CRYPTREC Review of EdDSA

Steven D. Galbraith

Mathematics Department, University of Auckland, Auckland, New Zealand. s.galbraith@auckland.ac.nz

1 Executive summary

The EdDSA signature scheme is a digital signature based on the Elliptic Curve Discrete Logarithm Problem (ECDLP). It was proposed by Bernstein, Duif, Lange, Schwabe and Yang in 2012 [18]. It builds on a long line of discrete logarithm based signature schemes, including Elgamal, Schnorr, DSA, and ECDSA. The security of signature schemes of this type is well-understood, and elliptic curve cryptography is a mature field.

My conclusions and opinions:

1. EdDSA is a good design for a signature scheme (except perhaps for the key clamping, see Section 6.1, which seems to cause more difficulties than it provides benefits).
2. EdDSA is more closely related to Schnorr signatures than ECDSA, and so enjoys many of the rigorous security guarantees that are known for Schnorr signatures, including recent work on tight security proofs.
3. Deterministic signatures solve some of the security problems of discrete log signatures, but constant time implementation is still critical in many settings.
4. EdDSA is superior to ECDSA when doing batch verification of a large number of signatures.
5. Curve 25519 provides a high level of security for the next 10-20 years, and 448-bit keys (such as in Ed448) are over-conservative and not recommended.
6. It is unlikely that quantum computers capable of solving 256-bit ECDLP instances can be built within the next 10 years.
7. I am confident that EdDSA using Curve25519 is a good signature scheme for use up to 2030.

2 Introduction and outline

The original paper [18] on EdDSA describes a very specific scheme, with respect to a fixed elliptic curve. But of course the basic concept can be generalised to work with other curves. The specific field in [18] is for the prime $p = 2^{255} - 19$. The specific curve in [18] is the twisted Edwards curve

$$-x^2 + y^2 = 1 + (-121665/121666)x^2y^2$$

over $\mathbb{F}_p$. Then $\ell = 2^{252} + 27742317777372353535851937790883648493$ is prime, and $\#E(\mathbb{F}_p) = 8\ell$.

We give a high level overview of the EdDSA signature scheme in the case of an arbitrary elliptic curve E over a finite field $\mathbb{F}_q$. Let $B \in E(\mathbb{F}_q)$ be a point of prime order ℓ. User Alice has private key $0 \leq a < \ell$ (in fact, the private keys in EdDSA are not sampled uniformly in $\mathbb{Z}_\ell$, due to a process called "key clamping" that we explain later) and public key $A = [a]B$. Elliptic curve points have a compressed representation that we denote $\underline{A}$. Alice also has a fixed secret key h. Let H be a cryptographic hash function and interpret its values as integers. To sign a message $\mathsf{m} \in \{0,1\}^*$ Alice computes $r = H(h, \mathsf{m}) \pmod{\ell}$, $R = [r]B$, and $S = (r + H(\underline{R}, \underline{A}, \mathsf{m})a) \pmod{\ell}$. The value r is sometimes called an "ephemeral value" or a "nonce", and the point R is sometimes called a "commitment". Here the notation $H(h, \mathsf{m}) = H(h \| \mathsf{m})$ means that the binary strings h and m are concatenated before signing. The signature is $(\underline{R}, S)$, where $\underline{R}$ is a bitstring that is the compressed representation of the point and where S is an integer in $\mathbb{Z}_\ell$. There are two ways to verify signatures (see Section 6.3). One was is to compute the hash value $H(\underline{R}, \underline{A}, \mathsf{m})$ and then compute

$$R' = [S]B - [H(\underline{R}, \underline{A}, \mathsf{m})]A$$

and compress R' and check that this equals $\underline{R}$.

A key feature is that the ephemeral value r is computed in a deterministic way (but varying, depending on the message), as the hash value $H(h, \mathsf{m})$. This is to avoid relying on perfect random bits in signing, which have often been a problem in applications (an extreme example is the Sony Playstation 3 hack, where a failure to choose random ephemeral values allowed attackers to determine the private key for signing all applications). One consequence is that, for a given public key, there is a unique signature for each message.

For comparison, we give a high level presentation of ECDSA, which is essentially same as Elgamal signatures: Alice has private key $0 \leq a < \ell$ and public key $A = [a]B$. To sign a message $\mathsf{m} \in \{0,1\}^*$ Alice chooses a uniform random integer k such that $0 < k < \ell$, computes $R = [k]B$ and converts R to an integer r (typically this is done by taking the x-coordinate of the point on a Weierstrass equation and converting to an integer). Set $r = r \pmod{\ell}$ and if $r = 0$ then repeat with fresh k. Compute $k^{-1} \pmod{\ell}$. Compute $s = k^{-1}(H(\mathsf{m}) + ar) \pmod{\ell}$. If $s = 0$ then repeat with fresh k. The signature is (r, s), being a pair of elements of $\mathbb{Z}_\ell$. To verify one computes $w = s^{-1} \pmod{\ell}$, $u_1 = H(\mathsf{m})w \pmod{\ell}$ and $u_2 = rw \pmod{\ell}$ and

$$X = [u_1]B + [u_2]A.$$

If X is the identity then reject the signature, else convert X to an integer r' and accept the signature if $r' \equiv r \pmod{\ell}$.

One annoying aspect of ECDSA is that there are a number of special cases and checks in the signing algorithm. The scheme also uses an un-natural conversion of a point R to an integer r. Finally, it seems to have always been more difficult to get security proofs for ECDSA than for Schnorr signatures. None of these issues has prevented ECDSA from being very widely used, but it is preferable to have signature schemes such as Schnorr and EdDSA that are simpler to implement and cleaner to analyse.

EdDSA and ECDSA are both of the "Fiat-Shamir" type, where one starts with an interactive identification protocol (a Σ-protocol) and replaces the challenge with a hash value. The security of such signature schemes is well-understood. In the random oracle model one can prove security using the forking lemma, and we sketch the details in Section 7. There are even more recent proof techniques that give tighter results. We will review some old attacks on discrete logarithm based signature schemes and discuss some other possible attacks or issues in Sections 7 and 8. Our conclusion is that EdDSA is a good design that provides a high level of security.

We highlight some particular features of EdDSA signatures:

1. Use of the Edwards model for the ellptic curve. This model has a complete group law and so there are no branch conditions in the group operations, and in particular no branches that depend on secret data.
2. Deterministic signatures (no random sampling needed to generate the ephemeral value).

The stability and maturity of elliptic curve cryptography (ECC) is beyond doubt. Very little has changed in the 20 years since Johnson, Menezes and Vanstone wrote their survey [51] on ECDSA and since Lenstra and Verheul wrote their influential paper on selecting cryptographic key sizes [58]. Their predictions about the Elliptic Curve Discrete Logarithm Problem (ECDLP) have been confirmed.

For elliptic curves over prime fields, there has not been any discovery of "weak" elliptic curves since the 1990s. For elliptic curves over prime fields the known sets of weak curves are: supersingular curves (due to the MOV/Frey-Rück attack) and anomalous curves. There is also the small speedup for Pollard rho that arises when considering curves with more automorphisms than just $[1]$ and $[-1]$. All these results were known in the 1990s. So I claim that in the last 20 years there have been no results on elliptic curves to question their security. Of course, at any time there could be a new idea or mathematical breakthrough that lowers the security of elliptic curves, but I consider this unlikely for several reasons which I discuss in Section 4.6.

Progress on the Certicom challenges from 1997 has been surprisingly slow. More than 10 years ago Bos et al [25] wrote "with the current state of the art in cryptanalysis we would be surprised if a public effort can make a dent in 160-bit prime field ECC by the year 2020", and this prediction seems to have been more than true. Indeed, the largest ECDLP instances solved over a prime field remain the 109-bit challenge, solved in 2004, a 111-bit case solved in 2009 by Bos et al [26], and a 114-bit ECDLP in an interval solved using Pollard kangaroo by Pons and Zieniewicz [69].

Back in November 2009 a large team (see Bailey et al [5]) initiated a major computation to solve the ECDLP on an elliptic curve over a 130-bit Koblitz curve (meaning the elliptic curve E is defined over $\mathbb{F}_2$ and the group under consideration is $E(\mathbb{F}_{2^{131}})$. Breaking this example would be expected to be easier than breaking ECDLP over a 130-bit prime field. The computation seems to have never finished, and has been abandoned.

For signature schemes the first security requirement is to avoid a **total break**, which is where an attacker tries to compute the private key from the information given in the public key. For EdDSA this is solving an instance of the ECDLP. As we will explain in Section 5, the default elliptic curve for EdDSA is Curve25519, which avoids all known "weak curve" families and has a high security level (at least 128-bits of security). This is a good choice of elliptic curve and we have no concerns about its security.

Potentially, the biggest threat to EdDSA is from quantum computers, and I discuss this in Section 9. It is important to distinguish the use of ECC for signatures and authentication from its use in encryption. For encryption we may need data to be still protected in 20 or 30 years, which means that if large-scale quantum computers are considered a risk within that time, then one should migrate to a quantum-resistant forms of encryption. However, for many applications of authentication (but not all), it is acceptable to continue using elliptic curve public key signatures until such time as there is a realistic threat that the ECDLP can be solved using a quantum computer. An exception to this remark is systems that have a long lifespan in the field and are hard to update, and where digital signatures play a fundamental role in their security (for example, firmware updates in an embedded micro-controller, where the master public key and signature verification algorithm are hardwired in the system, and where the device is intended to have a lifespan of 10 or more years).

As will be discussed in Section 9, it seems that quantum computers that can run Shor's algorithm on real-world ECDLP instances are still far from being built. Even with major advances, it seems unlikely all the technical obstructions to large scale quantum computing will be solved within the next 10 years. In short, for most applications of digital signatures, we can still confidently use elliptic curves for the coming decade or longer.

Plan of report: In this report I discuss all known attacks against EdDSA. I first discuss general attacks on ECDLP, and survey the progress on ECDLP in the last 20 years. This leads to a security evaluation of the particular curves used in EdDSA, especially Curve25519. I also give a security analysis of the construction as a signature scheme, including discussion of the security requirements of the hash function. I review security in the multi-user setting. I also briefly discuss side channel attacks, fault attacks, and previous attacks on signature schemes. Finally I mention the progress on quantum computing, and the future of using signatures based on ECDLP in the time of post-quantum cryptography.

3 Elliptic curves and Edwards curves

Basic references for elliptic curves in cryptography are [4, 21, 47, 43, 81]. In this report we focus on elliptic curves over finite fields $\mathbb{F}_p$ of characteristic not equal to 2. Several facts that we state are not true in characteristic 2.

An elliptic curve is typically given in **Weierstrass model** as

$$E : y^2 + A_1xy + A_3y = x^3 + A_2x^2 + A_4x + A_6$$

such that the discriminant is non-zero. If $P = (x, y)$ is a point on such a curve then $-P = (x, -y - A_1x - A_3)$. The point at infinity on the projective completion of the curve is denoted ∞, and can be expressed in projective geometry by using homogeneous coordinates. Other curve models such as Montgomery and twisted Edwards are useful for efficient implementation. From the point of view of the ECDLP, all curve equations are equivalent as one can transform an instance of the ECDLP from one curve equation to another efficiently.

A simple observation is that if $P = (x, y)$ is a point on E then $-P$ is the only other point on E with this x-coordinate. This leads to the idea of point compression: Given a point $P = (x, y)$ one can represent P by the pair (x, b) where b is a single bit that indicates the choice of y (for example, when p is an odd prime and $0 \leq x < p$ is an integer then b could be the value $x \pmod 2$ since x and $p - x$ have the opposite parity; alternatively b could be the sign bit if we represent $-p/2 < x < p/2$).

There is a group operation on elliptic curves. For Weierstrass models the point at infinity is usually taken to be the identity element for the group. Hence, if E is an elliptic curve over a finite field $\mathbb{F}_p$, then the set of rational points $E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 : y^2 + A_1xy + A_3y = x^3 + A_2x^2 + A_4x + A_6\} \cup \{\infty\}$ is a finite Abelian group. We write the group operation additively and for $P \in E$ and $n \in \mathbb{Z}$ define $[n]P$ as $[0]P = \infty$, $[1]P = P$, $[n]P = [-n](-P)$ and, for $n \in \mathbb{N}$, $[n]P = P + P + \cdots + P$ (n times).

Let K be a field such that $char(K) \neq 2$. Let $a, d \in K$ satisfy $a \neq 0, d \neq 0, a \neq d$. The **twisted Edwards model** is

$$ax^2 + y^2 = 1 + dx^2y^2.$$

The identity element is $(0,1)$ and the inverse of (x,y) is $(-x,y)$. (It would have been preferable to switch the roles of x and y so that the inverse of (x,y) was $(x,-y)$ as it is for Weierstrass curves, but this convention has not been adopted.) The point $(0,-1)$ has order 2 and the points $(\pm\sqrt{1/a},0)$ have order 4.

The EdDSA paper restricts to the case $a=-1$ and d not a square.

The group operation on twisted Edwards models is

$$(x_1,y_1)+(x_2,y_2)=\left(\frac{x_1y_2+x_2y_1}{1+dx_1x_2y_1y_2},\frac{y_1y_2-ax_1x_2}{1-dx_1x_2y_1y_2}\right). \tag{1}$$

This is not how the group operation is computed in cryptographic implementations. Instead there is a large choice of different coordinate systems such as projective twisted coordinates, inverted twisted coordinates, etc. Since the choice of coordinate system has no impact on security we do not explore such topics in this review.

The Edwards group law is **complete** for points defined over K in certain cases. This is important as it means computing the group law does not require conditional branches, which makes it easy to ensure that execution traces are independent of the private key.

Edwards curves are named for Harold M. Edwards, and were introduced in the paper "A normal form for elliptic curves" [37] (where he proposed the form $x^2+y^2=a^2(1+x^2y^2)$). They were first studied for crypto in "Faster addition and doubling on elliptic curves" by Bernstein and Lange [14] and "Twisted Edwards Curves" by Bernstein at al [11].

Every Edwards curve has a point of order 4. Conversely, every elliptic curve (over a field of characteristic not equal to 2) with a point of order 4 is birational to an Edwards curve (see Theorem 2.1 of [14] or Theorem 3.3 of [11]). Twisted Edwards models are more general: Theorem 5.1 of [11] states that every elliptic curve over a field K having three K-rational points of order 2 is 2-isogenous over K to a twisted Edwards curve. In summary, while not every curve can be put into Edwards or twisted Edwards form, these curves cover a large proportion of all elliptic curves. So there is no evidence that Edwards or twisted Edwards curves are less secure than random elliptic curves.

