ECC's Achilles' Heel: Unveiling Weak Keys in Standardized Curves

Enrico Talotti[1], Matteo Paier[1,2] and Marino Miculan[1,3,*]

[1]University of Udine - Dept. of Mathematics, Computer Science and Physics, Italy
[2]IMT Alti Studi Lucca, Italy
[3]Ca' Foscari University of Venice - Dept. of Environmental Sciences, Informatics and Statistics, Italy

Abstract

The strength of Elliptic curve cryptography (ECC) relies on curve choice. This work analyzes *weak keys* in standardized curves, i.e., private keys within small subgroups of the auxiliary group $\mathbb{Z}_p^*$. We quantify weak key prevalence across standardized curves, revealing a potential vulnerability due to numerous small divisors in auxiliary group orders. To address this, we leverage the *implicit baby-steps giant-steps* algorithm, which transforms the complex elliptic curve discrete logarithm problem into a simpler problem within $\mathbb{Z}_p^*$. This enables efficient detection of weak keys in small-order subgroups.

Our findings highlight the importance of rigorous key testing in applications using standardized ECC. While random weak keys are unlikely, malicious actors could exploit this by manipulating key generation libraries. To this end, we show how users can assess their private key vulnerabilities and mitigate risks by eliminating weak keys. Hence, this work contributes to improved ECC security through proactive key management practices.

Keywords

Elliptic curve cryptography (ECC), Key vulnerabilities, Weak keys, Standardized curves.

1. Introduction

Elliptic curve cryptography (ECC) plays a crucial role in securing modern communication and data storage due to its efficient implementation and strong security guarantees. However, the practical effectiveness of ECC hinges on the judicious selection of appropriate curves that resist various attacks. Evaluating the robustness of standardized curves against potential weaknesses is an ongoing challenge.

This paper focuses on the vulnerability of *weak keys* in standardized elliptic curves. We define "weak keys" as private keys residing within small subgroups of the auxiliary group $\mathbb{Z}_p^*$, potentially compromising their security. Our objective is to quantify the prevalence of such keys across prevalent standardized curves and provide efficient methods for their detection.

To achieve this, we leverage the *implicit baby-step giant-step algorithm* (iBSGS) presented in [1, 2, 3]. This algorithm leverages a group action mapping elements from $\mathbb{Z}_p^*$ to the elliptic curve group, effectively transforming the complex elliptic curve discrete logarithm problem into a simpler problem within $\mathbb{Z}_p^*$. This transformation capitalizes on the well-understood subgroup structure of $\mathbb{Z}_p^*$, enabling efficient identification of weak keys belonging to small-order subgroups.

Our analysis unveils a potential vulnerability in many standardized curves, where a significant portion of private keys reside in small subgroups of the auxiliary group. This vulnerability arises due to the presence of numerous small divisors in the auxiliary group order. While the likelihood of randomly encountering a weak key remains low, malicious actors could exploit this weakness by manipulating key generation libraries employed by applications. Such manipulation could allow them to construct private keys readily recoverable from their corresponding public keys using the iBSGS algorithm. To mitigate this risk, rigorous testing of generated keys within applications becomes crucial.

Furthermore, we have implemented all relevant algorithms in Rust and PARI/GP; these implementations are available at [4]. By utilizing these tools, users can assess the vulnerability of their private

ITASEC 2024: The Italian Conference on CyberSecurity, April 08–12, 2024, Salerno, IT
*Corresponding author.
talotti.enrico.1@spes.uniud.it (E. Talotti); matteo.paier@imtlucca.it (M. Paier); marino.miculan@uniud.it (M. Miculan)
0009-0000-7588-7169 (M. Paier); 0000-0003-0755-3444 (M. Miculan)

keys, thus proactively strengthen their cryptographic security by identifying potential weak keys.

Overall, this paper contributes to the field of ECC security by offering a comprehensive analysis of weak keys in standardized curves, introducing an efficient detection method using the iBSGS algorithm, and highlighting the importance of proactive key testing.

The rest of the paper is organized as follows. In Section 2 we recall basic definitions about elliptic curves, the discrete logarithm problem, and the baby-step giant-step algorithm. The implicit version of this algorithm, dubbed iBSGS, is presented in Section 3. Section 4 covers the main contribution of this paper: first, we show how to apply the iBSGS algorithm for testing whether a key is weak; then, we analyse elliptic curves actually used in practice, providing an estimation of the weak keys which can be found within a given bound. Conclusions and directions for future work are in Section 5.

