td>
<td>110</td>
<td>923,937,044,117,456,000,000 34909</td>
</tr>
</tbody>
</table>

*Table 5. Results for large group orders.*
ANSSIFRP256v1

$$A = 0x7c836c3107f9f9c7fb55773b5e389f347fcec65a50656cbe6480de1e3038a028$$
$$B = 0x96e848939e57d61da7cece3c6b48db5f451167cb24cb6017e53488a7fe2494$$

$$x(\hat{P}) = 0x91e1316c2a1a3f1fc38906c6b637c20c749b0b2d42f9fa6102968db3f0b56$$

$$y(\hat{P}) = 0xda16d846c68b761fe88d51aa971d49e130016c7e318953e87c938d08529bac$$

$$\text{ord } \hat{P} = (2^2 \cdot 23 \cdot 233983) \cdot (503 \cdot 461239) \cdot (5 \cdot 52470317) \cdot 4210883441 \cdot 5780236507 \cdot 1966093177 \cdot 58041243599 \cdot 30075857363$$

BN(2,254)

$$A = 0x23080b5443a53e447cee5a56e1a93a638ae09b1918a3519545bfa09a1f34eba$$
$$B = 0x1ade8a8ad7675bd5594f6666b6e3ec1c9290a0a2$$

$$x(\hat{P}) = 0x21a844cb11b93a3a46eed06dabef2b1ccc293dad5887e5d7d48a9b5f4532$$

$$y(\hat{P}) = 0x1847b560ed162d5293a49b667e98ec03c265b0c2415366a5f48439062714e79$$

$$\text{ord } \hat{P} = 2 \cdot 2898739 \cdot (3 \cdot 10977103) \cdot (2^2 \cdot 8383013) \cdot 45422513 \cdot (19 \cdot 2878037) \cdot (23 \cdot 2590279) \cdot (29917 \cdot 51407) \cdot 10561842803 \cdot 311895313749$$

brainpoolP256t1

$$A = 0x83dce5e1fce1e7500b4830eb5ee0e8089ead4280a861a286a2f8cc2823e6b4$$
$$B = 0xa4f48be1a842bb45457290fbb897c10516ec9b5b653ab9c3f7d833070e$$

$$x(\hat{P}) = 0x3ac17faa67673cf8b888816e464a53129f2e20f8cc37ef277e8424665ec992e$$

$$y(\hat{P}) = 0x555134db12150bb63e4a5c5308bbd211044ed63fe1d7330c0e9790755f1e9366$$

$$\text{ord } \hat{P} = 2 \cdot 734197267 \cdot (197 \cdot 5813 \cdot 38917) \cdot 163919008373 \cdot 168007838681 \cdot 262438726679 \cdot 296370932339 \cdot 548492026207$$

Curve25519

$$A = 0x7c6924b558914bbfa3661a2a2a1687de21ed7b0b2b011ca4f609da9c7d797e20$$
$$B = 0x8396db4aa76eb4405c99e5d5d5e978232c0222d00c8b0c0a88887dff7e55cd$$

$$x(\hat{P}) = 0x943069e813cca7ae6e9d920ea8be9b679a6a4f600d7891537887e2c5173e99b$$

$$y(\hat{P}) = 0x7d2fae0d809e984e69e2f29e6265fe40d110db3e833fba13282f28e11b537$$

$$\text{ord } \hat{P} = 255833749 \cdot (2^2 \cdot 53 \cdot 1563739) \cdot (3 \cdot 7 \cdot 1013 \cdot 26339) \cdot (23 \cdot 25395859) \cdot 1073269973 \cdot 36776837081 \cdot 49009622279 \cdot 134777522111$$

$$F_p = 256 \text{ (GM/T 0003.2-2012)}$$

$$A = 0x539f1a674e56855db0ftha00cb0b50a7e155b2e8e3f6ee4c150998752eb31bf7050$$
$$B = 0x4ff601fb2c38236ad69db8219a4a40ec6ebbf0aa9210cdddb389f66fdde60b$$

$$x(\hat{P}) = 0x24f198eba9e4efc64f6f56a30a1c94384993c18a6ba44be3a5f0a7f6e2995$$

$$y(\hat{P}) = 0x300eebc808a4ac372e022d0291220dd5258191c7f2d1205fe687c4efb8e96b$$

$$\text{ord } \hat{P} = (3^2 \cdot 4051573727) \cdot (2 \cdot 5 \cdot 6311 \cdot 634441) \cdot 51412214251 \cdot (106019 \cdot 633053) \cdot 93252768551 \cdot 288767400343 \cdot 444310543783$$
GOST R 34.10

\[ A = 0x11dc26d570c4f5328e738a6bf64968511d5d2356a7ec97adbebfc545f58cb89 \]
\[ B = 0x7ea42914cc45b3b7391b999c5c29ea96ec2e166123e6e1a145d5d56871015 \]
\[ x(\hat{P}) = 0xc1c7c3c623f7fbc4b587d16a6f8095f41cca51ab452e651d117a60f8d809b90f \]
\[ y(\hat{P}) = 0x50c5eaf878ebe7ee3895b853041c2d39c0e220911b1741100d9a8fa59bb136ed \]
\[ \text{ord } \hat{P} = (13 \cdot 29681521) \cdot 393794411 \cdot (2^3 \cdot 73276447) \cdot (5 \cdot 233492191) \cdot 9210725213 \cdot 14479106177 \cdot 59192041087 \cdot 70526802109 \]