4 ECDLP and algorithms

Let E be an elliptic curve over a finite field $\mathbb{F}_q$, where $q=p^n$ and p is prime. The **elliptic curve discrete logarithm problem (ECDLP)** is the following computational problem: Given points $P,Q\in E(\mathbb{F}_q)$ to find an integer a, if it exists, such that $Q=[a]P$.

Two special cases of easy instances of the ECDLP are the following.

1. An elliptic curve E over a prime field $\mathbb{F}_p$ is **anomalous** if $\#E(\mathbb{F}_p)=p$. It is known that one can solve the ECDLP in polynomial time on an anomalous curve. This attack was discovered by Semaev [74], and subsequently by Rück [71], Satoh and Araki [72] and Smart [75].
2. Let E be an elliptic curve over a field $\mathbb{F}_q=\mathbb{F}_{p^n}$ where p is prime, and suppose $\ell \mid \#E(\mathbb{F}_q)$ is a prime with $\ell\neq p$. Then the ℓ-th roots of unity in $\overline{\mathbb{F}}_p$ lie in $\mathbb{F}_{p^k}$ where k is the smallest integer $k\geq 1$ such that $\ell \mid (p^k-1)$. The Weil (or Tate) pairings can be used to transform the ECDLP from $E(\mathbb{F}_p)$ to $\mathbb{F}_{p^k}^*$. When k is small typically $k\leq 6$) then subexponential-time DLP algorithms in finite fields can be used to solve the DLP faster than on the original elliptic curve. This is the Menezes, Okamoto and Vanstone [60] (MOV) and Frey and Rück [41] attack. The integer k (or, sometimes k/n) is called the **embedding degree**. The ECDSA standard [1] requires the embedding degree to be at least 100, while the Brainpool standard [59] requires $k>(\ell-1)/100$, which seems to be over-conservative.

In practice it is easy to avoid these two cases.

We now discuss the main algorithms to solve the ECDLP, which are algorithms that apply for any group. Some of these algorithms (such as baby-step-giant-step) are deterministic and some (such as Pollard rho) should be treated as randomised algorithms. For randomised algorithms we usually consider the expected running time, and this can be studied in two flavours: (1) For fixed instances, with the probability over the random choices of the algorithm (worst case expected running time); (2) the expectation of the running time over both the choice of instance and the randomness in the algorithm (**average case expected running time**).

Before continuing, we should make a comment about assessing the complexity of algorithms. It is traditional to focus on running time, but one should not neglect the cost of storage when evaluating algorithms with large storage

requirements. This topic was explored by Wiener [82], where he explained that one should take storage costs and memory access into account when analysing algorithms. In the context of baby-step-giant-step he explained that the complexity exceeds square root, in contrast to Pollard rho using distinguished points, which can be implemented with any fixed storage bound. Similarly, Section 2.3 of Bernstein and Lange [17] discusses the **AT metric** (meaning the product of the **area** of a chip and the **running time**). They explain that in this model no algorithm is known that beats Pollard rho, even taking into account precomputation or non-uniform algorithms (see Section 4.5).

We now make an additional comment about success probability. In computational number theory we usually consider algorithms that succeed in all cases (or succeed with probability extremely close to 1). However in cryptography it is common to consider algorithms that run in time t and succeed with probability ϵ. For ECDLP (and many other problems) an algorithm that succeeds with probability $\epsilon < 1$ can be transformed into an algorithm that succeeds with probability close to 1. For example, for ECDLP we can transform any instance (P, Q) into a random instance $([u]P, [v]Q)$ and run our algorithm on this instance, which is an independent execution of the algorithm. Hence, the existence of an algorithm that runs in time t and succeeds with probability ϵ implies the existence of an algorithm that runs in time close to t/ϵ and succeeds with probability close to 1. However, the converse of this argument doesn't automatically hold. In other words, the existence of an algorithm that solves a problem in time t with probability 1 does not imply the existence of an algorithm that succeeds with probability ϵ and solves the problem in time $t\epsilon$. This problem has not received much study in the literature. We will discuss this in the following subsections, in the context of the ECDLP.

To conclude this preamble, we give a definition of the phrase **security level** for the ECDLP. The standard meaning, since all known algorithms for the ECDLP require at least $\sqrt{\ell}$ group operations, where ℓ is the (prime) order of P, is to say that the ECDLP has λ-bit security if $2^{2\lambda} < \ell < 2^{2\lambda+2}$. More precisely, it is standard to conjecture that if E is a "general" curve and if $\ell > 2^{2\lambda}$ then every ECDLP algorithm that runs in time t under a suitable metric (e.g., t is a count of the number of finite field operations or elliptic curve operations and also the cost of memory accesses) and succeeds with probability ϵ, then $t/\epsilon > 2^{\lambda}$.

4.1 Pohlig-Hellman

Let (P, Q) be an instance of the ECDLP, where P has order N which is not prime. If the factorisation of N is known then it is straightforward to reduce the ECDLP to a collection of ECDLP instances of prime order ℓ, for each $\ell \mid N$. Hence the difficulty of the ECDLP depends on the largest prime divisor of N. For this reason (and also other reasons that arise in security proofs) we usually insist that P is a point of prime order. For the remainder of this section we assume P has prime order ℓ.

An elliptic curve E over $\mathbb{F}_q$ such that the group order $\#E(\mathbb{F}_q)$ is prime is called a **prime order** curve. An elliptic curve E over $\mathbb{F}_q$ such that the group order $\#E(\mathbb{F}_q) = c\ell$ where ℓ is prime and $c \leq 8$ is called a **nearly prime order** curve.

4.2 Baby step giant step (BSGS)

The baby-step-giant-step algorithm is based on the observation that, taking $M = \lceil \sqrt{\ell} \rceil$, one can write $a = a_0 + a_1 M$ where $0 \leq a_0 < M$ and $0 \leq a_1 < M$. The idea is to compute and store all points $[a_0]P$ (these are the "baby steps") and then to sequentially compute $Q - [a_1]([M]P)$ (the "giant steps") and seek a match in the list. The point $[M]P$ is computed once at the start of the algorithm. The algorithm performs $O(\sqrt{\ell})$ group operations and requires $O(\sqrt{\ell})$ group elements of storage. As already mentioned to, in cost metrics that take into account memory costs, the complexity of BSGS is worse than $O(\sqrt{\ell})$.

Pollard (Section 3 of [68]) proposed to "interleave" the computations to improve the average-case performance. This gives an algorithm with average-case running time of $\frac{4}{3}\sqrt{\ell}$ group operations. A complete analysis of the BSGS algorithm for elliptic curves is given in [44].

Since it is appropriate in crypto to consider algorithms that succeed only with probability ϵ, we analyse the interleaved BSGS algorithm in this context. Intuitively, if an algorithm computes t group elements then one would expect there to be at most t^2 combinations of pairs of points. The algorithm solves the ECDLP if Q is equal to one of these combinations of points. So the success probability to solve the ECDLP can be at most t^2/ℓ. In other words, if have

computed $t = \theta\sqrt{\ell}$ points for some $0 < \theta < 1$ then the success probability is at most θ^2. Conversely, to have success probability ϵ we take $\theta = \sqrt{\epsilon}$ and find that

$$t/\epsilon \geq \theta\sqrt{\ell}/\theta^2 = \sqrt{\ell}/\theta = \sqrt{\ell}/\sqrt{\epsilon} \geq \sqrt{\ell}.$$

An alternative way to get an algorithm with success probability $\epsilon = 1/2^n$ is to guess n bits of the solution to the DLP and then solve the rest. Since the size of the problem space is reduced to $\ell/2^n$ we can solve the problem in square root steps, giving a BSGS algorithm that requires $O(\sqrt{\ell/2^n})$ group operations. This gives the same result $t/\epsilon = O(\sqrt{\ell/2^n})2^n = O(\sqrt{\ell 2^n})$. This also shows that dividing up the problem space and distributing the computations over the internet does not give a linear speedup. In conclusion, we have an algorithm that runs in time $O(\sqrt{\ell})$ and succeeds with probability 1, but for small $0 < \epsilon < 1$ we do not have an algroithm that runs in time $\epsilon\sqrt{\ell}$ that succeeds with probability ϵ.

4.3 Pollard rho

The baby-step-giant-step algorithm requires large storage and is hard to parallelise or distribute over the internet. The rho and kangaroo algorithms using distinguished points require less storage and can be distributed. The basic idea is to reduce the discrete logarithm problem to the problem of collision-finding, and then use low-storage collision-detection methods. In the rho algorithm one seeks a collision of the form $[u]P + [v]Q = [u']P + [v']Q$ while in the kangaroo algorithm one seeks a collision of the form $[u]P = Q + [v]P$. Both algorithms exploit pseudorandom walks. We do not present all the details of these algorithms as there are several good references [4, 43], but the main principle is to design stateless pseudorandom walks in the group so that:

1. The next group element in the walk is computed as a deterministic function of the current group element;
2. The cost of each step in the walk is approximately the cost of a single group operation;
3. Collisions can be detected using distinguished points.

The heuristic analysis of the Pollard rho algorithm is based on the birthday paradox. The probability of success for Pollard rho after sampling t group elements is approximately $1 - \exp(-t^2/2\ell)$. Van Oorschot and Wiener [80] showed that the heuristic average case expected running time (in the serial case) is $(\sqrt{\pi/2} + o(1))\sqrt{\ell}$ group operations. It is natural to believe that this running time is "optimal", in the sense that no algorithm based on finding collisions in a set of size ℓ should be able to do better than the birthday paradox. Note that the worst-case running time for Pollard rho is unbounded.

The Pollard rho algorithm with distinguished points is easily distributed over the internet. Clients can run pseudorandom walks and then transmit distinguished points to a central server. The speedup is linear (in the sense that if there are L processors of equal power running in parallel then the elapsed clock time to solve the ECDLP instance is $O(\sqrt{\ell}/L)$.

As with BSGS, for small values $0 < \epsilon < 1$, there isn't a variant of the Pollard rho algorithm that runs in time $\epsilon\sqrt{\pi\ell/2}$ and succeeds with probability ϵ. Hence, for a group of order $\approx 2^{256}$, if one runs Pollard rho for $t = 2^{80}$ group operations, then the expected success probability is approximately $\epsilon = 1 - \exp(-2^{160-257}) \approx 2^{-97}$. It follows that $t/\epsilon \approx 2^{80+97} = 2^{177}$ is much larger than the required 2^{128}. In other words, to the best of our current knowledge, 256-bit group orders provide much more than 128-bits of security against attackers that run for substantially fewer than 2^{128} operations. Indeed, one can show using the above formulae that 206-bit group orders should be sufficient to ensure that if Pollard rho is run for $t = 2^{80}$ group operations, then the expected success probability is approximately 2^{-48}. In other words, 206-bit group orders have 128-bit security if we are only concerned with attackers that succeed with probability at most 2^{-48}.

The Pollard kangaroo method is designed for a variant of the ECDLP where we have $Q = [a]P$ for some integer $0 \leq a < N < \ell$. The kangaroo method is slower than the rho method when Q is uniformly distributed in the subgroup.

4.4 Computational records and predictions

Certicom initiated an ECDLP challenge in November 1997 in order to encourage and stimulate research on the ECDLP. The Pollard rho algorithm is the method used to solve large-scale ECDLP computations. We gather the recent results

in Table 1. The phrase "Koblitz" denotes a curve E over $\mathbb{F}_2$ with group $E(\mathbb{F}_{2^n})$ where n is prime; in this case the Frobenius automorphism $\psi(x, y) = (x^2, y^2)$ acts on the group and so one gets a speedup to the attack by exploiting orbits of size $2n$ from Frobenius and inversion.

Authors	Bit size	Year	Field	Hardware
Monico	108	2002	Large p	CPU
Monico	108	2004	Char 2	CPU
Bos et al	111	2009	Large p	PS3
Wenger, Wolfger	112	2014	Koblitz	FPGA
Wenger, Wolfger	113	2015	Char 2	FPGA
Bernstein et al	117	2016	Char 2	FPGA
Zieniewicz, Pons	114	2020	Large p	GPU

Table 1. Summary of record ECDLP computations.

In November 2009 an attempt was initiated [5] to solve the Certicom challenge ECC2K-130. This computation has never been completed as far as I am aware.

Bos et al [25] wrote in 2009 "with the current state of the art in cryptanalysis we would be surprised if a public effort can make a dent in 160-bit prime field ECC by the year 2020" and this prediction has come true.

In 2001 Lenstra and Verheul [58] made predictions for elliptic curve group order sizes for security into the future, and there is no evidence that their predictions should be adjusted. They suggest that for security to the year 2030 one should use fields of between 176 and 215 bits, and for security to the year 2050 it would suffice to use fields of between 206 and 272 bits. Here the lower bounds (176 and 206 respectively) are in the setting where there is no cryptanalytic progress, in the sense of new or improved algorithms for the ECDLP, which is the situation according to the public literature.

Table K.2 of the ANSI ECDSA standard ANS X9.62 [1] claims that 112-bit security level (elliptic curves over 224-bit fields) are secure to 2030 and curves over 256-bit fields are secure "beyond 2030". So this is consistent with usage of EdDSA for at least another decade.

In conclusion, predictions over the last 20 or more years about progress in solving ECDLP have all turned out true. There have been no major surprises over this period. All evidence suggests that 256-bit field sizes are safe to use until at least 2050, unless there is major progress in quantum computers (see Section 9). I see no reason to consider group orders of larger than 256 bits. In particular, I see no reason to consider Ed448 for any practical application.

4.5 Precomputation / multiple instances

If many users are sharing a fixed elliptic curve group then it is natural to consider whether one can attack all their keys more efficiently than solving each key individually. Ideally, if there are n users one would want the cost to break all their keys to be n times the cost of attacking a single key. A related problem is whether there is a benefit to doing precomputation when a fixed group is used. The meaning of precomputation is that some work is done before seeing a particular public key that the attacker wants to break.