2. Preliminary Work

In this section, we lay the groundwork for our analysis by revisiting some fundamental concepts. We first recall elliptic curves; next, we describe the discrete logarithm problem in additive groups, such as those arising from elliptic curves. Finally, we recall the baby-step giant-step algorithm, a powerful tool for solving the DLP.

2.1. Elliptic Curves

An elliptic curve (EC) E over a field $\mathbb{K}$, denoted $E/\mathbb{K}$, is given by the *Weierstraß equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where the coefficients $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ are such that for each point $(x_1, y_1) \in \overline{\mathbb{K}}^2$ satisfying Equation (1), the partial derivatives do not vanish simultaneously.

We refer to points on the curve E as points with coordinates in $\overline{\mathbb{K}}$ satisfying Equation (1). Points on E with coordinates in the base field $\mathbb{K}$ form the set of $\mathbb{K}$-*rational points of* E; we denote this set by

$$E(\mathbb{K}) = \left\{(x_1, y_1) \in \mathbb{K}^2 \ : \ y_1^2 + a_1y_1 + a_3y_1 = x_1^3 + a_2x_1^2 + a_4x_1 + a_6\right\}$$

It is well-known that this set can be turned into an additive group $(E(\mathbb{K}), \oplus, \mathcal{O})$ where the group law is given by the *chord-tangent operation* $\oplus$ and the identity is given by the point at infinite $\mathcal{O}$ [5, 6, 7].

In this work we focus exclusively on elliptic curves defined over finite fields of characteristic different from 2 and 3 so we can assume that the EC is given by a short Weierstraß equation:

$$E : y^2 = x^3 + a_4x + a_6 \tag{2}$$

Moreover, we can assume the elliptic curve group $E(\mathbb{K})$ to be of order p where p is a prime. This is done for cryptographic purposes, since it increase the complexity of the Discrete Logarithm Problem.

2.2. Discrete Logarithm Problem

Let $(G, \oplus)$ be an additive cyclic group of prime order p and P a generator for G. The map

$$\begin{aligned} \varphi : \mathbb{Z} &\to G \\ n &\mapsto [n]P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{n \text{ times}} \end{aligned}$$

has kernel $p\mathbb{Z}$, thus φ leads to an isomorphism between $(G, \oplus)$ and $(\mathbb{Z}/p\mathbb{Z}, +) := \mathbb{Z}_p$. The problem of computing the inverse map is called the *discrete logarithm problem* (DLP) to the base of P. It is the problem, given P and Q, to determine $\alpha \in \mathbb{Z}$ such that $Q = [\alpha]P$. Note that α is unique only modulo the group order.

The complexity of this problem depends on the choice of G and its operation. If $G = \mathbb{Z}_p$ with generator 1, the discrete logarithm of $n \in \mathbb{Z}_p$ is n itself. If we choose a generator $a \in \mathbb{Z}_p$, where $a \neq 1$, the problem is still easy to solve because it reduces to compute the inverse of a modulo p.

2.3. Baby-Step Giant-Step Algorithm

The *baby-step giant-step algorithm* (BSGS algorithm) was firstly published by Shanks to compute ideal class numbers of quadratic number fields [8, 9]. Here we present it in a more general form and we use it to solve the DLP.

The BSGS method is based on the following:

Lemma 1. *Let n be a positive integer. Put $m := \lfloor \sqrt{n} \rfloor + 1$. Then for any α with $0 \leq \alpha < n$ there are integers i, j, with $0 \leq i, j \leq m - 1$, such that $\alpha = i + jm$.*

Proof. If we divide α by m we have $\alpha = jm + i$ with $0 \leq i \leq m - 1$. We note that $\alpha \leq n - 1 = m^2 - 1 = m(m - 1) + (m - 1)$ and we know that $i \leq m - 1$. Hence we have $0 \leq j \leq m - 1$. □

Assume now that the order of $P \in G$ is n and α an integer modulus n. Let $Q = [\alpha]P$, then we have

$$Q \oplus [-jm]P = [i]P,$$

for some i, j as in Lemma 1. We have the following:

Proposition 1. *Let $(G, \oplus)$ be a finite, additive, cyclic group of order n and let P be a generator of G. The following algorithm solves the discrete logarithm problem $[\alpha]P = Q$ in $O(m \log m)$ steps, where m is define as follows.*

1. *Let $m := \lfloor \sqrt{n} \rfloor + 1$.*
2. *Create the two lists:*

$$\begin{aligned} &\textit{baby-steps:} && P, [2]P, \ldots, [m]P \\ &\textit{giant-steps:} && Q \oplus [-m]P, Q \oplus [-2m]P, \ldots, Q \oplus [-m^2]P \end{aligned}$$

Then there exist $0 \leq i, j < m$ such that $Q \oplus [-jm]P = [i]P$ and $\alpha = jm + i$ is the solution to the discrete logarithm problem.

Proof. To compute the two lists we take at most $2m$ group operations. By Lemma 1, there exists a match between the two lists, that can be found in $\log \sqrt{n}$ steps by using standard searching algorithms or hash tables. Hence, the total running time for the algorithm is $O(m \log m)$ steps. □

3. Implicit Baby-Step Giant-Step Algorithm

In this section we describe an implicit version of the BSGS algorithm [1, 2, 3]. The main idea is to define a group action of $\mathbb{Z}_p^*$ on the additive group G. In this way we can use the multiplicative group $\mathbb{Z}_p^*$ as *auxiliary group*, thus reducing the discrete logarithm problem of $(G, \oplus)$ to a problem in $\mathbb{Z}_p^*$. The advantage of this approach is that $\mathbb{Z}_p^*$ has many subgroup and one can exploit its rich and well understood subgroup structure.

Assume $(G, \oplus)$ to be a finite, additive, cyclic group of prime order p and let P be a generator. We define the following (faithful) group action:

$$\begin{aligned} \rho : \mathbb{Z}_p^* &\longrightarrow \mathrm{Aut}(G) \\ \alpha &\longmapsto \ \rho_\alpha : G \longrightarrow G \\ &\qquad\quad P \longmapsto [\alpha]P \end{aligned}$$

It is easy to see that ρ is a group homomorphism with kernel

$$\ker \rho = \{\alpha \in \mathbb{Z}_p^* : \rho_\alpha(P) = P\} = \{1\}$$

If $\varphi \in \mathrm{Aut}(G)$ and P is a generator for G, then $\varphi(P) = [\alpha]P$ for some $\alpha \in \mathbb{Z}_p^*$. Thus we have an isomorphism $\mathbb{Z}_p^* \simeq \mathrm{Aut}(G)$ and we can identify the element $[\alpha]P \in G$ with $\alpha \in \mathbb{Z}_p^*$.

We want to solve the discrete logarithm problem in G by using the auxiliary group $\mathbb{Z}_p^*$. Let z be a primitive element of $\mathbb{Z}_p^*$, then $\alpha = z^k$ for some $0 \leq k < p$ and $Q = [z^k]P$. Let $m := \lfloor \sqrt{p} \rfloor + 1$. If we divide k by m we get i, j with $0 \leq i, j \leq m-1$ such that $k = i + mj$, as in Lemma 1. It follows that $Q = [z^k]P = [z^{i+jm}]P = [z^i][z^{jm}]P$, which leads to

$$[z^{-jm}]Q = [z^i]P$$

However, we know that $Q = [\alpha]P$, thus we have $[z^{-jm}][\alpha]P = [z^i]P$ and this implies $z^{-jm}\alpha = z^i \mod p$. Hence, if we find such an i and j, we can compute $\alpha = z^{i+jm}$ and we have the solution of the discrete logarithm problem. We can now proceed as in Proposition 1.

We put $m := \lfloor \sqrt{p} \rfloor + 1$ and we build the two following lists:

$$\begin{aligned} \text{baby-steps:} \quad & [z]P, [z^2]P, \ldots, [z^m]P \\ \text{giant-steps:} \quad & [z^{-m}]Q, [z^{-2m}]Q, \ldots, [z^{-m^2}]Q \end{aligned}$$

Using binary search, we find a match that solves the DLP in time $O(m \log m)$. The algorithm described above is what we call *implicit baby-step giant-step* (iBSGS).

