Failures in NIST's ECC standards

Daniel J. Bernstein[1,2] and Tanja Lange[1]

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org

[2] Department of Computer Science, University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

Abstract. NIST's ECC standards create (1) unnecessary losses of simplicity, security, and speed in ECC implementations and (2) unnecessary tensions between simplicity, security, and speed in ECC implementations.

1 Introduction

> *The poor user is given enough rope with which to hang himself—something a standard should not do.* —Rivest, 1992 [50], commenting on the NIST/NSA "DSA" proposal

NIST's standards for elliptic-curve cryptography (ECC) consist of

- NSA's choices of *primes*, such as the "P-256 prime" $2^{256}-2^{224}+2^{192}+2^{96}-1$;
- NSA's choices of *curves* modulo those primes, such as "NIST P-256", the curve $y^2 = x^3 - 3x + 41058363725152142129326129780047268409114441015993725554835256314039467401291$ modulo the P-256 prime;
- NSA's choices of *coordinates* for transmitting points on those curves, such as "uncompressed short Weierstrass coordinates" (x, y) for NIST P-256;
- NSA's choices of *computations* to be used inside implementations, such as "addition-subtraction methods" for NIST P-256; and
- NSA's choices of *protocols* using these curves, such as the "ECDSA" signature algorithm.

NIST's FIPS 186-4 specifies the choices of primes and curves, and refers to ANSI X9.62 for the choices of coordinates, computations, and protocols. FIPS 186-4

This work was supported by the European Commission under Contract ICT-645421 ECRYPT-CSA; by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005; and by the U.S. National Science Foundation under grant 1018836. "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation." Permanent ID of this document: 1c4db65fd2bf7a58d36655b8e8690a541cd8ecab. Date: 2016.01.06. This is a revision (extra references, clarifications, etc.) of comments sent to NIST on 2015.12.04.

is a "Digital Signature Standard" but the curves are also used in non-signature protocols, notably ECDH.

1.1. Potential problems with the NIST ECC standards. News reports in September 2013 [44] revealed that NSA had a $250 million/year program designed to "make [systems] exploitable through SIGINT collection"; in particular, to "insert vulnerabilities" into systems and "influence policies, standards and specification for commercial public key technologies". The reports indicated, as an example, that NIST's standard "Dual EC" elliptic-curve random-number generator was backdoored by NSA. This prompted extensive public discussion of whether vulnerabilities could have been somehow inserted not merely into NIST's standards for elliptic-curve *random-number generation* but also into NIST's much more widely used standards for elliptic-curve *cryptography*.

The Dual EC vulnerability is exploitable only by attackers in possession of a secret back-door key. The news reports make reasonably clear

- that NSA possesses this key and
- that NSA does not have a stellar track record of keeping secrets;

it is therefore easy to imagine many different attackers having copies of the key. We have seen some commentators speculating that NSA inserts vulnerabilities into systems *only* if those vulnerabilities have secret back-door keys, but we have seen no evidence to support this speculation.

The *risk* of an intentional vulnerability is particularly difficult to assess. One hopes that the extensive public study of ECC has been sufficiently comprehensive, and that all curves passing public security criteria are in fact secure; if this is true then the choice of a curve cannot create a vulnerability. However, it is possible that this hope is incorrect, and that the curve generator is aware of security variations among these curves. A curve generator with the flexibility to choose from among a pool of N curves can turn a curve vulnerability with probability ϵ into a curve vulnerability with probability $1-(1-\epsilon)^N \approx N\epsilon$ (assuming statistical independence). It is entirely possible that $N\epsilon$ is large even if ϵ is small.

Even worse, N is surprisingly high, as shown in our paper "How to manipulate curve standards: a white paper for the black hat" [12] with Chou, Chuengsatiansup, Hülsing, Lambooij, Niederhagen, and van Vredendaal. The examples in that paper[3] illustrate that claims of protection against backdoors need to be carefully studied, just like other security claims. It would be particularly embarrassing if, e.g., NIST adopts some half-baked idea for using "verifiably random" Bitcoins to generate curves, and a subsequent study then shows that manipulating those Bitcoins would have been well within the budget of today's attackers.

1.2. Definite problems with the NIST ECC standards. We are concerned that attention to the *possibility* of back doors is distracting NIST from what is *definitely* going wrong with NIST's ECC standards. Most importantly, NIST's ECC standards create unnecessary complexity in ECC implementations. This unnecessary complexity is important because it

[3] We incorporate that paper by reference into our comments to NIST.

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

It is theoretically *possible* to implement NIST's ECC standards in a secure way, but it is unnecessarily *difficult*—even in situations where there are no performance constraints. Even worse, the path to a secure implementation is littered with unnecessary traps: ways that simpler implementations seem to work but are not in fact secure. Implementors naturally try to reduce complexity, and end up falling into these traps, compromising security.

Are we saying that cryptographers should always apply every imaginable simplification? Of course not. For example:

- ECB is simpler than GCM. Should GCM users switch to ECB? No: that would be an oversimplification. The problem here is that ECB doesn't authenticate and doesn't securely encrypt.
- Multiplicative groups are simpler than elliptic-curve groups. Should ECC users switch to multiplicative groups? No: that would be an oversimplification. The problem here is that multiplicative groups are vulnerable to index calculus. This produces bigger keys and slower computations; this also makes the security analysis more difficult and less stable, reducing confidence.

As these examples illustrate, the cryptographer's top priority is security, and the cryptographer's second priority is to meet the user's performance requirements. Simplicity is only the third priority.

Sometimes, starting from examples of oversimplification damaging security or speed, people say "Simplicity damages security" or "Simplicity damages speed" or "Simplicity in cryptography is bad". These are wild overgeneralizations, often used to cover up deficient analyses of speed and security. Many simplifications don't hurt security at all and don't hurt speed at all. In fact, the simplicity of next-generation ECC *contributes to security* and *contributes to speed.*

The next five sections of this document analyze ways that simplicity, security, and speed are compromised by NSA's choices of primes, curves, coordinates, computations, and protocols. The remaining sections of this document answer various questions that were specifically asked by NIST.

2 Protocols

The following high-level description of ECDH is uncontroversial. There is a standard base point B on a standard elliptic curve E over a standard finite field. Alice generates a secret integer r and a corresponding public key rB. Bob generates a secret integer s and a corresponding public key sB. Alice computes rsB

as $r \cdot sB$, and Bob computes the same rsB as $s \cdot rB$. Some sort of hash of this shared secret rsB is used to encrypt and authenticate data.

For *signatures* there are many more choices to make at the same level of ECC protocol description. Some amount of extra complexity seems unavoidable,[4] but poor choices add unnecessary further complexity, along with damaging speed and security.

The rest of this section focuses on the details of two signature protocols: ECDSA, which is part of NIST's standards, and EdDSA, which we introduced in a paper [15] with Duif, Schwabe, and Yang and generalized in a paper [17] with Josefsson, Schwabe, and Yang.[5] See [10] for a step-by-step explanation of how to obtain EdDSA through modifications to the original ElGamal signature system.[6]

In EdDSA, the verifier checks the equation $SB = R + H(R, A, M)A$. Here B is a standard base point as above; H is a standard hash function; A is the signer's public key; R is a curve point included in the signature; S is an integer included in the signature; and M is the message being verified.

In ECDSA, the verifier checks the equation $H(M)B + x(R)A = SR$. This forces the signer to perform a division, damaging speed and simplicity. It also forces the verifier to perform a triple-scalar multiplication, or to instead check $(H(M)/S)B + (x(R)/S)A = R$, again damaging speed and simplicity.

An ECDSA signature (R, S) could be encoded, without any change of security, as (R, S') where $S' = S/H(M)$. The verifier would then check the equation $B + H'(R, M)A = S'R$, where $H'(R, M) = x(R)/H(M)$. This view shows that, from a security perspective, moving from ECDSA to EdDSA means

- putting S in front of B rather than R;
- replacing H' with a conventional hash function H; and
- including A as an extra hash input.

The first change was introduced by Schnorr and eliminates divisions. The second change eliminates the multiplicative structure of H', improving security as discussed in the next paragraph. The third change alleviates concerns that several public keys could be attacked simultaneously.