For symmetric cryptography the security depends on the exact model. Suppose a cipher is being used with λ bit keys, and the n users have keys $k_1, \ldots, k_n$. If the attacker has ciphertexts $c_i = \mathsf{Enc}_{k_i}(m)$ for $1 \leq i \leq n$ for a *fixed* message m, then the attacker can try all keys and hence break all n keys in $\tilde{O}(2^\lambda)$ executions of the encryption algorithm. On the other hand, if the attacker has message-ciphertext pairs $(m_i, c_i = \mathsf{Enc}_{k_i}(m_i))$ for $1 \leq i \leq n$, where the messages are distinct, then the cost of the attack is proportional to $n2^\lambda$ encryptions. This is a reason why random nonces or IVs are used in symmetric encryption.

Now suppose an attacker has n instances of the discrete logarithm problem in a fixed elliptic curve group of size ℓ and wants to solve them more quickly than n times the cost of each individual logarithm. Kuhn and Struik [55] studied re-using all previous distinguished point values when solving many instances and showed that if $n < \ell^{1/4}$ (which will always we true in practice) then one can solve all n instances in approximately $\sqrt{2\ell n}$ group operations. The

instances are solved consecutively, but note that, since all distinguished points are stored, the storage cost is greater than performing n computations in serial. As noted by Hitchcock, Montague, Carter and Dawson [48], their work is incomplete as it does not explain how to do a random walk that is independent of the ECDLP instances. Indeed, the Pollard kangaroo method is ideal for the case of many ECDLP instances since the steps in the random walks in the kangaroo method do not depend on the ECDLP instance (only the starting point depends on the instance). Further discussion on this problem was given by Bernstein and Lange [16, 17] and Fouque, Joux and Mavromati [40]. In short, for discrete logarithm systems in a fixed group, there is a security loss (by a factor $\sqrt{n}$) in the n user case compared with the ideal scenario. This is not a serious problem in practice, since security levels are chosen according to the single user case.

One can interpret the Kuhn and Struik result as saying that the additional cost of the $(n+1)$-th ECDLP instance in a group of size ℓ is proportional to $\sqrt{\ell(n+1)} - \sqrt{\ell n} \approx \sqrt{\ell/4n}$. One can view this as a precomputation: The computation of $\sqrt{\ell n}$ points reduces the collision time to $O(\sqrt{\ell/n})$ operations. Much has been written on precomputation approaches, including by Mihalcik [61], and Lee, Cheon and Hong [57].

A related topic, but more theoretical, is non-uniform algorithms for the ECDLP. In non-uniform algorithms there is an "advice string" (that may be hard to compute) which allows to compute instances more efficiently. This problem has been considered by several authors, including Bernstein and Lange [17], and Corrigan-Gibbs and Kogan [36]. The concept is that when using a fixed group (such as Curve25519) then an attacker could do a major precomputation (resulting in an "advice string") that speeds up the solving of subsequent ECDLP instances. For all these methods, the required pre-computation requires at least $\sqrt{\ell}$ operations, which we are already assuming is out of reach. It is also worth noting that Bernstein and Lange explain that the claimed speedup doesn't appear when one when considers the cost of memory access in the AT metric. Finally, Koblitz and Menezes [54] have critiqued the relevance of the non-uniform model in cryptographic situations. In conclusion, we do not consider these approaches a meaningful threat to Curve25519, since we are already assuming that 2^{128} operations is beyond the power of any realistic attacker within the next 10 years.

4.6 Future mathematical attacks

It is important to acknowledge that the hardness of ECDLP is an assumption that is only supported by the fact that many researchers have tried to break it, and have not succeeded in finding a general attack that is better than Pollard rho. It is in principle possible that some new mathematical idea will be found in the near future that requires a re-evaluation of the classical security of elliptic curve cryptosystems.

Indeed, some standards have "hedged" against future attacks by avoiding special cases even when an attack was not known. One example of this is the Brainpool standard [59], which chose to avoid elliptic curves with complex multiplication by an order of small class number. Precisely the standard states:

> The class number of the maximal order of the quotient field of the endomorphism ring $End(E)$ of E is larger than 10^7. Generally, E cannot be "lifted" to a curve E' over an algebraic number field L with $End(E) = End(E')$ unless the degree of L over the rationals is larger than the class number of $End(E)$. Although there are no efficient attacks exploiting a small class number, recent work ([JMV] and [HR]) also may be seen as argument for the class number condition.

The reference [JMV] is to a paper by Jao, Miller and Venkatesan about the random self-reduction of the ECDLP that is generally considered to be a positive result for the security of ECC. The reference [HR] is to the paper of Huang and Raskind on "Signature Calculus", which does not seem to have led to any practical attack on the ECDLP.

The Brainpool document is a reasonable use of the precautionary principle to avoid a special case. But it is notable that in the subsequent 10 years no attack on small class number curves (even class number 1) has been found. Hence, one could consider the Brainpool standard as over-cautious, and as evidence that there are no major attacks on the ECDLP using current mathematics that have been missed by current researchers.

Over the last 10 years there has been great activity in cryptanalysis of all sorts of schemes, including major advances in finite field discrete logarithms [6]. There is a large community of mathematical cryptographers, and high awareness among number theorists and algebraic geometers of the importance of the ECDLP. I believe that many great mathematicians have put some thought into ECDLP in recent decades. However it is notable that there has been no

substantial progress in the general case of the ECDLP for at least 20 years. For this reason, I feel it is unlikely that someone will suddenly realise that an existing mathematical idea (at least within number theory or algebraic geometry) provides a breakthrough in algorithms for ECDLP.

One other possibility is that someone discovers an entirely new set of mathematical tools, and that this leads to improved algorithms for ECDLP. It is certainly true that, occasionally, a brilliant researcher creates new techniques or approaches (these are the sort of people who win Fields Medals). It is impossible to predict such future events. For example, I hope that someday someone will create the tools that allow us to settle the P=NP question. My opinion on the development of mathematics is that these major events happen rarely, perhaps only a few times each year (much more common is the discovery that tools from one area of mathematics can be used to advance research in another area). Considering the breadth of mathematics, most such discoveries would be unrelated to elliptic curves or algorithmic problems. So it again seems unlikely (though not impossible) that there will be a breakthrough new idea for ECDLP within the next 5-10 years.

5 The security of Curve25519

The specific EdDSA curve over $\mathbb{F}_p$ where $p = 2^{255} - 19$ has 8ℓ points where ℓ was specified earlier. It is a specific curve over a specific field. Should there be any concerns?

The prime is of course chosen to enable fast modular reduction. There has never been any attack on ECDLP that exploits the particular structure of the field (apart from some results on pairing attacks that improve the finite field DLP using the special number field sieve, but such ideas are not relevant in this case). Hence, I do not have any concern about using prime fields of this special form.

The trace of Frobenius is

$$t = p + 1 - 8\ell = -2219385422189788282868155023270691879 62.$$

This shows that the curve is not anomalous (as anomalous curves over prime fields have trace $t = 1$). Moreover, we have

$$D = t^2 - 4p = -2^4 \cdot 16451 \cdot 8312956054562778877481 \cdot 8332672572899929670107862883852213333365522455698 7.$$

This means that the curve is not a "small disriminant CM curve" and does not appear to have any special structure of concern. Note that the discriminant D is divisible by 2^4 and so is not the discriminant of a maximal order. The quadratic field $\mathbb{Q}(\sqrt{D})$ has discriminant $D/2^2$. I have not been able to compute the class number of the field, but I have verified that the ideal class $(23, 18 + \sqrt{D/2^4})$ has order greater than 1000. This shows that the class number is greater than 1000. Of course we expect the class number to be of size approx $\sqrt{|D|} > 2^{125}$.

I have checked that the embedding degree is greater than 1000. So the curve is safe from the MOV/Frey-Rück pairing attack. This exceeds the requirements of the ECDSA standard [1] of embedding degree at least 100. Indeed, the extension degree is

$$k = 1206167596222043702328864427173832373476186059896651267666991823047575708498 = (\ell - 1)/6.$$

Hence the curve satisfies all the security requirements of the Brainpool standard [59], apart from not having prime order.

One important feature of Curve25519 is that the quadratic twist also has nearly prime order (precisely, $p + 1 + t = 4\ell'$ for a prime ℓ'). This is a deliberate choice for key exchange/encryption protocols using the Montgomery ladder and x-coordinate arithmetic. Such features are not used in digital signature schemes, so in some sense the curve is more secure than needed.

Another question is whether the curve is deliberately chosen to have some secret weakness. My first comment is that no such weakness is known, for any types of elliptic curves. So such concerns are not supported by evidence but only by a sense of paranoia. The second comment is that the curve arises from an elliptic curve with very small coefficients. Precisely, the curve arises from the famous Curve25519

$$v^2 = u^3 + 486662u^2 + u$$

over $\mathbb{F}_p$ where $p = 2^{255} - 19$ that was computed by Daniel Bernstein [12]. Bernstein's motivation was to find a curve with nearly prime order, small coefficients, and such that the quadratic twist has nearly prime order. At the very end of [12] there is a short discussion about why the particular constant 486662 was chosen: "The smallest positive choices for A are 358990, 464586, and 486662. I rejected $A = 358990$ because one of its primes is slightly smaller than 2^{252}, raising the question of how standards and implementations should handle the theoretical possibility of a user's secret key matching the prime; discussing this question is more difficult than switching to another A. I rejected 464586 for the same reason. So I ended up with $A = 486662$."

In short, it seems that Curve25519 is essentially the inevitable result of a sensible and justified search for good curves with small coefficients. Hence it is reasonable to consider that Curve25519 is an honestly generated elliptic curve with no special weakness known to its creator.

One can find an isomorphism from the Montgomery model for Curve25519 to a twisted Edwards form. More generally, if $y^2 = x^3 + Ax^2 + x$ is a Montgomery model for an elliptic curve over K, then there is an isomorphism to $aX^2 + Y^2 = 1 + dX^2Y^2$ for $a = A + 2$ and $d = A - 2$ (see Lemma 9.12.20 of [43]). This would lead to the twisted Edwards curve

$$486664X^2 + Y^2 = 1 + 486660X^2Y^2.$$

Instead, Bernstein and Lange [14] choose to take an isomorphism to the Edwards curve

$$x^2 + y^2 = 1 + (121665/121666)x^2y^2$$

which, since -1 is a square, is also isomorphic to

$$-x^2 + y^2 = 1 + (-121665/121666)x^2y^2.$$

The specific isomorphism is as follows: Let $A = 486662$, $d = 121665/121666$, and let (u, v) be an affine point on the Montgomery curve. Set $x = \sqrt{A+2}\,(u/v)$ and $y = (u-1)/(u+1)$. Then (x, y) satisfies $1 + dx^2y^2 = x^2 + y^2$ as required. The point at infinity on the Montgomery curve maps to $(0, 1)$. Since d is not a square in the field the group law is complete.

If one accepts that the original Curve25519 is honestly generated and has no known weakness, then it follows that this equation for the EdDSA curve is also honestly generated and safe.

Finally, there is the choice of the base point B. Due to the random-self-reducibility of ECDLP there is no way for some base points to be weaker than others. Hence the proposed base point is safe.

6 Detailed description of EdDSA

The EdDSA protocol contains a number of optimisations. One of the most striking features is the fact that the private key is a bitstring and that hashing is used during the protocol to derive other secret values from this value. The more traditional approach to signature protocols has the private key being an integer (namely the solution to an ECDLP instance arising from the public key), but in this setting the private key is really the seed which is used to generate the public key and also for generating ephemeral values.

The fixed system parameters are an elliptic curve E over a finite field $\mathbb{F}_p$ and a point B of prime order ℓ. We also fix an integer b such that $p < 2^b$ and a hash function H with $2b$-bit output. Finite field elements are represented as integers in the set $\{-(p-1)/2, \ldots, -1, 0, 1, \ldots, (p-1)/2\}$. In practice $p = 2^{255} - 19$ and $b = 256$. See [18, 19, 56, 52] for other curves and extensions.

6.1 Public keys and private keys

The private key of a signer Alice is a b-bit binary string $k \in \{0,1\}^b$. The hash of k is $H(k) = (h_0, h_1 \ldots, h_{2b-1}) \in \{0,1\}^{2b}$. In practice one can think of either $k \in \{0,1\}^b$ or $H(k) \in \{0,1\}^{2b}$ as being the private key. From this hash value Alice computes the integer

$$a = 2^{b-2} + \sum_{i=3}^{b-3} h_i 2^i.$$

Note that a is a multiple of 8 and that $2^{b-2} \leq a < 2^{b-1}$. For the Curve25519 implementation this is $2^{254} \leq a < 2^{255}$. Another way to view a is that its binary expansion has three least significant bits set to 0 and the $(b-2)$-th bit set to 1; this way of fixing the values of certain bits is called **key clamping**.

Alice's public key is a representation of the point $A = [a]P$. Specifically she computes the compressed representation $\underline{A}$ of A (see Section 6.4 for details).

Since $\ell \approx 2^{b-3}$ we actually have $\ell < a < p$ and so it is also appropriate to reduce a modulo ℓ before exponentiation (though this is not a serious matter as the speedup is very small and the operation is only performed once, during key generation). It is important to note that, once we reduce a modulo ℓ, the values are not uniformly sampled from $\{0, 1, \ldots, \ell - 1\}$ as one would traditionally require. The fact that a is a multiple of 8 does not reduce the key space, since 8 is coprime to ℓ. However it is clear that a depends only on $(h_3, h_4, \ldots, h_{b-3})$ and so there are exactly 2^{b-5} (which is 2^{251} in the standard 25519 setting) possible choices for a. Since $\ell \approx p/8 \approx 2^{252}$, it follows that the key space is about half the size it should be. This means that, in principle, one can solve the ECDLP of Alice's public key using the baby-step-giant-step algorithm in time $1/\sqrt{2}$ compared with the usual case of uniformly sampled $a \in \mathbb{Z}_\ell$. (The details are straightforward: Write $a = 8(a_0 + Ma_1)$ with $0 \leq a_0 < M$ and $0 \leq a_1 < M + 1$, for $M = \lfloor 2^{(b-5)/2} \rfloor$.) Of course, in practice we would use Pollard rho, and the reduction in key space does not result in any speedup of the attack. The reduction in key space is not large enough to get any benefit from using the Pollard kangaroo method.