If a divisor d of $p-1$ is known, this idea can be improved. Let $z_d = z^{\frac{p-1}{d}}$ be a generator for the order d subgroup of $\mathbb{Z}_p^*$. We put $m' := \lfloor \sqrt{d} \rfloor + 1$ and run the implicit baby-step giant-step by using z_d instead of z, that is, using the following lists:

$$\begin{aligned} \text{baby-steps:} \quad & [z_d]P, [z_d^2]P, \ldots, [z_d^{m'}]P \\ \text{giant-steps:} \quad & [z_d^{-m'}]Q, [z_d^{-2m'}]Q, \ldots, [z_d^{-m'^2}]Q \end{aligned}$$

If the unknown α lies in the d-order subgroup of $\mathbb{Z}_p^*$, then α will be equal to z_d^k for some k modulus d and the algorithm will find a match $[z_d^{-jm'}]Q = [z_d^i]P$. Hence, $[z_d^{-jm'}][\alpha]P = [z_d^i]P$, which lead to $\alpha = z_d^{i+jm'}$ and the DLP is solved in times $O(m' \log m')$.

In this case we either find α or verify that α is not in the order d subgroup after at most d iterations. Thus, if d is sufficiently small and α lies in the d-order subgroup, the DLP can be solved much easily.

To summarize:

Theorem 1. *Let G be an additive, cyclic group of prime order p, with generator P. Let $Q = [\alpha]P$ be another given element of G (with α unknown). For a given divisor d of $p-1$, let D be the subgroup of $\mathbb{Z}_p^*$ of order d. Then, one can decide whether α belongs to D in $O(\sqrt{d})$ steps. Moreover, if α belongs to D, the same algorithm will find the discrete logarithm α in $O(\sqrt{d})$ steps.*

In cryptographic applications using elliptic curve cryptosystems, the integer α represent the $private\ key$, while the point $Q = [\alpha]P$ is the $public\ key$. Therefore, every public key for which the corresponding private key lies in a small subgroup of $\mathbb{Z}_p^*$ is deemed to be weak.

4. Weak Key Testing in ECC

This section tackles the vulnerability of weak keys in standardized elliptic curves. We begin by demonstrating how the iBSGS algorithm can efficiently test whether a key within a given curve belongs to a small-order subgroup of the auxiliary group $\mathbb{Z}_p^*$. Subsequently, we conduct a comprehensive analysis of standardized curves, enumerating weak keys residing in subgroups with orders below a specified threshold for each individual curve. Finally, we present a concise overview of our implementation of these algorithms in Rust and PARI/GP.

4.1. Testing whether a key is weak

A simple approach to test whether the private key corresponding to a public key is weak is to set a bound B for the order of the subgroups of $\mathbb{Z}_p^*$. We can run the iBSGS algorithm on all divisors of $p-1$ that are less than B. However, this would be inefficient and redundant, because testing whether a key is in a subgroup of order d also covers all subgroups of order divisible by d. Thus, we instead generate a list of integers $d_1 < d_2 < \cdots < d_t \leq B$ dividing $p-1$ such that $d_i \nmid d_j$ for all $1 \leq i < j \leq t$.

As an example, let us consider the elliptic curve secp192r1 (P192) [10]. The curve is defined over the finite field $\mathbb{F}_q$ where q is a prime specified by the standard. The generator in affine coordinates is

$$P = (602046282375688656758213480587526111916698976636884684818,\\ 174050332293622031404857552280219410364023488927386650641)$$

and its order is the prime

$$p = 6277101735386680763835789423176059013767194773182842284081.$$

By using a computer algebra system (e.g., PARI/GP or SageMath [11, 12]), one can factor $p-1$ in few seconds, and compute the primitive element $z = 3$ in the auxiliary group $\mathbb{Z}_p^*$. Let us choose a bound $B = 2^7$. There are 9 non trivial divisors of $p-1$ below B, namely 2, 4, 5, 8, 10, 16, 20, 40, 80. In order to test whether a given private key is in any of the subgroups of these orders, it suffices to test only the subgroups of order $d = 80$ as the first eight subgroup orders divide 80, and thus any element of one of these smaller orders is also an element of the subgroup of order 80.

Let us choose a public key as the point

$$Q = (816153167907635701966328593112488159610772858343321595848,\\ 706528454659135103431962832049267760754872971757068059209)$$

If we run the iBSGS algorithm, we find a match in a couple of seconds. This gives us the discrete logarithm in base P of Q:

$$\alpha = 390246404348351761435775268611806867566379760936565703767 0$$

which is the corresponding private key.