In ECDSA, an attacker who finds a collision in H also finds a collision in H', breaking the system. EdDSA is collision-resilient: hash-function collisions do not break the system.

Schnorr used collision resilience as justification for taking a hash function with smaller output, and then used this to save some space in signatures, replacing

[4] This complexity is created by the basic data flow in signatures, where a signature is sent through a one-way communication channel to a receiver. A receiver who has a two-way communication channel with the sender can skip signatures and use ECDH to achieve even better integrity protection, guaranteeing that the sender is vouching for the data *now* ("freshness") rather than at some indeterminate time in the past. ECDH also easily provides confidentiality protection, protection for communication in the opposite direction, etc.

[5] We incorporate those papers by reference into our comments to NIST.

[6] We incorporate that blog post by reference into our comments to NIST.

R with the H output. EdDSA instead uses collision resilience as an extra line of defense, uses R in signatures to allow fast batch verification, and takes a double-size H output.

In all of these systems, the signer generates R as rB, where r is a one-time secret scalar. EdDSA generates r by deterministically hashing a secret together with M, so randomness is not used after key generation. This makes the signing procedure much easier to test and audit. This idea was proposed by Barwood [6] and Wigley [52] in 1997, many years before poor generation of r revealed the Sony PlayStation 3 ECDSA signing key [26] and Bitcoin secret keys [22].

3 Computations of multiples of points

There is a long history of successful timing attacks against cryptographic software, and in particular ECC software. For example, the very recent paper [2] demonstrates successful recovery, through timing-based side channels, of a secret key used by OpenSSL's ECDSA implementation[7] for just 6 signatures.

Evidently constant-time ECC software is critical for ECC security. This has an impact on several layers of choices inside ECC standards.

There are other important side channels. We have chosen to highlight timing because timing attacks are particularly dangerous. Timing is visible through networks; timing is visible to untrusted code running on the same machine (for example, in browsers); timing is unaffected by typical physical-security mechanisms. Analysis of other side channels is more complicated but generally supports the same conclusions regarding the choice of ECC mechanisms: see, e.g., the hardware-oriented survey [32] by Fan, Guo, De Mulder, Schaumont, Preneel, and Verbauwhede.

3.1. The Montgomery ladder. Curve25519 [9] is the Montgomery curve $y^2 = x^3 + Ax^2 + x$ over $\mathbf{F}_p$, where $p = 2^{255} - 19$ and $A = 486662$. Each curve point Q has an x-coordinate $X_0(Q)$ defined as follows: if $Q = (x, y)$ then $X_0(Q) = x$; if Q is the point at infinity then $X_0(Q) = 0$. Here is a spectacularly simple method of single-scalar x-coordinate multiplication on Curve25519:

```
x2,z2,x3,z3 = 1,0,x1,1
for i in reversed(range(255)):
  bit = 1 & (n >> i)
  x2,x3 = cswap(x2,x3,bit)
  z2,z3 = cswap(z2,z3,bit)
  x3,z3 = ((x2*x3-z2*z3)^2,x1*(x2*z3-z2*x3)^2)
  x2,z2 = ((x2^2-z2^2)^2,4*x2*z2*(x2^2+A*x2*z2+z2^2))
  x2,x3 = cswap(x2,x3,bit)
  z2,z3 = cswap(z2,z3,bit)
return x2*z2^(p-2)
```

[7] The attack was demonstrated against Bitcoin signatures, which use a non-NIST curve, but the attack strategy should also work for other curves.

The inputs are an integer n $\in \{0, 1, 2, \ldots, 2^{255} - 1\}$ and the x-coordinate x1 $= X_0(Q)$ of a curve point Q. The output is exactly $X_0(\mathtt{n}Q)$. The operations +, *, etc. are constant-time arithmetic operations in $\mathbf{F}_p$. The cswap function performs a constant-time conditional swap.

This scalar-multiplication method, the "Montgomery ladder", works for *all* inputs, as shown in [9]. There is no need to check for any special cases. The proofs in [9] apply to any Montgomery curve with a unique point of order 2: any curve $By^2 = x^3 + Ax^2 + x$ over any prime field $\mathbf{F}_p$ where $p \ge 5$, $B \ne 0$, and $A^2 - 4$ is non-square. One can also change 255 to another number in the code, allowing that number of bits in the scalar n. With some extra lines of code one can also compute the y-coordinate of $\mathtt{n}Q$ given the y-coordinate of Q, but (as pointed out by Miller in [43]) this extra complexity is not necessary for ECDH.

X25519 is a widely deployed ECDH system[8] using Curve25519, with these x-coordinates encoded as 32-byte public keys in a straightforward way. Practically all X25519 implementations use the Montgomery ladder to compute shared secrets. The Montgomery ladder is extremely fast (and fits into small hardware), when the pieces of the computation are appropriately optimized. There are many other ways to compute X25519 shared secrets, but none of them are simpler or faster than the Montgomery ladder, so there is no incentive for X25519 implementations to use them.

Some parts of the literature, instead of presenting a constant-length ladder as above, present a variable-length ladder, starting from the top bit that is *set* in the scalar. Brumley and Tuveri in [25] demonstrated remote extraction of secret keys through timings of OpenSSL's binary-field ECDSA implementation; this implementation used a ladder for multiplication by the one-time scalar, and thus leaked the fact that some scalars were particularly small, allowing the attack to recover the long-term secret key by lattice techniques. Small information leaks regarding ECDH keys do not seem to be as damaging, but we recommend a policy of systematically eliminating *all* timing leaks, rather than giving the auditor the tricky task of assessing the impact of many small leaks.

Several years before [25], X25519 already specified scalars n that always have $2^{254} \le \mathtt{n} < 2^{255}$, so a variable-length Montgomery ladder still ends up taking constant time. X25519 implementors are also encouraged in several ways to focus on implementing X25519, rather than to try to write "generic" implementations that handle multiple curves. One effect of this focus is that X25519 implementors have a natural incentive to avoid variable-length ladders: a variable-length ladder might seem simplest for a "generic" implementation but a length-255 ladder is obviously simplest for an X25519 implementation. To summarize, the X25519 ecosystem was proactively designed to discourage timing leaks.

3.2. Edwards curves. The simplest way to generate X25519 public keys is to reuse the Montgomery ladder to multiply the secret scalar by the standard base point B. It is well known, however, that one can save a factor of roughly

[8] Originally, in [9], this ECDH system was called Curve25519, but having a separate name for the ECDH system and the curve has turned out to be helpful.

3 in CPU time by instead computing this multiple of B as a sum of various precomputed multiples of B.

We emphasize that simply reusing the Montgomery ladder for key generation is fast enough for most ECDH applications. Obviously one should not add complexity (and code size) for a speedup that users won't actually notice. To find applications that care about the cost of ECDH key generation, one needs to find applications where ECDH is a bottleneck *and* each ECDH key is reused only a very small number of times. A key used as a long-term identifier, or an ephemeral server key reused for two hours,[9] involves mainly shared-secret computations, not key generation. Even in the rather extreme situation of a key being generated and used just once, saving a factor of 3 in key generation means saving a factor of just 1.5 in total time.

Furthermore, there are timing leaks in all of the simplest ways to work with tables of precomputed multiples of B. Consider, for example, a table containing $B, 2B, 4B, \ldots, 2^{254}B$. Simply scanning for the bits of `n` that are set, and adding the corresponding table entries, will (obviously) leak the Hamming weight of `n` through total time, and will (less obviously) leak all bits of `n` through higher-bandwidth timing channels such as cache-timing attacks and branch-prediction attacks. One way to protect this computation is to read *every* table entry, using conditional moves to replace irrelevant results with 0, but this is not as simple and fast as the original computation. See, e.g., [**15**] for the work needed to write reasonably fast constant-time key-generation code.

At a lower level, the implementor is faced with the problem of how to add two points. The Montgomery x-coordinate is not enough information to allow general additions.[10] The simplest solution is to use a different curve shape, "complete Edwards curves", which we published in [**18**] in 2007. A complete Edwards curve is a curve $x^2 + y^2 = 1 + dx^2y^2$ where d is non-square. The sum of two points (x_1, y_1) and (x_2, y_2) on the curve is

$$\left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}\right);$$

the denominators here are always nonzero. Various optimizations reduce the cost of each addition below 10 field multiplications.