I do not know why the EdDSA specification uses key clamping. The key generation algorithm still requires access to a reliable source of randomness to create the private key $k \in \{0,1\}^b$. So it seems reasonable to just choose the integer a to be a uniformly generated integer modulo ℓ. On a practical level the key clamping does not affect the scheme much, and intuitively should not influence security. But from a theoretical point of view it causes a number of difficulties in the security proofs, as I will explain later.

The public key size is 32 bytes and the private key size is 32 bytes for k (or 64 bytes if one prefers to store $H(k)$).

6.2 Signing

To sign a message m, Alice does the following. First, if the message is long, she may "pre-hash" the message using a hash function PH to get a smaller message tag (there seem to be variants that do this, but the original Ed25519 paper takes PH to be the identity.). Then she hashes her private key k to get $h = H(k)$. The lower half of h is used to recompute her secret a, as explained above. The higher half of h is used to compute the ephemeral value r for signing. Specifically, Alice computes the "random" $2b$-bit integer $r = H(h_b, h_{b+1}, \ldots, h_{2b-1}, \mathsf{m})$. Here the notation $H(h, \mathsf{m}) = H(h \| \mathsf{m})$ means that the binary strings h and m are concatenated before signing. Note that $0 \leq r < 2^{2b}$ and the distribution of r should be very close to uniform if there is sufficient entropy in the message space.[1] Then Alice computes $R = [r]B$ (the most efficient way to do this is to reduce r modulo ℓ first, and this is what all implementations do as far as I know) and then computes the compressed representation $\underline{R}$. In traditional Elgamal or Schnorr signatures the value r would be chosen randomly and not repeated, whereas EdDSA uses a deterministic signing algorithm. Finally Alice computes

$$S = r + H(\underline{R}, \underline{A}, \mathsf{m})a \pmod{\ell}.$$

The signature is $(\underline{R}, S)$. (Technically we sometimes use a specific representation $\underline{S}$ for S, but this is not relevant to security.)

The signature size is 64 bytes.

A standard framework to design and analyse signature schemes of this type is to first consider interactive identification schemes (also called Σ-protocols) which consist of a commitment, then a challenge, and then a response. The Fiat-Shamir transform is a standard technique to turn such an interactive protocol into a non-interactive one. In the setting of EdDSA, the commitment is R (or $\underline{R}$) and the challenge is the hash value $H(\underline{R}, \underline{A}, \mathsf{m})$. It is notable that in EdDSA the public key $\underline{A}$ is included in the Fiat-Shamir transform; this is not always the case (for a recent discussion of these issues see Bernhard, Pereira and Warinschi [10]). The use of the public key in the hash is sometimes called **key prefixing**. We will discuss some of the security implications of this in Section 7.7, also see Section 5.3 of Brendel et al [31].

[1] Later we explain why the scheme is secure even when messages are chosen from a low entropy space.

6.3 Verifying

To verify a signature, given the public key $\underline{A}$ of Alice, a verifier computes the points A and R from $\underline{A}$ and $\underline{R}$ and checks they are points on E. The verifier should also check that $0 \leq S < 2^b$. Finally, the verifier computes $H(\underline{R}, \underline{A}, \mathsf{m})$, reduces it modulo ℓ, and checks whether

$$[8S]B = [8]R + [8H(\underline{R}, \underline{A}, \mathsf{m})]A \tag{2}$$

and the signature is accepted if all checks are true.

Of course, as with all public key cryptography, it is necessary to ensure the authenticity and validity of the public key $\underline{A}$. This typically involves some PKI and will include checking certificates and revocation lists. These issues are universal and we do not discuss them further.

Validation checks: While it is clear that every implementation should check equation (2), some other checks should be performed: It should be checked that B, A and R are points on $E(\mathbb{F}_p)$. One could also check that these points have order ℓ (which is expensive) or that their order is divisible by ℓ (by checking that $[8]B, [8]A$ and $[8]R$ are not the identity). Some documents also suggest checking that $0 \leq S < \ell$, which matters if one is concerned with strong forgery (see Section 7). There is variation in implementations, for example see the blog post by Henry de Valence [78], and Chalkias, Garillot and Nikolaenko [33].

Validation of public keys is a topic with a long history. It can mean several things, such as checking that the public key is an element of a correct subgroup, or proving to a PKI that a user knows their secret key (see Section 6 of [51]). In the context of public key encryption there have been attacks based on providing points of the incorrect order or in an incorrect group. Such attacks are less serious in the context of signatures, since the signer uses their private key on values that they themselves have computed (rather than on adversarially chosen group elements). Hence we do not explore the topic further in this report, apart from to confirm that a verifier should check that Alice's public key $\underline{A}$ really does correspond to a point of the correct order on the correct elliptic curve. This check is usually implied by verifying Alice's public key certificate. It could be performed every time during signature verification, or could be performed once if Bob is communicating regularly with Alice and caches a copy of her public key.

The multiplication by the co-factor 8 is an interesting choice in the protocol. In good implementations this should not be necessary: If A and B have been checked to lie on E and to have the correct order ℓ, then equation (2) implies that $[8]R$ lies on E and has order ℓ.

If S is such that equation (2) is satisfied, then $S \pmod{\ell}$ also satisfies the equation. So there is no security gain from checking the range of values for S. The only reason that I know why one would be concerned with checking the size of S is to worry about strong existential forgery, which seems unnecessary.

Note that r is not uniformly sampled in $\mathbb{Z}_\ell$, but the bias is very small since we are essentially reducing a "random-looking" 512-bit integer to get a random 252-bit integer r. There is no reason to think that computing r from $\underline{R}$ is any easier than a truly random ECDLP instance.

Variants: We have presented the version that computes the point R from $\underline{R}$ and then checks equation (2). A variant (which we presented in the introduction) is to compute

$$R' = [S]B - [H(\underline{R}, \underline{A}, \mathsf{m})]A$$

and to check if R' compresses to $\underline{R}$. Again, if we are certain that A and B lie on $E(\mathbb{F}_p)$ and have order ℓ, then $R' \in E(\mathbb{F}_p)$ has order ℓ and so it should happen that $R' = R$ and so it compresses to $\underline{R}$. This scheme is equivalent to the one presented, and so there is no loss of security from using this variant. This variant may be more efficient for single signatures as it avoids computing square roots that are needed to decompress $\underline{R}$, but it is less efficient for batch verification.

6.4 Point decompression/decoding

A point $R = (x, y)$ is represented in compresssed form as (y, b) where $y \in \mathbb{F}_p$ and b is the sign bit of x (namely, it is 0 if $0 \leq x < p/2$ and 1 if $-p/2 < x < 0$.

Suppose we are given a compressed point (y, b) and wish to recover the original point. We need to solve for x in $x^2 = (y^2 - 1)/(dy^2 + 1) \pmod{p}$ which is taking a square root modulo p (note that the denominator is always

non-zero modulo p). We are not in the easiest case $p \equiv 3 \pmod 4$, but there is still a simple way to do the calculation. Note that when $p = 2^{255} - 19$, $\alpha = 2^{(p-1)/4} \pmod p$ satisfies $\alpha^2 = -1 \pmod p$.

Let $u = y^2 - 1$ and $v = dy^2 + 1$. Compute

$$x' = (u/v)^{(p+3)/8} = uv^3(uv^7)^{(p-5)/8} \pmod p$$

which combines the inversion and the exponentiation. It is easy to check that $(x')^2 = (u/v)^{(p+3)/8}$.

It now follows that either $v(x')^2 = u \pmod p$, in which case x' is a square root of u/v, or $v(x')^2 = -u \pmod p$, in which case $x'\alpha$ is a square root of u/v (so re-define $x' = x'\alpha$). There is a third possibility which should not occur, namely that $v(x')^2 \not\equiv \pm u \pmod p$, which means there is no rational point with the given y-coordinate. Once we have a suitable solution x' to the quadratic equation we use the bit b to decide the choice. There are four cases: If $b = 0$ and $0 \le x' < p/2$ then we set $x = x'$; If $b = 0$ and $p/2 < x' < p$ then set $x = p - x'$; If $b = 1$ and $0 \le x' < p/2$ then we set $x = p - x'$; If $b = 1$ and $p/2 < x' < p$ then set $x = x'$. Return the decoded point (x, y).

6.5 Fast batch verification.

As noted earlier, there are two ways to verify a signature $(\underline{R}, S)$ on a message m for public key $\underline{A}$. For both methods the verifier first computes $H(\underline{R}, \underline{A}, \mathsf{m})$ and reduces it modulo ℓ. One way to verify is to decompress $\underline{R}$ and then check whether

$$[8S]B = [8]R + [8H(\underline{R}, \underline{A}, \mathsf{m})]A$$

The other way is to compute

$$R' = [S]B - [H(\underline{R}, \underline{A}, \mathsf{m})]A$$

and to check if R' compresses to $\underline{R}$.

Now suppose we have a collection $(\underline{R}_i, S_i)$ of signatures on messages m_i for $1 \le i \le t$. Let $H_i = H(\underline{R}_i, \underline{A}, \mathsf{m}_i) \pmod \ell$ and let R_i be the decompression of $\underline{R}_i$ for each i. We have $[8S_i]B = [8]R_i + [8H_i]A$. The verifier now chooses t random 128-bit integers z_i and computes

$$S = \sum_{i=1}^{t} z_i S_i \pmod \ell \quad \text{and} \quad H = \sum_{i=1}^{t} z_i H_i \pmod \ell.$$

Then it suffices to check that

$$[8S]B = [8H]A + \sum_{i=1}^{t} [8z_i]R_i.$$

Using fast methods to compute multi-point exponentiations, this computation is much faster that verifying t signatures individually. One can do something similar when the signatures are all for different public keys as well. For more details see [18].

The security relies on the unpredictability of the integers z_i chosen by the verifier. So these values must be chosen randomly. There is always a negligible chance that a batch of signatures that contains 2 or more invalid signatures may pass the test and so be declared as correct.

In contrast, for ECDSA it is much less simple to get fast batch verification. There are a number of papers on the topic, many of them making variations to the ECDSA protocol to enable fast batch verification. Antipa, Brown, Gallant, Lambert, Struik and Vanstone [2] were the first to address the issue. They gave a variant called ECDSA* that permits fast batching, but also noted "the advantages can also be enjoyed if the signer appends a small number of bits of side information to standardized ECDSA signatures" (which is also a deviation from the standard). Subsequent works include Cheon and Hyun Yi [35], Karati et al [53] and many more. We do not give a complete analysis of the situation for ECDSA. Suffice that the design of EdDSA is customized to enable fast batch verification.

7 Security of EdDSA

The widely accepted minimum standard of security for a digital signature scheme in the single-user setting is existential unforgeability under a chosen message attack. This notion considers an adversary who can ask for signatures to be generated (with respect to some fixed public key of the entity that is under attack) on messages of its choosing. The adversary wins if it can then generate a new signature on a message.

The reason for considering such strong security notions is *not* because this reflects a common situation in practice. I am not aware of any practical situation where an attacker wants to get a signature on message m for a given public key, but has access to a signing oracle that will sign *every single message except for* m. This notion clearly makes no sense in the real world. In many real-world applications an attacker does not have access to a sign oracle but only sees a number of valid signatures on known messages. Also, in most cases it is not sufficient to sign a "random" message but an attacker must forge a signature for a meaningful chosen message. The motivation for strong security models is that it allows to take a conservative approach: if we can show that our signature scheme resists attacks in a model where we are being very generous to an attacker (by giving them access to a signing oracle, and by setting a minimal forgery requirement for them to win) then it gives us confidence that the scheme is also secure in more realistic settings.

There are two flavours of security, and the difference is subtle: an **existential forger** wins if they can generate a valid signature on a message that was not one of the messages queried to the sign oracle; a **strong existential forger** wins if they can generate a valid signature that was not one of the outputs of the sign oracle. In other words, a strong forger wins even if it generates a *new signature on a previously signed message*. When one considers strong existential forgers then one has to pay attention to quite trivial issues. For example, in EdDSA we have already discussed in Section 6.3 that the value S in a signature is parsed in some implementations as a b-bit string, and in others as an integer modulo ℓ. Since $\ell \approx 2^{b-3}$ it follows that if $\sigma = (\underline{R}, S)$ is a signature on a message m, where $0 \leq S < \ell$ then $\sigma' = (\underline{R}, S + \ell)$ would also be a valid signature on a message m, because it still satisfies equation (2) and $S + \ell$ can be represented as a b-bit string. If one wanted to be strongly secure then one would have to ensure that the verification algorithm only accepts values $0 \leq S < \ell$, so that this "attack" can be avoided.

From a theoretical point of view, I understand the interest in studying strong existential forgery. But from a practical point of view, in my opinion, it is not necessary to consider strong security. A typical example to motivate strong security is using digital signatures to provide authentication of encryption: To get CCA-security of the encryption scheme one needs strong security of the signature, or else an attacker can modify a challenge ciphertext by modifying the signature component and then calling the decryption oracle to get the message. However, this also assumes the literal reality of the CCA-game, where an attacker has access to a decryption oracle that will decrypt every ciphertext except the challenge ciphertext. As I have already mentioned, these notions do not actually reflect the way crypto is used in practice and so it seems unnecessary to take them into account in the security analysis. So for this report I focus only on existential forgery, and so for security it does not matter whether or not the verification algorithm requires the value S to be in the range $[0, \ell)$.

We now state a precise definition of security. We give the definition in the concrete security setting, where we consider an adversary that runs in time t and succeeds with probability ϵ, rather than polynomial-time adversaries and negligible success.

Formally, a signature scheme consists of three randomised algorithms: KeyGen, Sign, Verify. The randomised algorithm KeyGen takes as input a security parameter λ and outputs (in time polynomial in λ) a pair $(\mathsf{pk}, \mathsf{sk})$ consisting of a public key pk and a private key sk. The randomised algorithm Sign (though for EdDSA it is a deterministic algorithm) takes as input the private key sk and a message m and outputs (in polynomial time) a signature σ. The algorithm Verify (which is usually deterministic) takes as input pk, m and σ and outputs (in polynomial time) 1 (for 'valid' or 'accept') or 0 (for 'invalid' or 'reject').