4.2. Analysis of Weak Keys

In this subsection we investigate elliptic curves described in [10, 13] and other curves used by OpenSSL [14]. For each curve, we enumerate the weak keys appearing in subgroups of order bounded by $B \in \{2^{32}, 2^{64}, 2^{128}, 2^{160}\}$. As described in Section 3, the cost to determine whether a given key is weak with respect to the bound B is roughly $2^{16}, 2^{32}, 2^{64}, 2^{80}$ groups operations. Due to the magnitude order of the results, and to facilitate an easier comparison, we compute the base-2 logarithm of each number (i.e. the number of bits of its representation).

For each curve we describe:

$b(p)$ number of bits of the prime number p which is the order of the generator of group $E(\mathbb{F}_q)$;

n_B base-2 logarithm of the number of weak keys with order bounded by B. Since $\phi(d)$ is the number of generators of a cyclic group of order d, i.e., the number of elements of order exactly d, we define

$$n_B = \log_2 \sum_{\substack{d \mid p-1 \\ d \leq B}} \phi(d)$$

c_B base-2 logarithm of the worst-case number of elliptic curve scalar multiplications required to test whether a key comes from a subgroup of order bounded by B using iBSGS algorithm. Let $R(p, B) = \{d_1, \ldots, d_t : d_i \mid p-1,\ d_i \leq B,\ d_i \nmid d_j \text{ for all } 1 \leq i < j \leq t\}$. We define

$$c_B = \log_2 \sum_{d \in R(p,B)} 2\lceil \sqrt{d} \rceil$$

Curve	$b(p)$	$n_{2^{32}}$	$c_{2^{32}}$	$n_{2^{64}}$	$c_{2^{64}}$	$n_{2^{128}}$	$c_{2^{128}}$	$n_{2^{160}}$	$c_{2^{160}}$
brainpoolP160r1	160	33.6	19.4	67.0	36.5	130.0	67.3	160.0	81.0
brainpoolP192r1	192	11.7	6.8	11.7	6.8	108.1	55.1	108.1	55.1
brainpoolP224r1	224	10.0	6.0	10.0	6.0	10.0	6.0	10.0	6.0
brainpoolP256r1	256	4.2	3.3	4.2	3.3	4.2	3.3	4.2	3.3
brainpoolP320r1	320	34.7	20.3	42.2	22.2	42.2	22.2	42.2	22.2
brainpoolP384r1	384	33.3	18.7	66.0	35.5	130.0	67.5	160.7	82.3
brainpoolP512r1	512	35.0	20.6	68.1	37.7	132.7	70.3	163.3	85.0

Table 1
Weak keys analysis Brainpool curves ($\log_2$ scale).

4.2.1. Analysis of Weak Keys on secp192r1

Here we describe the weak keys analysis of the curve secp192r1, as in Section 4.1. We use the computer algebra system PARI/GP [11] for computations. Similarly, we can perform the weak keys analysis for other NIST's curves using the values of p presented in [10, 15].

The elliptic curve group of secp192r1 is cyclic of prime order

$$p = 6277101735386680763835789423176059013767194773182842284081$$

and the divisors of $p-1$ below the bound $B = 2^{32}$ are

$$\{2, 4, 5, 8, 10, 16, 20, 40, 80, 2389, 4778, 9556, 11945, 19112, 23890, 38224, 47780, 95560, 191120\}$$

thus by removing redundant divisors we get

$$R_B(p) = \{191120\}$$

Doing the calculations we find that $n_B = 17.54$ and $c_B = 9.8$. This shows that secp192r1 has approximately $2^{17.54}$ weak keys lying in subgroups of order below the bound $B = 2^{32}$ and they can be detected in roughly $2^{9.8}$ group operations in the elliptic curve group. We can repeat the test with a bound $B = 2^{160}$ and this leads to $2^{109.0}$ weak keys, computable in roughly $2^{55.6}$ group operations.

4.2.2. Analysis of Standard Curves

Our analysis of weak keys within commonly used elliptic curves, specifically focusing on recommended and standardized ones, aims to identify them and to specify the worst-case scenario for their detection, expressed as the required number of scalar multiplications.