The problem of general additions also appears in all of the signature systems described in Section 2. Complete Edwards curves are again the simplest solution. The Ed25519 signature system uses a complete Edwards curve for key generation, signing, and signature verification.

There is a slight speedup from switching to the "dual" addition formulas introduced by Hisil, Wong, Carter, and Dawson in [**37**], but these formulas have the disadvantage of producing incorrect results for some bad inputs. Most of the cryptographic literature assumes that computations are performed correctly, and the occasional analyses of the consequences of incorrect computations often show

[9] [**27**, Section 4.2] reported that Microsoft's SChannel works this way.

[10] To be precise: Given $X_0(Q)$ and $X_0(R)$ one can, with some effort, compute all possibilities for $X_0(Q + R)$. Usually there are exactly two possibilities.

that those computations can do tremendous damage. With these "dual" formulas one therefore has to be very careful

- to have the implementation check for bad inputs (in constant time!) and handle them appropriately; or, alternatively,
- to prove that the higher-level scalar-multiplication strategy cannot produce bad inputs; or, alternatively,
- to prove that the higher-level scalar-multiplication strategy together with the randomization of inputs has negligible chance of producing bad inputs.

Given the difficulty that this produces for auditors, together with the rarity of applications that will notice this slight speedup, we recommend that implementors avoid taking the "dual" approach to additions.

3.3. The NSA/NIST approach. FIPS 186-4 points to ANSI X9.62 for scalar-multiplication algorithms. ANSI X9.62 specifies[11] a single-scalar-multiplication algorithm (Appendix D.3.2) whose inner loop consists of

- a doubling,
- an addition if the scalar satisfies a particular property (probability 1/4), and
- a subtraction if the scalar satisfies another property (also probability 1/4).

The underlying addition algorithm (Appendix B.3) has five different cases.

Adding further complexity to the scalar-multiplication method, ANSI X9.62 cites "several variations of this method which can be used to speed up the computations". Even in applications where the original algorithm provides acceptable speed, the algorithm is obviously vastly more complex than the Montgomery ladder, and has many more traps for implementors. For example, random tests of this scalar-multiplication algorithm will exercise only two of the cases in the addition algorithm, so they won't protect an implementor who omits or mangles the other cases. Furthermore, both levels of algorithms in ANSI X9.62 are full of timing leaks.

It is of course possible to build a correct constant-time implementation (and the literature explains various ways to make this somewhat less painful than it might seem at first, for example by reducing the number of cases). However, within the space of all possible implementations, a secure implementation is surrounded in all directions by traps for the implementor: implementations that are faster but leak information through timing, implementations that pass typical tests but that are wrong for some attacker-supplied inputs, etc. The problem here is much more severe than the problem described above for "dual" addition formulas: the problem before was a slight tension between speed and security, while the problem here is a much stronger tension between speed and security, combined with a strong tension between simplicity and security.

One of the core choices in ANSI X9.62 was to use what are called "Jacobian coordinates" to represent points inside these computations. (This means

[11] The citations here are to the September 1998 draft. Subsequent drafts might have different material but aren't easy to find online and are unlikely to be checked by most implementors.

using (X, Y, Z) to represent a point $(x, y) = (X/Z^2, Y/Z^3)$ on a "short Weierstrass curve" $y^2 = x^3 + ax + b$.) This choice was publicly justified for speed reasons: IEEE P1363, a very similar standard developed in parallel by an overlapping team, claims that this choice provides "the fastest arithmetic on elliptic curves". However, all known methods for addition in Jacobian coordinates (and in the closely related "projective coordinates" for short Weierstrass curves), whether constant-time or not, are considerably slower and more complicated than constant-time addition on complete Edwards curves.

It is easy to explain why the standards don't mention Edwards curves: the standards were developed before Edwards curves were published. But it is much more difficult to explain why the standards don't mention the Montgomery ladder for single-scalar multiplication. The Montgomery ladder was published in the 1980s, and has always been simpler, faster, and less error-prone than the techniques recommended in the standards. To justify ignoring the Montgomery ladder it seems that one would have to (1) focus purely on speed without regard to simplicity, and also (2) focus purely on signatures without regard to ECDH.

4 Coordinates sent through the network

Section 3 described Montgomery curves, complete Edwards curves, and short Weierstrass curves as separate types of curves. However, there are actually many easy maps between these types of curves, so the choice of coordinates sent through the network is not dictated by the choice of coordinates used inside computations.

For example, define $p = 2^{255} - 19$ and $A = 486662$, and recall from Section 3 that Curve25519 is the Montgomery curve $y^2 = x^3 + Ax^2 + x$ over $\mathbf{F}_p$. It is easy to see that if (x, y) is a point on this curve then $(x + A/3, y)$ is a point on the short Weierstrass curve $y^2 = x^3 + ax + b$ where $a = 1 - A^2/3$ and $b = 2A^3/27 - A/3$. It is just as easy to work backwards from $(x + A/3, y)$ to (x, y).

The map from (x, y) on the Montgomery curve to the corresponding point $(x + A/3, y)$ on the short Weierstrass curve is an "isomorphism" between curves, preserving addition of points. An implementor who wants to compute the $\mathtt{n}$th multiple of a point Q on the short Weierstrass curve can

- invert this isomorphism (i.e., subtract $A/3$ from the first coordinate) to obtain the corresponding point on the Montgomery curve,
- use the Montgomery ladder to obtain the x-coordinate of the $\mathtt{n}$th multiple of that point,
- recover the y-coordinate of the $\mathtt{n}$th multiple, and
- apply the isomorphism to obtain $\mathtt{n}Q$.

As this example illustrates, it's possible for a protocol to use (e.g.) short Weierstrass curves while computations use (e.g.) Montgomery curves.

As another example, Curve25519 is also isomorphic to the complete Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ over $\mathbf{F}_p$, where $d = 1 - 1/121666$. The Ed25519 signature system is specified in terms of this complete Edwards curve, while the

X25519 ECDH system is specified in terms of the Montgomery curve. X25519 implementors who want faster key generation (see Section 3) use the Edwards curve for scalar multiplication and then apply the isomorphism to obtain a public key on Curve25519.

4.1. Invalid-curve attacks and twist security. Jager, Schwenk, and Somorovsky [39] recently announced a devastating break of "Java implementations using static EC keys" for TLS. Eight crypto libraries were analyzed, and it turned out that two of them (Bouncy Castle and Java Crypto Extension) didn't check whether incoming points (x, y) were on the curve that they were supposed to be on.

The attack works as follows. The attacker easily finds a point (x, y) of small prime order, let's say order 1009, on another curve over the same field, and sends that point to the server. The server blithely computes $k(x, y)$ where k is the server's secret key; this is the same as $(k \bmod 1009)(x, y)$. The attacker has a noticeable chance of correctly guessing $k \bmod 1009$, and confirms this guess by following normal protocol operations using the shared secret $(k \bmod 1009)(x, y)$, revealing $k \bmod 1009$. If the guess fails then the attacker tries again with another guess. By varying 1009 the attacker learns k modulo several small primes, and then easily computes k. The basic idea of this attack was published in 2000 [21] by Biehl, Meyer, and Müller, as an elliptic-curve adaptation of an idea published by Lim and Lee.

NIST has frequently asserted that the NIST curves *when implemented properly* are not subject to any attack known to the public:

> We remain confident in their security and are not aware of any significant attacks on the NIST curves when used as described in our standards and implemented correctly.

It is of course true that the implementors could have avoided the attack by checking whether (x, y) is on the correct curve. The NIST curves don't have any points of small order (except for orders dividing the "cofactors", something already taken care of by standard "cofactor multiplication"). However, the reality is that 25% of the implementations in this study *weren't* implemented correctly and *didn't* avoid the attack.

The underlying issue is that checking whether input points are on the curve is considerable extra complexity. Typical tests won't notice if this check is omitted, and even highly aggressive tests aren't effective at figuring out whether this check is implemented correctly for *all* inputs.[12] It might sound trivial to have implementors check whether an input (x, y) is on the correct curve, but this is actually a significant tension between simplicity and security, and this tension is exactly what led to the success of the attack.