Definition 1. *The signature experiment with security parameter λ consists of the following game between an attacker $\mathcal{A}$ and a challenger:*

- *The challenger runs* KeyGen *on input λ to get* $(\mathsf{pk}, \mathsf{sk})$.
- *The challenger runs $\mathcal{A}$ on input* pk.
- *The attacker $\mathcal{A}$ may ask for sign queries on messages* m *of its choice, and is provided with the output σ of* $\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$. *Let L be the list of all messages* m *queried to the* Sign *oracle.*

– *The attacker outputs a pair* (m^*, σ^*) *and wins if* $\mathsf{m}^* \notin L$ *and if* $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}^*, \sigma^*) = 1$.

A signature scheme is ***existentially unforgeable under a chosen message attack (EUF-CMA)*** *if for every attacker* $\mathcal{A}$ *that runs in time* t *and wins with probability* ϵ *then* $t/\epsilon > 2^\lambda$.

7.1 The random oracle model (ROM)

There is a large literature on the security of Schnorr signatures, DSA signatures and ECDSA signatures, including proofs in the random oracle model and proofs in the generic group model. It is beyond the scope of this report to survey the entire field.

The random oracle model is a very commonly used heuristic in theoretical cryptography. The main idea is to replace one or more hash functions in a cryptographic protocol with "random functions". The meaning of a "random function" is as follows: Let $m, n \in \mathbb{N}$ and consider a hash function $H : \{0,1\}^m \to \{0,1\}^n$; then a random function is a function chosen uniformly at random from the set of all possible functions from $\{0,1\}^m \to \{0,1\}^n$. The only way to represent a random function is as a look-up table. Since there are 2^m possible inputs, and each is an n-bit string, it follows that there are

$$(2^n)^{2^m} = 2^{n2^m}$$

possible random functions from m-bits to n-bits. Since such an object has exponential size it cannot actually be used in cryptography. There is no way to implement a random oracle in practice or to build a cryptosystem that uses a random oracle. However, a key insight is that one can *simulate* a random oracle for any polynomial-time algorithm by constructing the look-up table on-the-fly; since the algorithm has polynomial-time it can make only polynomially-many queries to the random oracle and hence the table only has polynomial size and can be simulated perfectly in polynomial-time (assuming the simulator has access to enough truly random bits).

The power of the random oracle model in theoretical cryptography is that the hash function is no longer "internal" to the protocol. The attacker must make queries to the random oracle, which means that the challenger can learn what values are being put into the hash function by the attacker. In other words, the challenger can "see what is going on inside the attacker" in a way that would not be true in real-world attacks on cryptography. The other power of the random oracle model is that it allows the challenger to specially calculate responses to random oracle queries. This is sometimes called *programming* the random oracle. It allows the challenger to do things that would otherwise be hard to do, such as generate signatures on messages without known the private key. It also allows the challenger to do powerful thought-experiments with an attacker, such as re-winding (see below for more discussion).

In short, the random oracle model is not about the randomness of hash functions, but is a sneaky way to get security proofs to work.

While the random oracle model does not have any meaning in practice, it allows to give some degree of theoretical validation to practical schemes. The **random oracle heuristic** is that if a scheme is proved secure in the random oracle model and if the random oracle is replaced by a "good" hash function (e.g., one that has uniformly distributed output, is collision-resistant, and is not based on the same sort of mathematics as the signature scheme) then the scheme is secure. The random oracle heuristic is not a mathematical fact; it is only a belief. It is known that there are cryptosystems that are secure in the random oracle model but insecure when implemented with any hash function, but these cryptosystems are "artificial" in the sense they are clearly designed to create counterexamples to the random oracle heuristic and would never naturally arise from any attempt to build a cryptosystem for real-world use.

In practice, I believe the use of the random oracle heuristic is sensible and does provide some intuition and assurance about the security of systems that are intended to be practical.

7.2 Relationship with ECDSA and Schnorr signatures

The original work [18] on EdDSA did not discuss the provable security of the protocol. Due to the high-level similarities of the design, it is plausible to think that if there was an attacker against EdDSA then it could be turned into an attacker against ECDSA or Schnorr signatures. If that is true then the security of EdDSA is inherited from the extensive security analysis of ECDSA and Schnorr signatures. This would mean that the security of EdDSA in the random oracle model follows directly from previous results.

We now briefly explain that there is not actually a reduction between ECDSA signatures and EdDSA signatures. To recall, an ECDSA signature on public key $(B, A = [a]B)$ and message m is (r, s) such that if $u_1 = H(m)s^{-1} \pmod{\ell}$, $u_2 = rs^{-1} \pmod{\ell}$, and

$$X = [u_1]B + [u_2]A.$$

then X is not the identity and when we convert X to an integer r' we have $r' \equiv r \pmod{\ell}$. If one attempts to pass between EdDSA and ECDSA one could try to treat r and R as interchangeable (there are actually a number of subtleties in this, which we ignore for the moment) and imagine that messages in ECDSA correspond to the string $m = \underline{R}\|\underline{A}\|\mathsf{m}$ where m is the message in EdDSA. In this setting, ECDSA signatures are a pair (R, s) such that

$$R = [H(m)s^{-1}]B + [rs^{-1}]A = [H(\underline{R}, \underline{A}, \mathsf{m})s^{-1}]B + [S']A$$

where $S' = rs^{-1} \pmod{\ell}$. In comparison, EdDSA verification checks the equation

$$R = [S]B - [H(\underline{R}, \underline{A}, \mathsf{m})]A. \tag{3}$$

These equations are not precisely of the same form, and I am not aware of any simple algebraic manipulation or change of view that allows to transform one to the other. Hence, as far as I am aware, one cannot derive the security of EdDSA in the random oracle model from existing results about the security of ECDSA.

There is more-or-less a direct reduction between EdDSA and Schnorr signatures (apart from some subtleties in the distribution of public keys due to the key clamping in EdDSA). First we recall the Schnorr signature scheme: The public key is (B, A) such that $A = [a]B$, where a is uniformly sampled. To sign a message m, Alice generates a random ephemeral value r, computes $R = [r]B$, $c = H(R, m)$ and $S = (r + ca) \pmod{\ell}$. The signature is (c, S). To verify the signature (c, S) on message m for key A we compute

$$H([S]B - [c]A, m)$$

and check this is equal to c.

Since point compression is often used in elliptic curve crypto we will assume that Schnorr signatures are actually implemented using $H(\underline{R}, m)$. In this case, a valid EdDSA signature $(\underline{R}, S)$ on message m corresponds to the Schnorr signature $(c, S) = (H(\underline{R}, m'), S)$ on message $m' = \underline{A}\|\mathsf{m}$. To see this, note that

$$R = [S]B - [H(\underline{R}, \underline{A}, \mathsf{m})]A = [S]B - [H(\underline{R}, m')]A = [S]B - [c]A.$$

And thus the check $c = H(\underline{R}, m')$ will hold.

Conversely, for a valid Schnorr signature (c, S) on message $m' = \underline{A}\|\mathsf{m}$ define $R = [S]B - [c]A$. Then $(\underline{R}, S)$ is a valid EdDSA signature on message m. The requirement that the message be appended with the public key value $\underline{A}$ is awkward, and can be avoided by modifying the random oracle in the Schnorr scheme, so that the random oracle $H^{\text{Schnorr}}(\underline{R}, m)$ is defined to be the value of the random oracle $H^{\text{EdDSA}}(\underline{R}, \underline{A}, m)$. The process of adding the public key to the message was also used in Section 2.5 of Bernstein [13], where he called it **key tag prefixing**.

In summary, apart from the constraints on the private key a and the differences in the methods of generating the ephemeral value r, one can transform a signing oracle for one scheme into a signing oracle to the other, and a forgery for one scheme into a forgery for the other. It follows that any state-of-the-art security proof for Schnorr signatures more-or-less immediately gives a security proof for EdDSA.

Section 3.2.2 of Pointcheval and Stern [67] gives a security proof for Schnorr signatures in the random oracle model using the forking lemma. Security proofs in the random oracle model based on this approach are also described in textbooks, such as Section 22.1 of [43] and Section 19.2 of Boneh and Shoup [24].

7.3 Security of EdDSA in the random oracle model based on ECDLP

The discussion in Section 7.2 already gives a reasonably convincing argument about the security of EdDSA in the random oracle model. However there are a few technical subtleties that have not been taken care of carefully.

Brendel et al [31] have given a direct security proof of EdDSA in the random oracle model, which bypasses the requirement to show the relationship between a forger for EdDSA and a forger for Schnorr (in particular, it directly handles the methods to generate ephemeral values). We state their result and sketch their proof below.

Theorem 1. *(Theorems 1, 2 and 3 of [31] combined and simplified) Let $H : \{0,1\}^* \to \{0,1\}^{2b}$ be a random oracle.*

Suppose there is a EUF-CMA adversary $\mathcal{A}$ for the EdDSA signature scheme that runs in time t, succeeds with probability ϵ and makes Q_H queries to the random oracle of the form $H(\underline{R}, \underline{A}, \mathsf{m})$ (which includes implicit queries invoked by calls to the Sign *oracle). Then there is an algorithm to solve the ECDLP that runs in time approximately $2t$ and succeeds with probability*

$$\tfrac{1}{2}(\epsilon/Q_H)^2 + \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter.

Concretely, this means that an attacker against the signature scheme that runs in time 2^{79} operations, makes $2^{20}-1$ queries of the given form to the random oracle, and succeeds with probability $\epsilon = 2^{-4}$ corresponds to an algorithm for ECDLP that runs in time 2^{80} operations and succeeds with probability approximately 2^{-49}. This shows the loss of tightness in the reduction, as is usual with proofs based on the forking lemma.

The result can be improved to $\frac{1}{2}\epsilon^2/(Q_H+1) + \mathsf{negl}(\lambda)$ by using the Bellare and Neven version of the forking lemma [8].

For the proof, let $(B, A = [a]B)$ be an ECDLP instance. We are creating an algorithm (called the Simulator) that will solve the ECDLP problem, by using the adversary $\mathcal{A}$ as a subroutine.

The first issue is that (B, A) may not be a valid EdDSA public key, since a may not be consistent with the key clamping in the EdDSA specification. The simplest thing to do is to randomise the instance by choosing uniformly at random an integer u such that $0 \leq u < \ell$ and computing $[u]A$. Obviously, if we can extract the discrete log of $[u]A$ then we can compute the discrete log of A. With probability $2^{b-5}/\ell \approx 1/2$ this is a valid public key for EdDSA. This explains the factor of $\frac{1}{2}$ in the theorem statement. So assume this is done, and we use the symbol A for the randomised value.

Now the Simulator runs the adversary $\mathcal{A}$ on the public key (B, A). The Simulator maintains a table of pairs (x, y) such that the random oracle H has been queried on x and answered with y; this table is initially empty.

The adversary will make a number of hash queries of different forms, and the Simulator can predict the expected value of this number (for example, by running the adversary a few times on random public keys with known private keys). In particular, the adversary will make some hash queries of the form $H(\underline{R}, \underline{A}, \mathsf{m})$, by which we mean the first b bits of the input represent a point on E of order ℓ, the next b bits represent the fixed public key point A, and there are some remaining bits that can be called a message. Let Q_H be the expected number of such queries. We may assume $Q_H \geq 1$ since the forgery cannot be correct with non-negligible probability without at least one such query (the existence of this query explains the $Q_H + 1$ formulation in [31]). These are the queries that matter in terms of the re-winding argument in the security proof. In order for the adversary to output a valid signature (m, σ) it is necessary that such a value is queried to the random oracle. The Simulator might even be able to predict which of the Q_H queries is likely to be the one related to the forgery, but in the worst case the Simulator can choose a random integer $1 \leq I \leq Q_H$ as a guess for which hash query will correspond to the forgery. The forgery output by $\mathcal{A}$ at the end of the game will correspond to the I-th hash query of this type with probability at least $1/Q_H$.

The idea of the Forking Lemma is that if $\mathcal{A}$ forges a signature successfully, such that the forgery corresponds to the I-th such hash query, then one can "re-wind" $\mathcal{A}$ and play it again but with the value of the I-th hash query changed to a different value. More precisely, we consider $\mathcal{A}$ as an algorithm that consults an external random source for its randomness. We run $\mathcal{A}$ with a fixed random source and answer all hash queries in a certain way. By "re-winding" we mean we run $\mathcal{A}$ again with the same random source and the same answers to hash queries except for the I-th one. Since $\mathcal{A}$ is behaving as a deterministic algorithm now, the behaviour of the two executions of $\mathcal{A}$ are identical up to that I-th hash query. So it is possible that $\mathcal{A}$ also outputs a forgery on the I-th query.

If the forking argument works and we get two signatures on the I-th query but with two different hash values then we have two signatures $(\underline{R}, S)$ and $(\underline{R}, S')$ such that $h = H(\underline{R}, \underline{A}, \mathsf{m})$ for the first signature and $h' = H(\underline{R}, \underline{A}, \mathsf{m})$ for the second. With overwhelming probability we have $h \not\equiv h' \pmod{\ell}$. It follows from the verification equation that

$$[S]B - [h]A = [S']B - [h']A$$

and so the solution to the ECDLP instance is $(S - S')(h - h')^{-1} \pmod{\ell}$. Since the probability $\mathcal{A}$ succeeds is ϵ, and also we need the forgery to correspond to the already chosen index I in both cases, the probability to solve the ECDLP using this approach is at least $(\epsilon/Q_H)^2$.

The above is the main idea of the proof, but now we pay attention to some other aspects of the simulation.

When $\mathcal{A}$ makes a Sign query on message m the Simulator proceeds in the usual way of random oracle proofs: Choose a random $0 \leq h < 2^{2b}$ and a random $0 \leq S < \ell$, compute $R = [S]B - [h]A$, compute $\underline{R}$, and define the random oracle as

$$H(\underline{R}, \underline{A}, \mathsf{m}) := h.$$

Then the Simulator returns $(\underline{R}, S)$ as the signature and adds $((\underline{R}, \underline{A}, \mathsf{m}), h)$ to the table of RO values. Sure enough, this signature verifies correctly, since by definition

$$[8S]B = [8]R + [8H(\underline{R}, \underline{A}, \mathsf{m})]A$$

as required. The remaining important detail is whether there is a conflict in defining the random oracle. The proof needs to consider the possibility that $H(\underline{R}, \underline{A}, \mathsf{m})$ has already been defined to some other value. Since the signature scheme is deterministic, if $\mathcal{A}$ makes two sign queries on the same message then the same signature should be returned (in other words, we don't generate random h and S the second time). So the only conflict is if H was already queried on the input $(\underline{R}, \underline{A}, \mathsf{m})$, and the probability of this is low since R is a uniformly sampled point of order ℓ (because S and h are uniformly distributed). The remaining observation is that the value h is a uniform binary string, and so this is consistent with the definition of a random function.