The results are summarized in tables, according to curve characteristics and standards: curves from the Brainpool standard (Table 1), curves defined over prime fields (Table 2), curves over binary fields (Table 3), and curves specifically used in Wireless Transport Layer Security (Table 4), a security protocol employed in the WPA architecture to ensure privacy, data integrity, and authentication during communication between wireless devices.

The data show that many curves have an abundance of weak keys at all levels, due to rather smooth factorization of $p-1$ and, in particular, many divisors of $p-1$ below the given bound B. The actual counts of weak keys vary among the curves, but several of them has around 2^{160} weak keys within the bound $B = 2^{160}$.

Tables 1 to 4 show notable examples of curves that have remarkably few weak keys, especially Brainpool256r1, Brainpool224r1, secp224k1 and ECCp-359. Curves such as secp193r2, Curve25519, c2pnb163v3 and ECCp-353 have few weak keys at lower bounds, but many at $B \geq 2^{128}$. Therefore, the difficulty of identifying weak keys in these curves varies depending on the attacker's computational resources. While computationally constrained attackers may find this task arduous, adversaries with sufficient computational power encounter a significantly reduced barrier, due to the fact that the density of weak keys does not scale linearly.

Curve	$b(p)$	$n_{2^{32}}$	$c_{2^{32}}$	$n_{2^{64}}$	$c_{2^{64}}$	$n_{2^{128}}$	$c_{2^{128}}$	$n_{2^{160}}$	$c_{2^{160}}$
secp112r2	109	3.7	19.5	66.3	36.0	109.8	55.9	109.8	55.9
secp128r1	128	31.8	18.0	66.0	35.4	128.0	65.0	128.0	65.0
secp128r2	126	31.5	16.8	31.5	16.8	126.0	64.0	126.0	64.0
secp160k1	160	18.0	10.0	18.0	10.0	94.8	48.4	160.5	81.8
secp160r2	160	21.6	11.9	21.6	11.8	93.1	47.8	160.0	80.6
P-192	192	17.5	9.8	17.5	9.8	109.0	55.6	109.0	55.6
secp224k1	224	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6
P-224	224	29.0	15.5	29.0	15.5	29.0	15.5	29.0	15.5
Curve25519	253	7.04	4.8	7.04	4.8	114.3	58.2	144.7	73.4
secp256k1	256	24.1	13.1	64.7	34.2	129.4	67.0	147.9	75.0
secp256r1	256	36.0	21.5	69.3	38.8	133.2	70.8	165.3	86.9
SM2	256	32.5	18.13	59.7	30.8	59.7	30.8	59.7	30.8
P-384	384	13.5	7.8	13.5	7.8	103.3	52.7	103.3	52.7
E448	446	33.1	18.9	64.7	34.2	128.8	66.2	160.8	82.2
P-521	521	31.4	16.7	50.0	26.0	128.8	66.3	130.5	66.2

Table 2
Weak keys analysis for curves over prime fields ($\log_2$ scale).

Curve	$b(p)$	$n_{2^{32}}$	$c_{2^{32}}$	$n_{2^{64}}$	$c_{2^{64}}$	$n_{2^{128}}$	$c_{2^{128}}$	$n_{2^{160}}$	$c_{2^{160}}$
c2pnb163v1	162	37.8	23.6	70.9	40.7	134.1	71.9	160.8	82.8
c2pnb163v2	162	26.4	14.2	26.4	14.2	26.4	14.2	160.3	82.2
c2pnb163v3	162	8.8	5.4	8.8	5.4	8.8	5.4	160.9	82.3
c2tnb191v1	190	10.3	6.2	50.7	26.3	124.7	63.4	149.6	75.8
c2tnb191v2	190	20.1	11.0	20.1	11.0	118.68	60.3	118.6	60.3
c2tnb191v3	188	31.8	17.0	37.6	19.8	117.0	59.5	156.6	78.3
c2pnb208w1	192	31.4	16.7	31.4	16.7	31.4	16.7	31.4	16.7
sect193r2	193	2.0	2.0	2.0	2.0	110.2	56.1	110.2	56.1
c2tnb239v1	238	30.2	16.4	55.6	28.8	55.6	28.8	55.6	28.8
c2tnb239v2	237	14.6	8.3	14.6	8.3	14.6	8.3	14.6	8.3
c2tnb239v2	235	30.6	16.4	64.0	33.5	84.4	43.2	158.4	80.2
c2tnb271v2	256	13.0	7.5	66.3	36.0	123.0	62.5	146.0	74.0
c2pnb304w1	288	30.0	16.0	66.3	36.2	132.1	70.0	164.1	85.8
ECCp-353	353	6.3	4.3	6.3	4.3	108.9	55.5	158.3	80.2
c2pnb368w1	352	33.4	19.3	36.9	19.4	128.1	66.0	129.1	65.6
c2tnb431r1	417	31.4	16.7	31.4	16.7	118.6	60.3	118.6	60.3
ECCp-359	359	5.2	3.6	5.2	3.6	5.2	3.6	5.2	3.6