[12] One of the OpenSSL bugs announced in 2015 was an error in the point-on-curve test for a very small fraction of inputs. It still isn't clear whether this error is exploitable. One can hope to eliminate all errors in ECC software through formal verification, and recent work shows considerable progress in this direction; this work is targeting X25519 implementations precisely because those implementations are so simple.

X25519 proactively avoids this attack as follows:

- Instead of sending short-Weierstrass (x, y) through the network, send Montgomery x. This choice of coordinates drastically limits the attacker's choice of curves: it does not quite force the attacker to send a point on the correct curve, but the only other possibility is that the attacker sends a point on what is called "the twist" of the curve.
- Encourage use of the Montgomery ladder. This has many benefits discussed in Section 3. What matters here is another effect, namely that scalar multiplication is computed correctly for all inputs x, both on the original curve and on the twist.
- Choose a curve so that the curve and the twist *each* have no points of small order, except for orders dividing the cofactor.

The Montgomery x is not the only possibility here: there are analogous defenses that instead send the Edwards y. Choosing to send the Edwards y would slightly simplify implementations that use Edwards coordinates internally. However, choosing the Montgomery x slightly simplifies implementations that use the Montgomery ladder internally. These implementations are simpler in the first place, so the marginal extra simplification is more noticeable. For similar reasons, it is better to send the Montgomery x than the short-Weierstrass x.

In short, this tension between simplicity and security was and is avoidable through the choice of ECDH mechanisms. By blaming implementors for this attack, NIST has been refusing to acknowledge its own role in causing the problem.

4.2. Signatures and point compression. As noted in Section 3, the signature context is more complicated than the ECDH context. Signature verification uses general point additions; a single coordinate can support a ladder but can't support general point additions. It's possible to arrange signature verification as *addition verification*, which can be done with a single coordinate, but this interferes with speed and isn't particularly simple. Working with both the Edwards x and y coordinates is simpler.

In this two-coordinate context, ECC standards can and should require *compression* of the second coordinate. For example, Ed25519 sends the Edwards y and a single bit specifying x. This saves 32 bytes, sometimes producing a measurable speedup. Recovering x from the single bit sometimes produces a measurable slowdown, but the savings typically outweighs the slowdown. More importantly, instead of telling implementors to check the whole curve equation, Ed25519 is telling implementors to check whether x was recovered correctly, i.e., to check an equation of the form $x^2 = s$. This is considerably less code, considerably reducing the tension between simplicity and security. It's also much more fault-tolerant than checking the curve equation, since it's checking ℓ bits derived from $\ell + 1$ bits of attacker input rather than 2ℓ bits of attacker input.

Today's ECC standards allow compression as an *option*, but this obviously isn't good enough to stop attacks, and it also ends up adding complexity for implementors.

4.3. Unified implementations of ECDH and signatures. The bigger picture of the ECDH+signature ecosystem is that some implementations support only ECDH, some implementations support only signatures, and some implementations support both. Using Edwards coordinates for ECDH would slightly simplify implementations of the third type, but using Montgomery coordinates for ECDH slightly simplifies implementations of the first type. Implementations of the first type are spectacularly simple (see Section 3), so slight simplifications are much more noticeable for them than for implementations of the other types. We therefore recommend Montgomery coordinates for ECDH, as in X25519.

Using the same prime for signatures and for ECDH is of course helpful for implementations of the third type. Using isomorphic curves (as X25519 and Ed25519 do) gives these implementors more options to share code between ECDH key generation, signing, etc.

5 Curves

Not all elliptic curves are compatible with the next-generation ECC options explained in previous sections. However, it is very easy to find compatible curves. Specifically, approximately 25% of all elliptic curves over large prime fields satisfy the following two conditions:

- The curve has a point of order 4.
- The curve has a unique point of order 2.

Any such curve supports complete Edwards coordinates, and also supports Montgomery coordinates with the $A^2 - 4$ non-squareness condition used in the proofs in [9].

One needs to search through many of these curves to find curves with small cofactors (a standard requirement). Furthermore, one needs to search through many curves with small cofactors to find curves whose *twists* also have acceptably small cofactors (see Section 4).

Taking a very small parameter d for an Edwards curve $x^2 + y^2 = 1 + dx^2y^2$ produces a small but measurable performance improvement compared to taking a random d. It has become common practice to take the *smallest* acceptable integer d; the performance justification here is miniscule, but taking any larger value of d would be hard to justify. This practice is comforting for people concerned about the flexibility available to the curve generator, as in Section 1. Before Edwards curves were known, analogous considerations dictated taking the smallest acceptable integer $(A - 2)/4$ for a Montgomery curve $y^2 = x^3 + Ax^2 + x$; this is how Curve25519 was generated.

6 Primes

The literature on multiprecision arithmetic—software for multiplying larger integers than the hardware is designed to multiply—is full of variable-time algorithms. Constant-time field arithmetic therefore requires special attention from software implementors.

6.1. Constant-time arithmetic. Constant-time software implementations are similar to hardware implementations: they allocate a constant number of bits for each integer, and always perform arithmetic on all bits, without skipping bits. For example:

- If the goal is to add a to b, where 255 bits are allocated for a and 255 bits are allocated for b: Allocate 256 bits for $a + b$. Of course, it's possible that $a + b$ would fit into 255 bits, but don't check.
- If the goal is to multiply a by b, where 256 bits are allocated for a and 256 bits are allocated for b: Allocate 512 bits for ab.

The reason that this strategy does not spiral out of control is that for an elliptic curve modulo p one is free to reduce modulo p at any moment. The details of this reduction depend on the choice of p.

6.2. Reduction modulo $2^{255} - 19$. Consider, for example, 600 bits allocated to hold an integer c modulo $p = 2^{255} - 19$. Replace c with $19q + r$, where $r = c \bmod 2^{255}$ and $q = \lfloor c/2^{255} \rfloor$, after allocating 350 bits for $19q + r$. It's easy to see that $19q + r$ is the same as c modulo p. Repeating the same idea reduces 350 bits to 256 bits, small enough for the next multiplication.

At the end of scalar multiplication it's important to *completely* reduce the output modulo p. This takes two iterations of constant-time conditional subtraction. One conditional subtraction, by definition, replaces c with $c - (1 - s)p$ where s is the sign bit in $c - p$.

6.3. Reduction modulo the P-256 prime. For comparison, the NIST P-256 prime p is $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. The ECDSA standard specifies the following reduction procedure given an integer "A less than p^2":

- Write A as the vector
$$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9, A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0),$$
meaning $\sum_i A_i 2^{32i}$.
- Define
$$\begin{aligned}
T &= (A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);\\
S_1 &= (A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);\\
S_2 &= (0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);\\
S_3 &= (A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);\\
S_4 &= (A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);\\
D_1 &= (A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});\\
D_2 &= (A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});\\
D_3 &= (A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});\\
D_4 &= (A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).
\end{aligned}$$
- Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.
- Reduce modulo p "by adding or subtracting a few copies" of p.

This is considerably more complicated, and considerably slower, than reduction modulo $2^{255} - 19$. Even worse, what does "a few copies" mean in the final step? This sounds like a loop, taking variable time, presumably a security problem.

With some effort one can see that the following constant-time procedure always works for the final step:

- Conditionally add $4p$.
- Conditionally add $2p$.
- Conditionally add p.
- Conditionally subtract $4p$.
- Conditionally subtract $2p$.
- Conditionally subtract p.

This is, however, quite slow. One might try to delay this procedure until the end of the computation, but then the next multiplication will violate the "A less than p^2" requirement. One can write down a revised reduction algorithm that works for larger values of A, but this adds even more complexity.

Even worse, this reduction procedure assumes that integers are expressed in radix 2^{32}, but the literature clearly shows that 2^{32} is not the best radix for additions and multiplications on most platforms. Implementors who try a different radix for faster additions and multiplications will need to insert many bit shifts and extractions into the NIST P-256 reduction procedure, incurring further complexity and cost.

6.4. A few good primes. The literature contains cost analyses, at various levels of detail, of field arithmetic on many different platforms modulo many different primes. These analyses strongly suggest that a few primes provide particularly good cross-platform performance for their size.