The simulator also has to handle random oracle queries directly, and for this we need to remember that H plays two roles in the protocol: First as $r = H(k, \mathsf{m})$ where k is the long-term secret key (totally hidden from the attacker's view) and also second as $H(\underline{R}, \underline{A}, \mathsf{m})$ in signatures themselves. The basic idea is to answer all such queries with a uniformly chosen binary string, and to add the input output pair to the table of values so that future evaluations of H can be consistently handled. Since k is not known it seems we do not need to consider any conflict of the form $r = H(k, \mathsf{m})$.

This ends the sketch of the proof. For full details of a security proof we refer to Brendel et al [31].

7.4 Security of EdDSA in the Random Oracle Model based on Multi-Base ECDLP

Very recently, Bellare and Dai [9] have given a new approach to proving the security of Schnorr signatures. An important feature of their proof is that it does not require rewinding, and hence does not incur an ϵ^2 tightness gap. Instead of proving security based on ECDLP they prove security under the following assumption.

Definition 2. *(**Multi-Base Discrete Logarithm Problem (MBDL)**): Let G be a group of prime order ℓ generated by* KeyGen *on input a security parameter λ. Let $(B, Y, A) \in G^3$ be uniformly sampled. Consider an algorithm $\mathcal{A}$ that takes (B, Y, A) as input, and is allowed to make* one *query to an oracle that returns the discrete logarithm of any group element in G to the base A. The algorithm $\mathcal{A}$ wins if it outputs the discrete logarithm of Y to the base B. The **MBDL assumption** is that no polynomial-time (in λ) algorithm $\mathcal{A}$ can succeed in this game with non-negligible probability.*

This problem has not been shown to be equivalent to ECDLP. It is unlikely to be equivalent to ECDLP since it is interactive (i.e., requires a query to an oracle). Bellare and Dai give evidence in the generic group model that MBDL should be hard. Overall, MBDL is a rather new and unstudied assumption, but it certainly seems an interesting assumption on which to base the security of a signature scheme.

Theorem 2. *(Theorem 4.3 of [9]) Let $\mathcal{A}$ be a polynomial-time EUF-CMA adversary against Schnorr signatures that makes at most $Q_H \geq 1$ queries to the random oracle and succeeds with probability ϵ. Then there is an algorithm for the MBDL problem that succeeds with probability $\epsilon/Q_H + \mathsf{negl}(\lambda)$ for some negligible function* negl.

I now sketch the idea of the proof. The Simulator is given the MBDL instance (B, Y, A). The Simulator runs the attacker $\mathcal{A}$ on the input (B, A). The Simulator also guesses the index I of the hash query used for the forgery. The difference with the usual proof is the way the Simulator responds to this hash query $H(R, m)$. The idea is to take the point Y in the MBDL instance and to use the one query to the discrete logarithm oracle to compute an integer c such that $[c]A = Y - R$. Then we program the random oracle so that $H(R, m) = c$. When the adversary outputs a forgery, the value S will satisfy the equation

$$[S]B = R + [c]A.$$

Now note that $[c]A = Y - R$ and so $[S]B = R + Y - R = Y$ and S is the required solution to the MBDL instance.

This proof technique immediately adapts to EdDSA, since it is based on the same verification equation $[S]B = R + [H(\underline{R}, \underline{A}, \mathsf{m})]A$. It follows that there is a tighter security proof for EdDSA, though based on a non-standard and interactive assumption. It is a matter of taste which result should be used to set parameters.

7.5 Security of EdDSA in Generic Group Model or Algebraic Group Model

It is possible to study the security of signature schemes in the generic group model or algebraic group model. Brown [29] gave such an analysis for ECDSA, and one can find discussion and critique about ECDSA in the generic model in the CRYPTREC reports by Stern [76] and by Boneh and Rogaway [23]. The Algebraic Group Model is a more recent variant, which is a little simpler to work with. I expect that such security proofs can be adapted to the case of EdDSA. In my opinion, the generic group and algebraic group models are of limited value to assessing the security of cryptographic protocols, as they are too dissimilar to real groups used in cryptography. Hence I do not consider them any further in this report.

7.6 Security of EdDSA in the Standard Model

We do not have the tools to prove security of practical signature schemes like EdDSA in the standard model (i.e., without any abstraction such as a random oracle or a generic group). In principle, one would have to consider a fixed hash function and prove that it does not interact in any unexpected way with the group.

Nevertheless, one can consider types of natural attack, and see how they are avoided. There are two main requirements of the hash function for EdDSA:

- Uniformity of outputs.
  This is needed for generating the ephemeral values r in EdDSA, due to the deterministic signatures property. The hash is also used to create the challenges in the Fiat-Shamir transform, and these must be unpredictable for signatures to be convincing to the verifier: if H was highly biased then a forger would be able to guess the hash value with non-negligible probability and hence construct a forgery. Hence we need the outputs to be well-destributed, even if the inputs are from a relatively low entropy source.
- Second preimage resistance.
  Suppose an attacker sees a signature $\sigma = (\underline{R}, S)$ on a message m for public key A. If the attacker can find a binary string $\mathsf{m}' \neq \mathsf{m}$ such that $H(\underline{R}, \underline{A}, \mathsf{m}) = H(\underline{R}, \underline{A}, \mathsf{m}')$, then σ is also a signature on m'. Hence second preimage resistance is essential.

It is conjecture that EdDSA is secure when implemented with any hash function that has uniform outputs and second preimage resistance. The ANSI ECDSA standard [1] allows any ANSI approved hash function. For EdDSA I consider any standard hash algorithm with 512-bit output to be secure, for example SHA-512 or SHA-3.

A recent paper by Chen et al [34] makes the very strong suggestion that a linear hash function would be secure for Fiat-Shamir signatures when the message space is very small (smaller than the output size of the hash function). This result does not seem to have practical implications on the security of EdDSA since we usually consider large message spaces.

7.7 Multi-user model

The security analysis above is for a single user with a fixed public key A. Because there is a fixed elliptic curve for EdDSA signatures, we are in the setting where many users are sharing the same parameters. Hence we should consider security in the multi-user model. In this model an attacker has list of public keys $A_1, \ldots, A_n$ and has access to a signing oracle for each public key. The goal of the attacker is to forge a signature on a message m for any one of the public keys A_i. The attacker wins if m was not queried to the sign oracle for this i-th key (but m could have been queried to some or all of the sign oracles for the other public keys).

This is a different situation to the one considered in Section 4.5 where we were interested in breaking all n users.

One would not expect a scheme to be completely insecure in the multi-user model, since an attack in the multi-user case can be turned into an attack in the single-user case by generating $n-1$ keys and hoping that the forgery is for the desired key (which will happen with probability $1/n$). Hence one would only expect the multi-user case to be weaker by at most a factor $1/n$.

The original paper [18] does not discuss multi-user security of EdDSA, but this is handled by Bernstein [13].

There has been a lot of discussion about multi-user security of signature schemes. The basic idea (going back to a paper of Galbraith, Malone-Lee and Smart [42]) is to transform an attacker in the multi-user setting into an attack in the single-user setting, by taking a single public key A and sign oracle, and creating n random public keys $A_i = A + [r_i]B$ and n signing oracles (each making a query to the original sign oracle for the key A). For the proof one has to show that a forgery for any of these new public keys can be converted into a valid forgery against the original public key A (and the analysis in [42] was incorrect as it did not consider queries to different sign oracles on the same message m). This allows us to deduce that the factor $1/n$ does not appear, and so it is no easier to forge in the multi-user case than the single-user case. The definitive analysis has been given by Bernstein [13].

Before going into Bernstein's analysis, I make one comment about the key clamping in EdDSA. As discussed in Section 5.3 of [13], if A is a well-formed EdDSA public key and if the r_i are chosen uniformly in $\mathbb{Z}_\ell$ then the random keys $A_i = A + [r_i]B$ are not necessarily correctly formed. This causes an unfortunate security loss in the reduction, by a factor of $(2^{251}/\ell)^n \approx 1/2^n$. This issue could have been easily avoided by using uniformly sampled private keys instead of using key clamping. Section 5.3 of [13] briefly mentions choosing the r_i from a narrower range, but this complicates the analysis.

Putting aside the problem of whether or not the keys A_i are well-formed, the fundamental problem with EdDSA is that the hash value in a signature includes the public key A_i. This makes it hard to relate a signature with respect to key A_i (and hence involving $H(\underline{R}, \underline{A_i}, \mathsf{m})$) to a signature with respect to key A (and hence involving $H(\underline{R}, \underline{A}, \mathsf{m})$). The proof in [13] avoids this issue by reducing to "plain" Schnorr signatures that do not include the public key in the hash. In other words, the security reduction starts with a public key A and a sign oracle for single-user Schnorr signatures where the verification equation involves computing

$$R = [S]B - H(\underline{R}, \mathsf{m})A.$$

The reduction then generates the random keys $A_1, \ldots, A_n$ and provides sign oracles $\mathcal{O}_i$ that on input m produce a signature where the verification equation is of the form

$$R = [S]B - H(\underline{R}, \underline{A_i}, \mathsf{m})A_i.$$

The reduction implements the signing oracle for key A_i by calling the original signing oracle on the message $\underline{A_i}\|\mathsf{m}$. Similarly, the final existential forgery of a message m^* for the key A_i becomes an existential forgery for the single-user scheme for key A and messsage $\underline{A_i}\|\mathsf{m}^*$.

In short, we do not have a reduction from the multi-user security of EdDSA to the single-user security of EdDSA. Instead, Bernstein's proof in [13] proves the multi-user security of EdDSA by building on previous work on the security of Schnorr signatures in the single-user case (and without key prefixing). This is of course a valid argument, and would show that there is no $1/n$ security loss in the multi-user case if it wasn't for the inconvenient loss of approximately $1/2^n$ due to the key clamping in EdDSA.

I end with the remark that no actual attack is known in the multi-user setting. Apart from side-channel attacks, the only known attacks on EdDSA or any other signature of this type involve solving the ECDLP. When presented with n ECDLP instances $A_1, \ldots, A_n$ there is no known algorithm that runs in fewer than $\sqrt{\ell}$ operations, and outputs a solutions to one of the instances. Hence the theoretical analysis supports our heuristic idea about attacks on the schemes: there is no loss of security in the multi-user setting.

7.8 Tightness and interpretations

We have touched upon tightness already. Suppose we have an EUF-CMA attacker that runs in time t, makes Q_H queries to the random oracle (of appropriate form) and succeeds with probability ϵ. Then one would like to conclude that there is an algorithm to solve ECDLP that runs in time $t' \approx t$ and succeeds with probability $\epsilon' \approx \epsilon$. If this were so then we could choose typical group sizes for security. However, such a tight reduction is completely out of reach.

The Bellare and Neven [8] version of the forking lemma results in $\epsilon' \approx \epsilon^2/Q_H$. The approach of Bellare and Dai [9] results in $\epsilon' \approx \epsilon/Q_H$, but this is to a different problem than ECDLP.

In practice, we tend to ignore such results when choosing parameters, which is why EdDSA using Curve25519 assumes a 253-bit group order provides 128-bit security. This is because there are no known attacks that exploit the tightness gap. In other words, the only known attack on EdDSA signatures (excluding side channels and fault attacks) is to solve an instance of the ECDLP.

7.9 Attacks base on biased ephemeral values, and side channel attacks

It has been long known that the ephemeral values r in $R = [r]P$ must be random and unpredictable, or else an attacker can try to learn the private key from $S = r + ca \pmod{\ell}$ where c is the public hash value that can be deduced from the signature. Further, it is known (going back at least to Howgrave-Graham and Smart [49] in 2001 and Nguyen and Shparlinsi [65] in 2002, and developed in a long line of research that culiminates in the very recent paper by Aranha et al [3]) that the ephemeral value r should be very close to uniformly distributed modulo ℓ. There is a history of attacks on elliptic curve signatures based on such failures of randomness.

It was also noted that side-channel attacks can lead to attacks of this type, especially timing attacks that reveal the length of the ephemeral value r. An important paper reporting on attacks of this type is Brumley and Tuveri [30].

We briefly list some well-known attacks of this type:

1. The Sony Playstation 3 hack was caused by an extremely weak implementation of the ECDSA signature that used a fixed choice of r. From two valid signature the hackers could cancel r and solve for the private key. This enabled them to forge signatures for any message, and hence install software of their choosing on any Playstation. For details see `https://www.schneier.com/blog/archives/2011/01/sony_ps3_securi.html` for example.
2. The Minerva [50] attack exploited timing side channels to recover partial information about ephemeral values in ECDSA (essentially the timing profile shows if the most-significant bits in the ephemeral value are zero). The attack is completed using lattice attacks.
3. Breitner and Heninger [28] have surveyed ECDSA signatures used in blockchain applications and documented cases where weak randomness seems to have led to the theft of cryptocurrency.
4. The TPM-Fail attack by Moghimi, Sunar, Eisenbarth and Heninger [62] uses these ideas to extract secret keys from a trusted hardware (Trusted Platform Module).

Recall that the ephemeral value in EdDSA is computed deterministically in response to the message, by computing the $2b$-bit integer $r = H(h_b, h_{b+1}, \ldots, h_{2b-1}, \mathsf{m})$ and reducing modulo ℓ. This means that the signing algorithm does not need access to a reliable stream of random bits. Assuming there is no bias in H (and no way to choose messages m to impose a bias on hash values modulo ℓ), then the problem is avoided.

There are two conflicting arguments about whether deterministic signatures are more or less secure against such attacks. On the positive side, the claim is that deterministic signatures do not require generating pseudorandom bits, and hence are simpler to implement (less risk of implementation errors) and less prone to weak randomness. By this argument, EdDSA should be resistant to attacks based on biased ephemeral values. A good discssion of this issue is given in the blog post "Why EdDSA held up better than ECDSA against Minerva" by Daniel Bernstein [20].