Table 3
Weak keys analysis for curves over binary fields ($\log_2$ scale).

In most cases, however, our analysis show that the probability that a randomly selected key is weak (simply given by the number of weak keys divided by the total number of keys) is very low. Thus, only verifiable randomness in key generation, demonstrably achieved through rigorous audits, can ensure the low probability of picking weak keys susceptible to this attack.

Moreover, a malicious party could cause users to be assigned weak keys, for example via compromised key generation software. To mitigate potential vulnerabilities, it is thus highly advisable that each participant validates the security of their own private key through self-testing, before exposing the public key for any purpose. Conversely, in scenarios demanding high security, a participant should verify the public key of the other party in order to ensure the exchange is robust to attacks up to a bounded computational effort.

It is interesting to notice that we can also use the described method to generate private keys that are secure against this kind of attacks. For this aim we restrict them to be a random power of $z^{\frac{p-1}{r}}$, where z is a primitive root in $\mathbb{Z}_p$ and r is a large prime factor of $p-1$, thus forcing the private key to lie in an r-order subgroup of $\mathbb{Z}_p^*$.

Curve	$b(p)$	$n_{2^{32}}$	$c_{2^{32}}$	$n_{2^{64}}$	$c_{2^{64}}$	$n_{2^{128}}$	$c_{2^{128}}$	$n_{2^{160}}$	$c_{2^{160}}$
wap-wsg-idm-ecid-wtls1	112	32.7	18.6	42.1	22.1	112	57.0	112	57.0
wap-wsg-idm-ecid-wtls3	162	19.2	10.6	59.3	30.6	122.0	62.0	160.2	82.0
wap-wsg-idm-ecid-wtls4	112	33.1	18.6	64.6	34.1	112.0	57.0	112.0	57.0
wap-wsg-idm-ecid-wtls6	112	17.1	9.5	17.1	9.5	112	56.9	112	56.9
wap-wsg-idm-ecid-wtls7	160	31.5	16.8	55.3	28.6	127.6	65.3	159.2	81.1
wap-wsg-idm-ecid-wtls8	112	30.0	16.2	47.7	24.8	112	57.0	112	57.0
wap-wsg-idm-ecid-wtls9	160	33.1	18.9	65.0	34.6	129.0	66.8	159.5	82.0
wap-wsg-idm-ecid-wtls10	231	33.0	18.6	64.0	33.5	73.2	37.6	159.8	81.5
wap-wsg-idm-ecid-wtls11	232	33.0	18.3	64.7	34.3	87.6	44.7	160.6	82.0
wap-wsg-idm-ecid-wtls12	224	29.0	15.5	29.0	15.5	29.0	15.5	29.0	15.5

Table 4
Weak keys analysis for Wireless Transport Layer Security curves ($\log_2$ scale).

4.3. Software Implementation

We implemented the finite fields arithmetic and the elliptic curve arithmetic using the Rust programming language. We employ the Montgomery ladder algorithm in the generation of private and public key pairs, thus fortifying resistance against side-channel attacks.

Moreover, we implemented the iBSGS algorithm presented in [2, 3] to test whether a given public key derives from a weak private key (up to a certain bound).

Using PARI/GP we implemented a script to calculate the number of weak keys up to a given bound and the worst-case complexity of finding them in terms of scalar multiplications.

It is noteworthy that identifying weak keys in a specific curve primarily involves computing the divisors of $p-1$, which translates to factoring. The comprehensive analysis of standardized curves was completed within several days, primarily using a computer with a Intel® Core™ i3-2350M CPU @ 2.30GHz $\times$ 4 cores, running Linux. All these tools are available at [4].