We are not aware of any reason to allow a prime that isn't Pareto-optimal: a prime for which it's possible to gain cross-platform performance by switching to another prime that's at the same (or higher) security level. We suggest being even more restrictive than this, and excluding any prime for which it's possible to gain cross-platform performance by switching to another prime that's within 1 bit of the same size (or larger). Small differences in security aren't meaningful: if someone has enough computer power to break a prime, it doesn't make any sense to respond by switching to another prime that's just one bit larger. This rule still leaves some particularly fast primes, such as $2^{255} - 19$ and $2^{521} - 1$.

7 "Digital signature schemes": questions from NIST

7.1. "Do the digital signature schemes and key sizes specified in FIPS 186-4 satisfy the security requirements of applications used by industry?"

No. See above.

7.2. "Are there other digital signature schemes that should be considered for inclusion in a future revision to FIPS 186? What are the advantages of these schemes over the existing schemes in FIPS 186?"

Yes. See above, particularly Section 2.

8 "Security of elliptic curves": questions from NIST

8.1. "Do the NIST-recommended curves satisfy the security requirements of applications used by industry?"
No. See above.

8.2. "Are there any attacks of cryptographic significance on Elliptic Curve Cryptography that apply to the NIST-recommended curves or other widely used curves?"
Yes. See above.

9 "Elliptic curve specifications and criteria": questions from NIST

9.1. "Is there a need for new elliptic curves to be considered for standardization?"
Presumably "standardization" here means standardization by NIST, rather than, e.g., standardization by IETF or de-facto standardization by the community. The answer still depends on what NIST means by "need" and by "new".

Next-generation ECC has already been widely deployed. Nicolai Brown maintains public lists[13] of "Things that use Curve25519" [23] and "Things that use Ed25519" [24]. The lists include, along with many other examples,

- Apple's iOS (iPhone, iPad, etc.) operating system;
- the TextSecure (Signal) messaging system;
- the standard OpenSSH remote-login software;
- the Tor network; and
- Google's QUIC protocol.

Our impression is that NIST P-256, the most popular NIST curve, has now been surpassed by Curve25519 in a wide range of usage metrics.

The reasons for users selecting Curve25519 over NIST P-256 have been amply documented. Maybe these reasons aren't relevant to NIST's "needs"; we haven't seen a definition of NIST's "needs". As for "new", there are clearly large differences between "new to NIST's ECC standards", "new to real-world ECC deployment", and "newly developed".

9.2. "If there is a need, what criteria should NIST use to evaluate any curves to be considered for inclusion?"
There was extensive discussion of curve criteria on the CFRG mailing list in 2014 and 2015. NIST may find this discussion useful as a starting point.

[13] We incorporate those lists by reference into our comments to NIST.

Obviously all proposed elliptic curves must resist all publicly known ECDLP attacks. The specified base point on the curve must have large enough prime order to resist rho attacks; must be immune to additive transfers; and must have large enough embedding degree to resist multiplicative transfers.

There might be submissions of curves that (like NIST K-163) are clearly *feasible* but *expensive* to break. Such submissions will force NIST to quantify its minimum acceptable security level. The first author has a new blog post [11] pointing out that NIST's previous evaluations of quantitative security levels have failed to take into account the damage done by *batch* attacks.[14]

In general the modern trend is towards conservative prime-field ECC. Note that asymptotically 100% of all curves over all finite fields, counted in the usual way, are curves over prime fields; in other words, curves over non-prime fields are, mathematically, extremely rare. The advent of "Weil descent" broke ECDLP for many curves over non-prime fields, and the limits of Weil descent are still not entirely clear, as illustrated by ongoing debates regarding whether Weil descent takes subexponential time to break ECDLP for all binary fields. There are interesting hardware performance advantages to binary curves (especially Koblitz curves), and NIST's current binary curves are clearly not broken by the best Weil-descent attacks known today; on the other hand, securely implementing binary-field arithmetic in *software* is considerably more difficult and error-prone than implementing prime-field arithmetic, and the extra structure of these curves makes security analysis more difficult and less confidence-inspiring.

There might also be, again for interesting performance reasons, proposals of genus-2 hyperelliptic curves; special curves over prime fields with extra "endomorphisms" (GLV curves); special curves over quadratic extension fields with extra endomorphisms ($\mathbf{Q}$-curves, including GLS curves); etc. We believe that considering any of these curves will be a distraction from fixing the problems with NIST's current ECC standards. We specifically recommend that NIST disregard any GLV/GLS/$\mathbf{Q}$-curve proposals: the GLV patents are still valid and cover all use of extra endomorphisms to speed up scalar multiplication, so these curves will simply end up adding worrisome structure without any noticeable benefits in speed and of course without any benefits in simplicity.

In this document and in our SafeCurves [19] web site,[15] we consider the security of ECC *implementations*, looking beyond the narrow question of ECDLP security. This means paying attention to the security impact of implementors pursuing simplicity, the security impact of implementors pursuing speed, risks of various types of errors in implementations, etc. We strongly encourage NIST to do the same.

To take into account the big picture of simplicity, security, and speed, we recommend that NIST adopt the following statement of principles:

- Principle 1: We want speed and simplicity (and speed-simplicity combinations) for secure implementations.

[14] We incorporate that blog post by reference into our comments to NIST.

[15] We incorporate the SafeCurves web site by reference into our comments to NIST.

- Principle 2: We want to avoid speed incentives and simplicity incentives towards insecure implementations.
- "Secure" includes the following properties: an implementation works correctly for all valid inputs; avoids compromising security when inputs are invalid; avoids all data flow from secrets to timing; etc.
- "Speed" includes speed on many different platforms: e.g., speed on 64-bit desktop CPUs, speed on 32-bit smartphone CPUs, speed on embedded microcontrollers, and speed of hardware implementations.
- "Speed" includes speed of many important ECC operations: e.g., key generation, signing, verification, and DH shared-secret computation. "Simplicity" includes the simplicity of implementations supporting all of these operations, the simplicity of implementations supporting only DH, the simplicity of implementations supporting only signatures, etc.
- "Simplicity" includes the simplicity of new implementations, and the simplicity of modifying preexisting implementations.

More specific requirements can be derived from these general principles. For example, SafeCurves requires twist-secure curves supporting (1) simple, fast, complete, constant-time single-coordinate single-scalar multiplication and (2) simple, fast, complete, constant-time multi-scalar multiplication. "Fast" means that implementations of scalar multiplication for the same curve cannot be much faster, and "simple" means that reasonably fast implementations of scalar multiplication for the same curve cannot be much more concise. Our analysis indicates that the details of these requirements are forced by Principle 2, avoiding incentives towards insecure implementations.

When specific requirements (or statements of desiderata) are in conflict with these general simplicity/security/speed principles, the requirements should be corrected. Consider, for example, the following claim from NSA in NIST's curve standard: "For efficiency reasons, it is desirable to take the cofactor to be as small as possible." All of the NIST prime-field curves have cofactor 1. However, this extreme cofactor requirement actually produces a slowdown: the NIST curves are considerably slower than Edwards curves at similar (or even somewhat higher) security levels, despite the fact that Edwards curves always have cofactor at least 4. For DH this slowdown was already clear from the literature predating NSA's claim. We do not see any reason for violating the cofactor limits stated in FIPS 186-4 (at least 2^{10}, depending on the size of p), but we also do not see any justification for those specific limits.

9.3. "Do you anticipate a need to create, standardize or approve new elliptic curves on an ongoing basis?"

No (despite the aforementioned caveats regarding NIST's "needs"). Replacing "ECDLP security" with "ECC security" as the goal is a one-time correction.

10 "Adoption": questions from NIST

10.1. "Which of the approved digital signature schemes and NIST-recommended curves have been used in practice?"

For a typical network packet today, the answer is “none”. NIST’s ECC standards are not being used to protect that packet against espionage, forgery, and sabotage. Either the packet is protected by a different (probably lower-security) mechanism, or it isn’t protected at all.

We believe that all network packets should be protected by strong cryptography, and that ECC is the community’s best hope for getting this done in the near future.[16] The packets that are *not* protected by ECC today should be viewed by NIST as the most important target for any new ECC standards.