The counter-argument is that side-channel attacks are hard to mount in practice due to noise in the system, which causes difficulties when making accurate measurements (e.g., of timing or power traces). But with deterministic signatures one can repeat the attack multiple times for the same message in the knowledge that the computation done by the victim is identical in each execution. This allows to form a more accurate estimation of the side channel and helps to perform the attack. It is also worth noting that deterministic ephemeral values can still be small, and this could be leaked by a poor implementation and exploited as in the Minerva [50] attack.

My view is that there are two independent issues here: (1) Difficulty of obtaining good pseudorandom sequences (which makes schemes fragile to implementation errors); (2) Vulnerabilities due to side channels. Deterministic signatures are a good step to solving the first issue, while side-channel resistant (e.g., constant running-time) implementations are required for the latter issue. So a well-written constant-time implementation of EdDSA should provide a high level of resistance to all such attacks.

7.10 Message space entropy

One side-effect of the deterministic ephemeral value generation process in EdDSA is that the scheme seems to be reliant on the entropy of the message space. The original EdDSA paper [18] says "standard PRF hypotheses imply that this pseudorandom session key r is indistinguishable from a truly random string generated independently for each M, so there is no loss of security". What the paper is saying is that the secret key $H(k)$ is hidden to an attacker and the function $F : \mathsf{m} \mapsto r = H(h_b, h_{b+1}, \ldots, h_{2b-1}, \mathsf{m}) \pmod{\ell}$ is a pseudorandom function (PRF). The security model for a PRF is exactly that one cannot distinguish the values $F(\mathsf{m})$ from uniformly chosen values, so the claim in [18] is correct. The security comes from the fact that the secret key (in this context acting as the PRF key) is unknown and comes from a high entropy set. When studying PRFs it is usual to assume there is a large set of inputs, otherwise the problem is not interesting. Most formulations of security consider F as a keyed function from $\{0,1\}^n$ to $\{0,1\}^n$, or $\{0,1\}^*$ to $\{0,1\}^*$. For EdDSA it would be more natural to consider F as a function from some distribution on messages into $\mathbb{Z}/\ell\mathbb{Z}$.

One might wonder what are the security implications if using EdDSA with a message space of small entropy. Of course, if the number of messages is tiny then (since signing is deterministic) an attacker with access to a Sign oracle could compute a complete list of signatures on all messages. This does not actually help to create a forgery, according to the EUF security definition.

More seriously, one could worry that low message entropy could lead to a bias or other weakness in the ephemeral values. However, there is no security issue in this context, since the ephemeral values are computed as

$$r = H(h_b, h_{b+1}, \ldots, h_{2b-1}, \mathsf{m})$$

and the secret key bits coming from $H(k)$, though fixed, are unknown and from a high entropy source. If the number of possible messages is very small (e.g., less than $\sqrt{\ell}$) then one would expect all resulting values r to be distinct modulo ℓ, which means there is no chance of cancelling out r even if one had a list of all signatures. Also, the values r will behave like samples from the uniform distribution, so there should be no bias that can be exploited for a lattice attack.

In conclusion, the unpredictability of the ephemeral values comes from the fact that we have a b-bit secret key that is unknown to an attacker. The uniformity of the ephemeral values comes from the use of a good hash function and an unknown secret key. The entropy in the ephemeral values (which is not actually a requirement for security) comes from the message distribution.

7.11 Other side channel and fault attacks

We have already discussed side-channel attacks based on biases in ephemeral values. There are also more traditional side-channel and fault attacks that target point multiplication of various sorts. We do not attempt a thorough overview of the literature on this problem. It suffices to comment that, as with all mathematical cryptography, constant time implementation is critical in settings where side channel attacks can be mounted. Futher, protection against fault attacks should be done whenever cryptography is being used in an environment that exposes it to the risk of fault attacks. Some general references include [73, 7, 66].

8 Historical attacks

In this section we survey some old attacks on DSA and ECDSA and explain why EdDSA avoids them all.

8.1 Vaudenay attack on DSA

Suppose a signature scheme is being used where the hash of a message alone is used in the signature in a linear equation modulo the order ℓ of the group elements. Vaudenay [79] considered the case where the hash output size is larger than ℓ. A malicious user could choose two messages m_1 and m_2 and see whether there is a suitable sized prime ℓ such that $\ell \mid (H(m_1) - H(m_2))$. If so, they could construct a finite field $\mathbb{F}_p$ so that $\ell \mid (p-1)$ and then a signature on m_1 is also a signature on m_2.

This attack is not able to be mounted when the group order ℓ is fixed for all users and the choice of ℓ is done in a way that users can be confident they are not being tricked by the entity who generated the key (this is sometimes called "nothing up my sleeve" key generation).

EdDSA has further protection against such an attack, by including the public key A and the ephemeral point R in the hash of the message. This means it is impossible to anticipate the hash inputs before the parameters of the system have been fixed.

Vaudenay also considers attacks that work by changing the base point. A trivial attack is to set the base point B and the public key A to the identity element. This is not very interesting, but a more subtle attack on ECDSA is as follows: Suppose Alice (with public key A) has signed a message m to get signature (r, s). By the ECDSA verification equation we compute

$$X = [s^{-1}H(m)]B + [s^{-1}r]A$$

and then check that X encodes to r. An attacker could choose a random integer c and change the base point and public key to $B' = [c]B$ and $A' = [c]A$ and claim this as their own public key. Then (r, cs) is a valid signature on the message m for the new key.

In EdDSA this attack is prevented by including R and A in the hash.

8.2 Duplicate key attack on DSA

This attack is mentioned in Section 8.3 of Johnson, Menezes, Vanstone [51]. Let (r, s) be an ECDSA signature on message m for public key (B, A). Then

$$X = [s^{-1}H(m)]B + [s^{-1}r]A$$

is such that X corresponds to r.

The idea of the attack is to compute a new public key (B', A') such that (r, s) is still a valid signature on the message m, and such that the attacker can demonstrate knowledge of the discrete logarithm of A' to the base B'.

To do this choose a random integer c and compute $t = s^{-1}H(m) + s^{-1}rc \pmod{\ell}$ until $t \neq 0$. It follows that t is invertible modulo ℓ, so let

$$B' = [t^{-1} \pmod{\ell}]X \quad \text{and} \quad A' = [c]B'.$$

Then

$$\begin{aligned}[s^{-1}H(m)]B' + [s^{-1}r]A' &= [(s^{-1}H(m) + s^{-1}rc) \pmod{\ell}]B' \\ &= [t][t^{-1} \pmod{\ell}]X \\ &= X\end{aligned}$$

so (r, s) is a valid signature on the message m.

The attack does not apply to EdDSA for several reasons. First, the base point B is fixed and cannot be set to a chosen value. Second, the public key is included in the hash function, so that one does not get the same hash value when signing the same message with two different public keys.

8.3 Bleichenbacher attack on Elgamal signature

In 1996 Bleichenbacher [22] showed an attack on Elgamal signatures implemented in the multiplicative group $\mathbb{F}_p^*$ of a finite field. The attack depended on a large smooth factor of $p - 1$ and certain algebraic relations. The attack does not generalise to elliptic curves with prime or nearly prime order. It has no impact on the security of EdDSA.

9 Quantum algorithms

Since we are studying a system based on the discrete logarithm problem, it is natural to be worried about the progress in quantum computing and Shor's algorithm. Currently there is no quantum computer known that can run Shor's

algorithm on crypto-sized elliptic curves. In this section I will briefly summarise progress in this area, but please note that I am not an expert in quantum computing.

The main conclusion of the analysis in this section is that progress in quantum computing is relatively slow, and that there seems to be no immediate threat to ECDLP. It is also worth noting that the use of digital signatures in many applications requires that no-one can forge a signature *today*. This is in contrast to encryption security which often requires ciphertexts to be hard to break even in 20 or 30 years into the future. Hence I believe it is safe to use EdDSA signatures for many applications for the next 5-10 years at least. One exception would be applications such as code-signing, for example for software updates, since devices manufactured today may be "in the field" for many years. It might be dangerous to rely on discrete logarithm-based signatures in such settings, since a future attacker with a quantum computer may be able to install malicious updates onto such a device.

At a high level, quantum computers are based on the notion of a qubit. It is important to distinguish *physical qubits*, which are related to the implementation of the machine, from *logical qubits* which are an abstraction that can be used when running a quantum algorithm. It is necessary to use error correction and certain physical structures to build a stable fault-tolerant logical qubit from many physical qubits. The exact overhead of error control is still an active area of research.

Researchers have been estimating the number of logical qubits needed to solve instances of the ECDLP using Shor's algorithm on a fault-tolerant quantum computer. I now review some notable papers on this topic. These papers focus on the number of Toffoli gates required to perform the algorithm. The Toffoli gate is also known as the controlled NOT gate, and is a fairly simple universal reversible logic gate which is commonly used by theoretical researchers to estimate the cost of a quantum computation.

1. Roetteler, Naehrig, Svore and Lauter [70] considered quantum circuits for elliptic curve exponentiation as required to run Shor's algorithm. Their analysis for elliptic curves over prime fields of size around 256-bits are that the implementation would require 2330 logical qubits, $1.26 * 10^{11}$ Toffoli gates and $1.16 * 10^{11}$ Toffoli depth. These figures are reproduced in Table 9.2 of the extensive report [32] by the German Federal Office for Information Security (BSI).

   A comment on the meaning of these numbers: It seems to be usual in the quantum algorithms community to consider circuits of gates, which is how these numbers are obtained, but this is not actually how such algorithms would be implemented. Another way to express the calculation is that Shor's algorithm for 256-bit ECDLP instances can be implemented to perform a total of $1.26 * 10^{11}$ Toffoli operations on qubits. The Toffoli depth is an upper bound on the number of Toffoli operations involving any single qubit.
2. In very recent work, Häner, Jaques, Naehrig, Roetteler and Soeken [46] have revisited these calculations. As well as giving some improvements to the circuits, they consider different trade-offs that are appropriate for different measures of cost. For elliptic curves over 256-bit prime fields, they can reduce the number of logical qubits from 2338 to 2124. Alternatively, they can reduce the number of Toffoli gates to $1.08 * 2^{31} \approx 2.3 * 10^9$ gates. Alternatively, they can reduce the Toffoli depth to $1.12 * 2^{24} \approx 1.9 * 10^7$ (but this now needs 2871 qubits).

Again, we stress that the above articles discuss logical qubits. Gidney and Ekerå [45], in the context of 2048-bit RSA, present results for both logical and physical qubits. Their calculations suggest one can factor in around 6000 logical qubits, and that this can be implemented with around $2*10^7$ physical qubits (in other words, over 3000 physical qubits for each logical qubit). Extrapolating such results to ECDLP (which may not be appropriate) would suggest that solving 256-bit instances we would need over 6 million physical qubits.

We now briefly review progress in quantum computing. The concept of quantum computing has been around since the 1980s, and the initial experiments with qubits and quantum computing took place in the late 1990s. The first implementation of Shor's algorithm (which produced the factorisation $15 = 3 \times 5$) took place in 2001. In 2019, after 20 years of progress, IBM produced a 53 (physical) qubit machine.

The main technical challenges seem to be to scale the number of qubits and to control errors. IBM has a roadmap [39] to 1000 qubit hardware in 3 years and hopes to reach 10^6 qubits a little later, but they have not set themselves any error rate targets. Note that these are counts of physical qubits rather than logical qubits. More precisely, IBM plans to develop the 127-qubit IBM Quantum Eagle in 2021 which "will allow us to implement the heavy-hexagonal error-correcting code that our team debuted last year, so as we scale up the number of physical qubits, we will also be able to explore how they'll work together as error-corrected logical qubits – every processor we design has fault tolerance

considerations taken into account." The plan for 2022 includes a 433-qubit IBM Quantum Osprey system and for 2023 the 1,121-qubit IBM Quantum Condor processor. Their announcement says "We think of Condor as an inflection point, a milestone that marks our ability to implement error correction and scale up our devices."

A survey has been conducted by Mosca and Piani for the Global Risk Institute [63]. In particular, they have surveyed a wide range of experts to determine estimates for when quantum computers will be a "significant threat" to public key crypto. The majority of experts believe the risk is low (less than 5 percent) for the next 5-10 years, though this still does not exclude the possibility of a breakthrough. Looking ahead 15 years, about half the responders consider the risk to be at least 50 percent. By this measure, a conservative recommendation would be to migrate to post-quantum crypto within the next 10 years.

As already mentioned, the German Federal Office for Information Security (BSI) commissioned a report [32] in 2018 on the current state of quantum computers. The report concluded "At this point in time, quantum processors that have been realized are far from those needed to attack cryptography". The report was updated in 2020, but the conclusions remain the same. In a lecture at the PQCrypto conference in late 2020, Frank Wilhelm-Mauch (one of the authors of the report) speculated that quantum computers will stay at the 50-100 qubit range for the near future, while researchers focus on controlling the errors.

Similarly conservative views have been made by other agencies and organisations. In 2018 the US National Academies of Sciences, Engineering, and Medicine released a report [64] that says "a quantum computer that could compromise todays cryptography is likely at least a decade away". Reports by Ericsson [38] and Thales [77] both say that quantum computers that can break current public key systems like EdDSA are "decades away". There are even theoretical objections to quantum computers in principle, though these do not seem to be widely believed.

As already mentioned, if one has a system that requires data to remain private for a long time then one should consider migrating to post-quantum encryption schemes. However, for many (but not all) applications of signatures it seems acceptable to continue using elliptic curve public key signatures until such time as there is a realistic threat that the ECDLP can be solved using a quantum computer. In conclusion, while progress on quantum computing should be closely watched, I see no reason why EdDSA should not be used for the coming 10 years at least.

10 Conclusions, suggestions and open problems

My conclusions, suggestions and opinions:

1. EdDSA is a good design for a signature scheme, except perhaps for the key clamping, which seems to cause more difficulties than it provides benefits. Why not abandon the key clamping in EdDSA? One could either replace private keys with uniformly sampled integers in $\mathbb{Z}_\ell$, or else use a much larger hash output as done with the ephemeral values r.
2. Curve 25519 provides a high level of security for the next 10-20 years, and 448-bit keys (such as in Ed448) are overly conservative and not needed.
3. Deterministic signatures solve some of the security problems of discrete log signatures, but constant time implementation is still critical in some settings.
4. The tighter security reduction by Bellare and Dai is interesting, but the MBDL assumption deserves further study.
5. I do not expect any major advance in classical algorithms for ECDLP in the next 5-10 years (or longer).
6. I doubt that quantum computers capable of solving 256-bit ECDLP instances can be built within the next 10 years.