5. Conclusions

In this work we have presented a comprehensive analysis of standardized elliptic curves employed in prevalent applications. Our objective has been to quantify the prevalence of *weak keys*, defined as private keys residing within subgroups of the auxiliary group $\mathbb{Z}_p^*$ whose orders fall below a specified threshold (and hence amenable to recovery attacks). Furthermore, we have established the worst-case complexity associated with verifying whether a given key possesses this vulnerability.

To achieve these results, we leverage the *implicit baby-step giant-step* algorithm [1, 2, 3]. This algorithm hinges on defining a group action that maps elements from $\mathbb{Z}_p^*$ to the elliptic curve group $E(\mathbb{K})$. By doing so, the elliptic curve discrete logarithm problem (DLP) is effectively transformed into a problem within $\mathbb{Z}_p^*$, where it benefits from a well-defined and well-understood subgroup structure. Consequently, the primary advantage of this approach lies in reducing the DLP's complexity for private keys belonging to small-order subgroups of $\mathbb{Z}_p^*$.

Our analysis reveals a potential vulnerability in many standardized curves where a significant number of private keys reside within a small subgroup of the auxiliary group. This arises from the presence of numerous small divisors in the auxiliary group order. While the likelihood of randomly selecting a weak key remains low, malicious actors could exploit this weakness by manipulating the key generation libraries utilized by applications. This manipulation would enable the construction of private keys readily recoverable from their corresponding public keys via the iBSGS algorithm. Consequently, it is imperative for applications to implement rigorous testing of keys generated by these libraries prior to their usage. In fact, we have implemented all the algorithms used in this paper in Rust and PARI/GP. These tools, available at [4], empower users to assess the susceptibility of private keys to these specific attack vectors.

Future Work. To counteract the positive impact of using the iBSGS algorithm, a possible solution would be to standardize elliptic curves whose group order is a *safe prime*, i.e., a prime p such that $p = 2r + 1$ where r is also a prime. More effort should be made in order to evaluate and assess the possible pitfalls of using such a prime.

Moreover, it would be interesting to use the implicit representation to improve other algorithms such as the Pohlig-Hellman [6, 7], akin to what it was done for the Pollard's kangaroo algorithm in [3].

Acknowledgments

This work has been partially supported by the Department Strategic Project on Artificial Intelligence (2020-25) of the University of Udine, and the project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU. We thank Luca Campa for discussions and suggestions about the content of this paper.

References

[1] U. M. Maurer, S. Wolf, The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms, SIAM Journal on Computing (1999). doi:10.1137/S0097539796302749.

[2] P. Kushwaha, A. Mahalanobis, A probabilistic baby-step giant-step algorithm, arXiv preprint arXiv:1701.07172 (2017).

[3] M. J. Jacobson, Jr., P. Kushwaha, Removable weak keys for discrete logarithm-based cryptography, Journal of Cryptographic Engineering (2020). doi:10.1007/s13389-020-00250-7.

[4] E. Talotti, Elliptic curve cryptography weak keys, Available at https://github.com/cysecud/ecc_weak_keys, 2024.

[5] J. Silverman, The arithmetic of elliptic curves, Springer, 2009.

[6] D. R. Stinson, M. B. Paterson, Cryptography theory and practice, Chapman and Hall/CRC, 2018.

[7] J. Hoffstein, J. Pipher, J. Silverman, An Introduction to Mathematical Cryptography, Springer, 2008.

[8] D. Shanks, Class number, a theory of factorization, and genera, in: Proc. Symp. Math. Soc., volume 20, 1971, pp. 415–440.

[9] H. Cohen, A Course in Computational Algebraic Number Theory, Springer, 1993.

[10] L. Chen, D. Moody, A. Regenscheid, K. Randall, Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters, Technical Report, National Institute of Standards and Technology, 2019.

[11] parigp, PARI/GP, Available at https://pari.math.u-bordeaux.fr/, 2023.

[12] sagemath, SageMath, Available at https://www.sagemath.org/, 2023.

[13] J. Merkle, M. Lochter, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, RFC 5639, 2010. URL: https://www.rfc-editor.org/info/rfc5639. doi:10.17487/RFC5639.

[14] The OpenSSL Project, OpenSSL: The open source toolkit for SSL/TLS, 2023. Available at www.openssl.org.

[15] crocs, Center for research on cryptography and security, Available at https://neuromancer.sk/std/search/, 2023.