This target is not accurately reflected by the corner case of people who are happily using NIST’s ECC standards today. Data regarding this corner case is of some interest but should not be misunderstood as data regarding future use of ECC.

10.2. “Which elliptic curves are accepted for use in international markets?”

The international situation is not very different from the U.S. situation. A few countries (France, Germany, Russia, and China) are encouraging use of their locally developed curves, but they seem to have had only limited success. See also Section 11.3.

11 “Interoperability”: questions from NIST

11.1. “If new curves were to be standardized, what would be the impact of changing existing implementations to allow for the new curves?”

The answer depends on the curve.

There are many freely available implementations of Curve25519. For example, TweetNaCl [16] is a complete self-contained portable cryptographic library that fits into 100 tweets, and implements the most important Curve25519 use cases (X25519 and Ed25519). There are many other implementations of Curve25519 optimized for many different platforms, supporting other languages, etc., and in most cases supporting the same well-known easy-to-use API, with extensive tests provided by the SUPERCOP benchmarking framework. Various Curve25519 implementations are also the targets of state-of-the-art ECC software verification. People can, should, and do simply use these implementations.

[16] It has been well known since the 1990s that quantum computers break ECC. There are proposals for post-quantum systems, and there will obviously be more and more users who deploy such proposals. However, this deployment is not a serious argument against deploying ECC: for the foreseeable future this deployment will be done as a *supplement* to ECC, not as a *replacement* for it. Most post-quantum public-key systems have not been thoroughly studied, so users want ECC as a backup to provide some security in case of catastrophic failure. There are some conservative post-quantum public-key systems, such as the systems recently recommended [5] by the European PQCRYPTO consortium, but those systems are expensive enough that users who can afford them as replacements for ECC can also afford them as supplements to ECC.

It would not be very difficult to build up a similar ecosystem of implementations of a higher-security next-generation curve, such as E-521. See our paper [13] with Chuengsatiansup for several reasons that a higher-security curve might be of interest.

Old-fashioned curves are considerably more difficult to implement, and much more likely to produce low-quality (complex, slow, breakable) implementations. History shows that the lowest-quality ECC implementations are "generic" implementations that try to handle many different curves[17] with a single code base. OpenSSL's "generic" ECC implementation is complex, slow, hard to audit, likely to be buggy, and likely to be breakable by timing attacks; OpenSSL has improved code quality by adding a separate NIST P-256 implementation, a separate NIST P-384 implementation, etc.

To the extent that code size is a problem (for auditing, for verification, for small devices, etc.), the obvious solution to the problem is to prune the list of supported curves. This solution is much more effective, and produces higher-quality results, than trying to merge implementations of many curves into a single "generic" implementation.

For someone who really wants to merge an implementation of curve C into an existing implementation of curve B, the exact implementation difficulty depends on B, C, and many details of the existing implementation. For example, if a NIST-P-curve implementation assumes curves of the form $y^2 = x^3 - 3x + b$ in Weierstrass coordinates, then generalizing it to $y^2 = x^3 + ax + b$ will typically require rearranging some field-arithmetic calls, and adapting it to Edwards coordinates will typically require some extra lines of code to convert from Edwards coordinates to Weierstrass coordinates and back. Of course, when evaluating existing implementations of the NIST curves, NIST should also consider the question of whether those implementations are secure and whether those implementations are successfully protecting typical users. Many existing implementations should simply be removed and replaced by new implementations of better curves.

11.2. "What is the impact of having several standardized curves on interoperability?"

A client's ECDH implementation cannot interoperate with a server's ECDH implementation unless there is at least *one* curve that is supported by both the client and the server. Furthermore, the client and server need to be able to securely *figure out* which curve that is, and exchange keys using that curve. Similar comments apply to other ECC protocols, such as signatures.

The simplest way for a protocol to guarantee interoperability is to specify a single curve to be used by all clients and all servers. One generalization that guarantees interoperability is to specify a set of curves to be supported by all clients; each server can then make its own curve choice (and securely authen-

[17] These "generic" implementations actually have various restrictions on the set of curves supported. Should an implementation be called "generic" if it can't handle binary fields? Quadratic extension fields? Elliptic curves that aren't of the form $y^2 = x^3 - 3x + b$? Curves beyond genus 1?

ticate this choice along with authenticating the server's ECC key). A different generalization that guarantees interoperability is to specify a set of curves to be supported by all *servers*; each *client* can then make its own curve choice. Note that these generalizations cause obvious damage to implementation simplicity and to performance, especially in situations where a client sends a key to a server whose curve choice isn't known to the client, or vice versa.

Some protocols don't actually guarantee ECC interoperability, even when both the client and the server support ECC. For example, TLS seems to have standardized a surprisingly large number of curves without thinking these issues through. Because of the same issues of implementation simplicity and performance, each widely used TLS implementation supports only a subset of the curves, and nothing in the TLS protocol guarantees that these subsets will overlap. If a certificate authority issues ECC certificates using a Brainpool curve or one of the more obscure NIST curves, will a client understand the certificates? The main reason that "ECDHE" works in TLS today is that implementors have collectively decided to support NIST P-256, in effect taking the simple approach to guaranteeing interoperability.

To the extent that the profusion of standardized curves causes interoperability problems, one can and should blame the standards for encouraging these problems. Does NIST think that having a long list of standardized curves is a *good* thing? For each of its existing curves, and for any new curves that are proposed, NIST's *default* assumption should be that having the curve standardized is a bad idea.

Obviously this default can, and occasionally should, be overridden. NIST ECC is failing quite disastrously in practice, in ways that are fixed by next-generation ECC, and there is already widespread adoption of next-generation ECC. But the question for any particular curve shouldn't be "Does standardizing this curve have a benefit?"; it should be "Does standardizing this curve have a large enough benefit to outweigh the costs?"

11.3. "What are the advantages or disadvantages of allowing users or applications to generate their own elliptic curves, instead of using standardized curves?"

Computing n independent ℓ-bit discrete logarithms on one properly chosen curve costs roughly $2^{\ell/2}\sqrt{n}$ additions, while computing n independent ℓ-bit discrete logarithms on n independent properly chosen curves costs roughly $2^{\ell/2}n$ additions (by the best attack algorithms known today). This means that there is a quantitative security advantage in having each key on its own curve.

However, taking a single curve with a somewhat larger ℓ produces a much larger quantitative security advantage at much lower cost. We recommend taking ℓ large enough that foreseeable attackers (pre-quantum) will be limited to $2^{\ell/2}\epsilon$ additions for a small value of ϵ. The attacker's chance of finding even *one* of the n discrete logarithms is then limited to approximately ϵ^2. This strategy was used in the design of Curve25519, and all of these security issues were described in the Curve25519 paper.

Regarding cost: As long as *primes* are standardized, allowing random curve parameters does not produce a huge slowdown in curve arithmetic. However, constantly changing curves means constantly transmitting new curve parameters, which is a noticeable cost in bandwidth. More importantly, it means constantly generating new curves, which is not only a significant expense in CPU time but also a nightmarish increase in complexity.

Of course, a user who generates his own elliptic curves is protected against the possibility of a malicious curve generator choosing the weakest curve from the same pool, using a weakness not known to the public. However, this is merely a *possibility*, while there are *definite* problems with excessive complexity (and slowness), so it seems unwise to try to eliminate this possibility at the expense of a drastic increase in complexity.

Many ECC-based protocols (e.g., group key exchange) require multiple users to share a curve, raising the question of who will be trusted to generate the curve. Multiparty computation is a theoretical answer but adds even more complexity.

More moderate curve pools (one curve per application, one curve per country, etc.) would practically eliminate the costs in CPU time and bandwidth, but would also practically eliminate the security advantages. The huge complexity disadvantage would remain.

12 "Performance": questions from NIST

12.1. "Do the performance characteristics of existing implementations of the digital signatures schemes approved in FIPS 186-4 meet the requirements of applications used by industry?"

This depends on the application. Note that this question is another example of how NIST can get a quite wrong picture by listening too much to people who are happily using the NIST curves today, and not thinking enough about the much larger set of people who *should* be using ECC.

Many applications can afford huge elliptic curves and can afford slow implementations. Often one or two ECC operations are used to protect a very long message (or a message requiring extensive non-ECC computations for other reasons). Often CPUs are idle, or are so busy with something other than ECC that the cost of ECC is negligible.