References

1. ANSI, Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA), ANS X9.62 (2005)
2. A. Antipa, D. R. L. Brown, R. P. Gallant, R. J. Lambert, R. Struik, and S. A. Vanstone, Accelerated verification of ECDSA signatures, in B. Preneel and S. E. Tavares (eds.), SAC 2005 Springer LNCS 3897 (2006) 307–318.
3. Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi and Yuval Yarom, LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage, IACR eprint 2020/615 (2020)
4. R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen and F. Vercauteren, Handbook of elliptic and hyperelliptic cryptography, Chapman and Hall/CRC (2006)

5. D.V. Bailey, L. Batina, D.J. Bernstein, P. Birkner, J.W. Bos, H.C. Chen, C.M. Cheng, G. van Damme, G. de Meulenaer, L.J.D. Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A.V. Herrewege and B.Y. Yang, Breaking ECC2K-130, IACR ePrint 2009/541 (2009)
6. Razvan Barbulescu, Pierrick Gaudry, Antoine Joux and Emmanuel Thomé, A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic, in P. Q. Nguyen and E. Oswald (eds.), EUROCRYPT 2014, Springer LNCS 8441 (2014) 1–16.
7. Alessandro Barenghi and Gerardo Pelosi, A Note on Fault Attacks Against Deterministic Signature Schemes (Short Paper), IWSEC 2016, Springer (2016) 182–192.
8. M. Bellare and G. Neven, *Multi-signatures in the plain public-key model and a general forking lemma*, in A. Juels, R. N. Wright, and S. De Capitani di Vimercati (eds.) CCS 2006, ACM (2006) 390–399.
9. Mihir Bellare and Wei Dai, The Multi-Base Discrete Logarithm Problem: Concrete Security Improvements for Schnorr Identification, Signatures and Multi-Signatures, IACR ePrint 2020/416 (2020)
10. David Bernhard, Olivier Pereira and Bogdan Warinschi, How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios, IACR eprint 2016/771 (2016)
11. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, *Twisted Edwards curves*, in S. Vaudenay (ed.) Africacrypt 2008, Springer LNCS 5023, (2008) 389–405.
12. Daniel J. Bernstein, Curve25519: New Diffie-Hellman Speed Records. in M. Yung, Y. Dodis, A. Kiayias and T. Malkin (eds.), PKC 2006, Springer LNCS 3958 (2006) 207–228.
13. Daniel J. Bernstein, Multi-user Schnorr security, revisited, IACR ePrint 2015/996 (2015)
14. Daniel J. Bernstein and Tanja Lange, Faster Addition and Doubling on Elliptic Curves, in K. Kurosawa (ed.), ASIACRYPT 2007, Springer LNCS 4833 (2007) 29–50.
15. D.J. Bernstein, T. Lange and P. Schwabe, On the correct use of the negation map in the Pollard rho method. in D. Catalano, N. Fazio, R. Gennaro, A. Nicolosi (eds.) PKC 2011, Springer LNCS 6571 (2011) 128–146.
16. D.J. Bernstein and T. Lange, Two grumpy giants and a baby, in E.W. Howe and K.S. Kedlaya (eds.), Proceedings of the Tenth Algorithmic Number Theory Symposium, MSP, Vol. 1 (2013) 87–111.
17. D.J. Bernstein and T. Lange, Non-uniform cracks in the concrete: The power of free precomputation, in K. Sako, P. Sarkar (eds.) ASIACRYPT 2013, Springer LNCS 8270 (2013) 321–340.
18. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe and Bo-Yin Yang, High-speed high-security signatures. J. Cryptogr. Eng. **2**, No. 2 (2012) 77–89.
19. Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe and Bo-Yin Yang, EdDSA for more curves, July 2015, IACR eprint 2015/677 (2015)
20. Daniel J. Bernstein, Why EdDSA held up better than ECDSA against Minerva, blog post (2019) `https://blog.cr.yp.to/20191024-eddsa.html`
21. I.F. Blake, G. Seroussi and N.P. Smart, Elliptic Curves in Cryptography, Cambridge (1999)
22. Daniel Bleichenbacher, Generating ElGamal signatures without knowing the secret key, EUROCRYPT96, Springer LNCS 1070 (1996) 10–18.
23. Dan Boneh and Phillip Rogaway, Security Level of Cryptography-ECDSA, CRYPTREC EX-1006-2001 (2001) `https://www.cryptrec.go.jp/exreport/cryptrec-ex-1006-2001.pdf`
24. Dan Boneh and Victor Shoup, A Graduate Course in Applied Cryptography, version 0.5 (2020) `http://toc.cryptobook.us/`
25. Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra and Peter L. Montgomery, On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography, IACR ePrint 2009/389 (2009)
26. Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra and Peter L. Montgomery, Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction, Int. J. Appl. Cryptogr. **2**(3) (2012) 212–228.
27. Joppe W. Bos, Thorsten Kleinjung and Arjen K. Lenstra, On the use of the negation map in the Pollard Rho method. in G. Hanrot, F. Morain, E. Thomé (eds.) ANTS IX, Springer LNCS 6197 (2010) 66–82.
28. Joachim Breitner and Nadia Heninger, Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies, in I. Goldberg and T. Moore (eds.), Financial Cryptography, Springer LNCS 11598 (2019) 3–20
29. Daniel R. L. Brown, Generic Groups, Collision Resistance, and ECDSA, Designs, Codes and Cryptography, vol 35 (2005) 119–152.
30. Billy Bob Brumley and Nicola Tuveri, Remote Timing Attacks Are Still Practical, in V. Atluri and C. Díaz, ESORICS 2011, Springer LNCS 6879 (2011) 355–371.
31. J. Brendel, C. Cremers, D. Jackson and M. Zhao, The provable security of Ed25519: Theory and practice, to appear at IEEE Symposium on Security and Privacy, S&P 2020.
32. Status of quantum computer development, Version 1.2, June 2020, Federal Office for Information Security, Bonn, Germany.
33. Konstantinos Chalkias, Franois Garillot and Valeria Nikolaenko, Taming the Many EdDSAs, Security Standardisation Research SSR 2020, Springer LNCS 12529 (2020) 67–90.

34. Yilei Chen, Alex Lombardi, Fermi Ma and Willy Quach, Does Fiat-Shamir Require a Cryptographic Hash Function? IACR ePrint 2020/915 (2020)
35. Jung Hee Cheon and Jeong Hyun Yi, Fast Batch Verification of Multiple Signatures, in PKC 2007, Springer LNCS 4450 (2007) 442–457.
36. Henry Corrigan-Gibbs and Dmitry Kogan, The Discrete-Logarithm Problem with Preprocessing, in J. B. Nielsen and V. Rijmen (eds.) EUROCRYPT 2018, Springer LNCS 10821 (2018) 415–447.
37. H. M. Edwards, *A normal form for elliptic curves*, Bulletin of the AMS **44** (2007) 393–422.
38. Ericsson, What next in the world of post-quantum cryptography? May 2020. https://www.ericsson.com/en/blog/2020/3/post-quantum-cryptography-symmetric-asymmetric-algorit
39. IBM, IBM roadmap to quantum computing, Sept 2020. https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/
40. P. Fouque, A. Joux and C. Mavromati, Multi-user collisions: Applications to discrete logarithm, Even-Mansour and PRINCE, in P. Sarkar, T. Iwata (eds.), ASIACRYPT 2014, Springer LNCS 8873 (2014) 420–438.
41. G. Frey and H.-G. Rück, A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves, Math. Comp., vol. 62 (1994) 865–874.
42. Steven D. Galbraith, John Malone-Lee and Nigel P. Smart, Public key signatures in the multi-user setting, Inf. Process. Lett., vol. 83, no. 5 (2002) 263–266.
43. Steven D. Galbraith, Mathematics of Public Key Cryptography, Cambridge, 2012.
44. Steven D. Galbraith, Ping Wang and Fangguo Zhang, Computing Elliptic Curve Discrete Logarithms with Improved Baby-step Giant-step Algorithm, Advances in Mathematics of Communications (AMC), vol. 11, no. 3 (2017) 453–469.
45. Craig Gidney and Martin Ekerå, How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, arXiv:1905.09749 (2019)
46. Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler and Mathias Soeken, Improved Quantum Circuits for Elliptic Curve Discrete Logarithms, in J. Ding and J.-P. Tillich (eds.), PQCrypto 2020, Springer LNCS 12100 (2020) 425–444.
47. D. Hankerson, A. Menezes and S. Vanstone, Guide to elliptic curve cryptography, Springer, 2004.
48. Y. Hitchcock, P. Montague, G. Carter and E. Dawson, The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves, Int. J. Inf. Secur., vol. 3 (2004) 86–98.
49. Nick Howgrave-Graham and Nigel P. Smart, Lattice Attacks on Digital Signature Schemes, Des. Codes Cryptogr., vol. 23, no. 3 (2001) 283–290.
50. Jan Jancar, Vladimir Sedlacek, Petr Svenda and Marek Sys, Minerva: The curse of ECDSA nonces (Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces), CHES 2020, IACR Trans. Cryptographic Hardware and Embedded Systems, vol. 4 (2020) 281–308.
51. Don Johnson, Alfred Menezes and Scott Vanstone, The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security, vol. 1 (2001) 36–63.
52. S. Josefsson and I. Liusvaara, RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA), Internet Research Task Force (IRTF) (2017) https://tools.ietf.org/html/rfc8032
53. S. Karati, A. Das, D. Roychowdhury, B. Bellur, D. Bhattacharya and A. Iyer, Batch verification of ECDSA signatures, in A. Mitrokotsa and S. Vaudenay (eds.), AFRICACRYPT 2012, Springer LNCS 7374 (2012) 1–18.
54. N. Koblitz and A. Menezes, Another look at non-uniformity, Groups Complexity Cryptology, vol. 5 (2013) 117–139.
55. F. Kuhn and R. Struik, Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms, in S. Vaudenay and A.M. Youssef (eds.), SAC 2001, Springer LNCS 2259 (2001) 212–229.
56. A. Langley, M. Hamburg and S. Turner, RFC: 7748: Elliptic Curves for Security, Internet Research Task Force (IRTF), January 2016. https://tools.ietf.org/html/rfc7748
57. Hyung Tae Lee, Jung Hee Cheon and Jin Hong, Accelerating ID-based Encryption based on Trapdoor DL using Precomputation, ePrint 2011/187 (2011)
58. Arjen K. Lenstra and Eric R. Verheul, Selecting Cryptographic Key Sizes, Journal of Cryptology, vol. 14, no. 4 (2001) 255–293.
59. M. Lochter and J. Merkle, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, IETF RFC5639, March 2010. https://tools.ietf.org/html/rfc5639
60. A. Menezes, T. Okamoto and S. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Transactions on Information Theory, vol. 39 (1993) 1639–1646.
61. Joseph P. Mihalcik, An analysis of algorithms for solving discrete logarithms in fixed groups, Master thesis, Naval Postgraduate School, March 2010.
62. Daniel Moghimi, Berk Sunar, Thomas Eisenbarth and Nadia Heninger, TPM-FAIL: TPM meets Timing and Lattice Attacks, in S. Capkun and F. Roesner (eds.), USENIX Security (2020) 2057–2073.
63. Michele Mosca and Marco Piani, Quantum Threat Timeline, Global Risk Institute, October 2019. https://globalriskinstitute.org/publications/quantum-threat-timeline/

64. US National Academies of Sciences, Engineering, and Medicine, New Cryptography Must Be Developed and Deployed Now, Even Though A Quantum Computer That Could Compromise Todays Cryptography Is Likely At Least A Decade Away. December 4, 2018.
65. Phong Nguyen and Igor Shparlinski, The Insecurity of the Digital Signature Algorithm with Partially Known Nonces, Journal of Cryptology, vol. 15 (2002) 151–176.
66. Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter and Paul Rsler, Attacking Deterministic Signature Schemes using Fault Attacks, 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London (2018) 338–352.
67. David Pointcheval and Jacques Stern, Security Arguments for Digital Signatures and Blind Signatures, J. Cryptol., vol. 13, no. 3 (2000) 361–396.
68. J.M. Pollard, Kangaroos, Monopoly and discrete logarithms, J. Cryptology, vol. 13, no. 4 (2000) 437–447.
69. Jean-Luc Pons and Aleksander Zieniewicz, Pollard's kangaroo for SECPK1, 2020. https://github.com/JeanLucPons/Kangaroo
70. Martin Roetteler, Michael Naehrig, Krysta M. Svore and Kristin E. Lauter, Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms, in T. Takagi and T. Peyrin (eds.), ASIACRYPT 2017, Springer LNCS 10625 (2017) 241–270.
71. H.-G. Rück, On the discrete logarithm in the divisor class group of curves, Math. Comp., vol. 68 (1999) 805–806.
72. T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, Commentarii Mathematici Universitatis Sancti Pauli, vol. 47 (1998) 81–92.
73. J. Schmidt and M. Medwed, A Fault Attack on ECDSA, 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC) (2009) 93–99.
74. I. Semaev, Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p, Math. Comp., vol. 67 (1998) 353–356.
75. N.P. Smart, The discrete logarithm problem on elliptic curves of trace one, Journal of Cryptology, vol. 12 (1999) 193–196.
76. Jacques Stern, Evaluation Report on the ECDSA signature scheme, CRYPTREC EX-1004-2001 (2001) https://www.cryptrec.go.jp/exreport/cryptrec-ex-1004-2001.pdf
77. Thales, Cryptography for a post-quantum era, November 2018.
78. Henry de Valence, It's 255:19AM. Do you know what your validation criteria are?, (2020) https://hdevalence.ca/blog/2020-10-04-its-25519am
79. Serge Vaudenay, Hidden Collisions on DSS, CRYPTO 1996, Springer LNCS 1109 (1996) 83–88.
80. P. van Oorschot and M.J. Wiener, Parallel collision search with cryptanalytic applications, J. Cryptology, vol. 12, no. 1 (1999) 1–28.
81. L.C. Washington, Elliptic Curves: Number Theory and Cryptography, 2nd ed., CRC Press, 2008.
82. Michael J. Wiener, The Full Cost of Cryptanalytic Attacks, J. Cryptology, **17** No. 2, 105–124.