M-221

\[ A = 0x11ab83a2b7b651fbd8ff1cc37e05111f94e7874beec1e818b1fa82 \]
\[ B = 0x3681d5a10403d463bdc6750424220018032c05a3c51cb90420b5c1 \]
\[ x(\hat{P}) = 0x93f693abe651831bc9c06d2374a780d179f135586f6dadb1073 \]
\[ y(\hat{P}) = 0x3049587cf7356670f417a594fe429ff95f7f41a044bcadebef8e53e \]
\[ \text{ord } \hat{P} = 2^2 \cdot 163 \cdot 202309 \cdot 191299609 \cdot (31 \cdot 19436929) \cdot (89 \cdot 15714541) \cdot 1781291429 \cdot 82270262003 \cdot 135177143867 \]

NIST P-224

\[ A = 0x5c01766a74b8c2d862b22285b00ac178f99026dc3963c7ecdef6a4 \]
\[ B = 0x18b5ff0f9b632632db98fce082baa0402c9d96600fc092b04f605a94c9de8b19411f \]
\[ x(\hat{P}) = 0xda6600c2e2d292ac129cc267407b28721ff305987e7903a20d32d9 \]
\[ y(\hat{P}) = 0x33310ad7a6635ee685276e17bf07a6b83765d137bb45053f2497c010 \]
\[ \text{ord } \hat{P} = (2 \cdot 67 \cdot 692701) \cdot (281 \cdot 469207) \cdot (2^2 \cdot 42767789) \cdot (8677 \cdot 408341) \cdot 6707397163 \cdot 205822362003 \cdot 213517584151 \]

NIST P-256

\[ A = 0x11c877b751dcab93a3dc546a7af6f26a4a7f506a0f64d54b143b9c9dbb10025a \]
\[ B = 0x3681d5a10403d463bdc6750424220018032c05a3c51cb90420b5c1 \]
\[ x(\hat{P}) = 0xcd7ca16b05e3dd64c99da4a31ce7fd7d48798d21340a4e3a3d4137bbab0 \]
\[ y(\hat{P}) = 0x4545c8a4ac21843080b6f51958a5e26d5f9ad5b3f506e1a026659c161f \]
\[ \text{ord } \hat{P} = (2 \cdot 3 \cdot 62663) \cdot 6487813 \cdot 17752487 \cdot 30034813 \cdot 620378903 \cdot 1316356273 \cdot 4747815593 \cdot 17399156003 \cdot 131964961211 \]

secp256k1

\[ A = 0xd7d03e8f857c179d91cd6a6778d0a4dc2678f09a90c94e71e8253e9dccf81f \]
\[ B = 0x301bcb72d9e3618092dd099909bf7088c386370d284142e454f68e0f8188962 \]
\[ x(\hat{P}) = 0x19ad7dace54bf4be73ca60fa348d2193c8861666d00d5ca21b9765c0d66a23 \]
\[ \text{ord } \hat{P} = (2^2 \cdot 2683 \cdot 81197) \cdot (7 \cdot 189270023) \cdot (3 \cdot 59 \cdot 8313647) \cdot (5^2 \cdot 4787 \cdot 16451) \cdot (41 \cdot 4937 \cdot 12577) \cdot 2991313439 \cdot 40403184727 \cdot 112516500491 \]
Dlog is Practically as Hard (or Easy) as DH

\[ A = 0x603da2f217bc7f6e0b6fcccfa7c2de9d6a4fe8839dc3b036f914ac669526b \]
\[ B = 0x82b5f187fe7a21c5b395635223fe3b3464f7174c635af077e8958a6f38dc447b \]
\[ x(\hat{P}) = 0xe9a2490ac62b38850f503eb84af318e31a74338ffe2394547d452ccc336 \]
\[ y(\hat{P}) = 0x40e4456b979905e7a6e34b8fb484af318e31a74338ffe2394547d452ccc336 \]
\[ \text{ord } \hat{P} = 29148461 \cdot 2^{2} \cdot 53 \cdot 182821 \cdot 2099 \cdot 55717 \cdot 11753255897 \cdot 26589277361 \cdot 7405790993 \cdot 1338325881 \cdot 280867123013 \]

B List of Auxiliary Curves for Larger Groups

NIST P-384

\[ A = 0x3771167d1314bed32995fad5a54b3036e08377f4eb206ff082f6d2310a \]
\[ B = 0x1ac5f8d531c7d95ec68be4ba7f9373a55d67090e89252dc2224bb2f6 \]
\[ x(\hat{P}) = 0xbf003a636cf9949c5bfc3f49a6df96784959dba38f79ad0815d6c9 \]
\[ y(\hat{P}) = 0xcd385505741daaeeb9becdf3d9d0ae088a3e0facbcbfa1c3a649cf6f317f \]
\[ \text{ord } \hat{P} = 3554867932881493 \cdot (17 \cdot 4431839 \cdot 4916959) \cdot (13 \cdot 199 \cdot 1759247842367) \cdot (11 \cdot 151 \cdot 4369951069619) \cdot (7 \cdot 2099 \cdot 1015347659219) \cdot 29130512831791977 \cdot 2084314468379488637 \]

NIST P-521

\[ A = 0x19aa7dfff1fae51b0596f45b38bc87402ac74b9a3fc10fb247886981f \]
\[ B = 0xf22dc68b0178729f4ec859afcf17391cf83c42724d9b63542f32c099 \]
\[ x(\hat{P}) = 0x1d0f9662dec76a512d9e2c454136af2e627a5b6751c68e4a87284d9a \]
\[ y(\hat{P}) = 0xe9a561d42850a5e54ca73a33a108d4e5d6f15676521be529a48d \]
\[ \text{ord } \hat{P} = (3 \cdot 5 \cdot 541 \cdot 173838373902713821) \cdot 1175755762983443626690433 \cdot 20433390575861429207316277 \cdot (4435900135201 \cdot 5611391852501 \cdot 108773751715661588194866439 \cdot 74809826645287118723881887169523 \]
References