On the other hand, sometimes ECC has a serious impact on cost. Consider, for example, the cost of protecting against denial of service when a busy Internet server is handling many packets per second from different sources. An attacker can try to flood the network, but an attacker can also try to flood the CPU with new public keys to process. If the CPU cannot keep up with the cost of public-key cryptography then the server will have to drop traffic from new *legitimate* keys. As another example, deployment of public-key cryptography in a wide variety of small devices relies critically on being able to make ECC fast enough and small enough.

On an Intel Sandy Bridge CPU (common in data centers today), OpenSSL 1.0.2's optimized NIST P-256 implementation takes 310000 cycles for ECDH shared-secret computation and 430000 cycles for ECDSA signature verification, while Tung Chou's software takes only 160000 cycles for X25519 shared-secret computation and only 206000 cycles for Ed25519 signature verification. This is just one example of how Curve25519 "over the past ten years has set speed records for conservative ECC on many different platforms, using implementations from 23 authors", as stated in [12, Section 7]. See [9] (original paper for various 32-bit CPUs), [34] (Core 2, Athlon 64), [30] (Cell), [15] (more Intel CPUs), [20] (NEON), [41] (more Intel CPUs), [42] (GPUs), [51] (FPGAs), [28] (more Intel CPUs), [31] (microcontrollers), and [38] (ASICs).

13 "Intellectual property": questions from NIST

13.1. "What are the desired intellectual property requirements for any new curves or schemes that could potentially be included in the Standard?"

Most clients and servers for Internet communication (web, email, chat, collaboration, etc.) are distributed and used for free, without payment of license fees. This has proven to be a productive environment for rapid deployment of patent-free cryptography, while patented cryptography is simply not considered.

As far as we know, the most important ECC patents have all expired, and the entire core of ECC is now patent-free. Patent 4995082 (Schnorr) expired in 2008. Patents 5159632, 5271061, and 5463690 (the Crandall patents on "shift and add" ECC primes) expired in 2011. Patent 6141420 (point compression) expired in 2014. Patents 5299262 and 5999627 (fixed-base scalar multiplication by Brickell–Gordon–McCurley and Lim–Lee) expired in 2012 and 2015 respectively. We have not found any current patents covering, e.g., X25519 (ECDH with Curve25519) or Ed25519 signatures.

Some fringes of ECC are still patented. The most interesting remaining patents are the GLV patents; these patents cover the use of endomorphisms to speed up ECC on special curves (GLV curves, GLS curves, $\mathbf{Q}$-curves, and so on). However, fixing the problems with NIST's ECC standards does not require going near these fringes.

To summarize, focusing on patent-free ECC obviously has important advantages, and there is no evidence of any important disadvantages.

13.2. "What impact has intellectual property concerns had on the adoption of elliptic curve cryptography?"

Certicom's web page asserts that Certicom owns "over 450 patents and patents pending worldwide covering key aspects of Elliptic Curve Cryptography (ECC)". A Wikipedia page, highlighting Certicom and citing an incident from a decade ago, asserts that "Patent-related uncertainty around elliptic curve cryptography (ECC), or ECC patents, is one of the main factors limiting its wide acceptance."

"Over 450 patents and patents pending worldwide" might sound like a large minefield, obviously difficult for anyone to review. In fact, Certicom has systematically inflated this number, by splitting each of its ideas into many separate patent applications. For example, Certicom's Dual EC patent application was split across at least four jurisdictions: Canada (CA 2594670), Europe (EP 06704329), Japan (JP 5147412), and the United States. In the United States it was split into patent 8396213 and patent application 2013/0170642, with the possibility of further splits. As another example, Certicom repeatedly split its GLV patent application, producing a quite noticeable fraction of all of Certicom's US patent applications. The number of different ideas actually patented by Certicom in the US is vastly smaller than 450, and it seems unlikely that Certicom has significantly more applications (or significantly different applications) in other jurisdictions.

It is reasonably clear that, as a historical matter, Certicom was successful in spreading fear, uncertainty, and doubt regarding its patents, drastically limiting deployment of ECC for many years. However, today the general consensus is that ECC is safe to deploy without patent fees. Our impression is that public perception was significantly affected by the publication of RFC 6090 in 2011: this RFC was carefully limited to sources so old that they were obviously not patented,[18] and nevertheless described a usable form of ECC similar to NIST's ECC standards. We are not aware of any contributions of Certicom to the forms of ECC that we recommend today.

References

[1] — (no editor), *Proceedings of the 23rd USENIX security symposium, August 20–22, 2014, San Diego, CA, USA*, USENIX, 2014. See [27].

[2] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, Yuval Yarom, *Amplifying side channels through performance degradation* (2015). URL: https://eprint.iacr.org/2015/1141. Citations in this document: §3.

[3] Diego F. Aranha, Alfred Menezes (editors), *Progress in cryptology—LATINCRYPT 2014—third international conference on cryptology and information security in Latin America, Florianópolis, Brazil, September 17–19, 2014, revised selected papers*, Lecture Notes in Computer Science, 8895, Springer, 2015. ISBN 978-3-319-16294-2. See [16].

[4] Vijay Atluri, Claudia Díaz (editors), *Computer security—ESORICS 2011—16th European symposium on research in computer security, Leuven, Belgium, September 12–14, 2011, proceedings*, Lecture Notes in Computer Science, 6879, Springer, 2011. ISBN 978-3-642-23821-5. See [25].

[5] Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, Tanja Lange, Mohamed Saied Emam Mohamed, Christian Rechberger, Peter Schwabe, Nicolas Sendrier, Frederik Vercauteren, Bo-Yin Yang, *Initial recommendations of long-term secure post-quantum systems* (2015). URL: https://pqcrypto.eu.org/docs/initial-recommendations.pdf. Citations in this document: §16.

[6] George Barwood, *Digital signatures using elliptic curves*, message 32f519ad.19609226@news.dial.pipex.com posted to sci.crypt (1997). URL: https://groups.google.com/group/sci.crypt/msg/b28aba37180dd6c6. Citations in this document: §2.

[18] We suggest labeling dates "B.C." for "Before Certicom".

[7] Lejla Batina, Matthew Robshaw (editors), *Cryptographic hardware and embedded systems—CHES 2014—16th international workshop, Busan, South Korea, September 23–26, 2014, proceedings*, Lecture Notes in Computer Science, 8731, Springer, 2014. ISBN 978-3-662-44708-6. See [13].

[8] Mihir Bellare (editor), *Advances in cryptology—CRYPTO 2000, proceedings of the 20th annual international cryptology conference held in Santa Barbara, CA, August 20–24, 2000*, Lecture Notes in Computer Science, 1880, Springer, 2000. ISBN 3-540-67907-3. MR 2002c:94002. See [21].

[9] Daniel J. Bernstein, *Curve25519: new Diffie-Hellman speed records*, in PKC 2006 [**54**] (2006), 207–228. URL: https://cr.yp.to/papers.html#curve25519. Citations in this document: §3.1, §3.1, §3.1, §8, §5, §12.1.

[10] Daniel J. Bernstein, *How to design an elliptic-curve signature system* (2014). URL: http://blog.cr.yp.to/20140323-ecdsa.html. Citations in this document: §2.

[11] Daniel J. Bernstein, *Break a dozen secret keys, get a million more for free* (2015). URL: http://blog.cr.yp.to/20151120-batchattacks.html. Citations in this document: §9.2.

[12] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, Christine van Vredendaal, *How to manipulate curve standards: a white paper for the black hat*, in SSR 2015 (2015). URL: https://bada55.cr.yp.to/. Citations in this document: §1.1, §12.1.

[13] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, *Curve41417: Karatsuba revisited*, in CHES 2014 [**7**] (2014), 316–334. URL: https://eprint.iacr.org/2014/526. Citations in this document: §11.1.

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, in CHES 2011 [**48**] (2011), 124–142; see also newer version [**15**]. URL: https://eprint.iacr.org/2011/368.

[15] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, Journal of Cryptographic Engineering **2** (2012), 77–89; see also older version [**14**]. URL: https://eprint.iacr.org/2011/368. Citations in this document: §2, §3.2, §12.1.

[16] Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe, Sjaak Smetsers, *TweetNaCl: a crypto library in 100 tweets*, in LatinCrypt 2014 [**3**] (2015), 64–83. URL: https://cr.yp.to/papers.html#tweetnacl. Citations in this document: §11.1.

[17] Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *EdDSA for more curves* (2015). URL: https://eprint.iacr.org/2015/677. Citations in this document: §2.

[18] Daniel J. Bernstein, Tanja Lange, *Faster addition and doubling on elliptic curves*, in Asiacrypt 2007 [**40**] (2007), 29–50. URL: https://eprint.iacr.org/2007/286. Citations in this document: §3.2.

[19] Daniel J. Bernstein, Tanja Lange, *SafeCurves: choosing safe curves for elliptic-curve cryptography* (2015). URL: https://safecurves.cr.yp.to. Citations in this document: §9.2.

[20] Daniel J. Bernstein, Peter Schwabe, *NEON crypto*, in CHES 2012 [**49**] (2012), 320–339. URL: https://cr.yp.to/papers.html#neoncrypto. Citations in this document: §12.1.

[21] Ingrid Biehl, Bernd Meyer, Volker Müller, *Differential fault attacks on elliptic curve cryptosystems (extended abstract)*, in Crypto 2000 [**8**] (2000), 131–146. URL: http://lecturer.ukdw.ac.id/vmueller/publications.php. Citations in this document: §4.1.

[22] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, Eric Wustrow, *Elliptic curve cryptography in practice*, in FC 2014 [**29**] (2014), 157–175. URL: https://eprint.iacr.org/2013/734.pdf. Citations in this document: §2.

[23] Nicolai Brown, *Things that use Curve25519* (2015). URL: http://ianix.com/pub/curve25519-deployment.html. Citations in this document: §9.1.

[24] Nicolai Brown, *Things that use Ed25519* (2015). URL: http://ianix.com/pub/ed25519-deployment.html. Citations in this document: §9.1.

[25] Billy Bob Brumley, Nicola Tuveri, *Remote timing attacks are still practical*, in ESORICS 2011 [**4**] (2011), 355-371. URL: https://eprint.iacr.org/2011/232. Citations in this document: §3.1, §3.1.

[26] "Bushing", Hector Martin "marcan" Cantero, Segher Boessenkool, Sven Peter, *PS3 epic fail* (2010). URL: http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf. Citations in this document: §2.

[27] Stephen Checkoway, Matthew Fredrikson, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, *On the practical exploitability of Dual EC in TLS implementations*, in USENIX Security 2014 [**1**] (2014). URL: https://projectbullrun.org/dual-ec/index.html. Citations in this document: §9.

[28] Tung Chou, *Sandy2x: new Curve25519 speed records*, in SAC 2015 (2015). URL: https://www.win.tue.nl/~tchou/papers/sandy2x.pdf. Citations in this document: §12.1.

[29] Nicolas Christin, Reihaneh Safavi-Naini (editors), *Financial cryptography and data decurity: 18th international conference, FC 2014, Christ Church, Barbados, March 3–7, 2014*, Lecture Notes in Computer Science, 8437, Springer-Verlag, 2014. See [22].

[30] Neil Costigan, Peter Schwabe, *Fast elliptic-curve cryptography on the Cell Broadband Engine*, in Africacrypt 2009 [**47**] (2009), 368–385. URL: https://cryptojedi.org/users/peter/#celldh. Citations in this document: §12.1.

[31] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, Peter Schwabe, *High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers*, Designs, Codes and Cryptography **77** (2015), 493–514. URL: http://link.springer.com/article/10.1007/s10623-015-0087-1/fulltext.html. Citations in this document: §12.1.

[32] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, Ingrid Verbauwhede, *State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures*, in HOST 2010 [**46**] (2010), 76–87. URL: http://rijndael.ece.vt.edu/schaum/papers/2010hostf.pdf. Citations in this document: §3.

[33] Pierrick Gaudry, *Variants of the Montgomery form based on Theta functions* (2006); see also newer version [**34**]. URL: http://www.loria.fr/~gaudry/publis/toronto.pdf.

[34] Pierrick Gaudry, *Fast genus 2 arithmetic based on Theta functions*, Journal of Mathematical Cryptology **1** (2007), 243–265; see also older version [**33**]. URL: http://webloria.loria.fr/~gaudry/publis/arithKsurf.pdf. Citations in this document: §12.1.

[35] Diana Goehringer, Marco Domenico Santambrogio, João M. P. Cardoso, Koen Bertels (editors), *Reconfigurable computing: architectures, tools, and applications—10th international symposium, ARC 2014, Vilamoura, Portugal, April 14–16, 2014, proceedings* (2014). ISBN 978-3-319-05959-4. See [51].

[36] Tim Güneysu, Helena Handschuh (editors), *Cryptographic hardware and embedded systems—CHES 2015—17th international workshop, Saint-Malo, France, September 13–16, 2015, proceedings*, Lecture Notes in Computer Science, 9293, Springer, 2015. ISBN 978-3-662-48323-7. See [38].

[37] Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, Ed Dawson, *Twisted Edwards curves revisited*, in Asiacrypt 2008 [**45**] (2008), 326–343. URL: `https://eprint.iacr.org/2008/522`. Citations in this document: §3.2.

[38] Michael Hutter, Jürgen Schilling, Peter Schwabe, Wolfgang Wieser, *NaCl's* `crypto_box` *in hardware*, in CHES 2015 [**36**] (2015), 81–101. URL: `https://cryptojedi.org/papers/#naclhw`. Citations in this document: §12.1.

[39] Tibor Jager, Jörg Schwenk, Juraj Somorovsky, *Practical invalid curve attacks on TLS-ECDH*, in ESORICS 2015 (2015). URL: `https://www.nds.rub.de/research/publications/ESORICS15/`. Citations in this document: §4.1.

[40] Kaoru Kurosawa (editor), *Advances in cryptology—ASIACRYPT 2007, 13th international conference on the theory and application of cryptology and information security, Kuching, Malaysia, December 2–6, 2007, proceedings*, Lecture Notes in Computer Science, 4833, Springer, 2007. ISBN 978-3-540-76899-9. See [18].

[41] Adam Langley, Andrew Moon, *Implementations of a fast elliptic-curve Digital Signature Algorithm* (2013). URL: `https://github.com/floodyberry/ed25519-donna`. Citations in this document: §12.1.

[42] Eric M. Mahé, Jean-Marie Chauvet, *Fast GPGPU-based elliptic curve scalar multiplication* (2014). URL: `https://eprint.iacr.org/2014/198.pdf`. Citations in this document: §12.1.

[43] Victor S. Miller, *Use of elliptic curves in cryptography*, in Crypto 1985 [**53**] (1986), 417–426. MR 88b:68040. Citations in this document: §3.1.

[44] Nicole Perlroth, Jeff Larson, Scott Shane, *N.S.A. able to foil basic safeguards of privacy on web* (2013). URL: `http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html`. Citations in this document: §1.1.

[45] Josef Pieprzyk (editor), *Advances in cryptology—ASIACRYPT 2008, 14th international conference on the theory and application of cryptology and information security, Melbourne, Australia, December 7–11, 2008*, Lecture Notes in Computer Science, 5350, Springer, 2008. ISBN 978-3-540-89254-0. See [37].

[46] Jim Plusquellic, Ken Mai (editors), *HOST 2010, proceedings of the 2010 IEEE international symposium on hardware-oriented security and trust (HOST), 13–14 June 2010, Anaheim Convention Center, California, USA*, IEEE, 2010. ISBN 978-1-4244-7810-1. See [32].

[47] Bart Preneel (editor), *Progress in cryptology—AFRICACRYPT 2009, second international conference on cryptology in Africa, Gammarth, Tunisia, June 21–25, 2009, proceedings*, Lecture Notes in Computer Science, 5580, Springer, 2009. See [30].

[48] Bart Preneel, Tsuyoshi Takagi (editors), *Cryptographic hardware and embedded systems—CHES 2011, 13th international workshop, Nara, Japan, September 28–October 1, 2011, proceedings*, Lecture Notes in Computer Science, 6917, Springer, 2011. ISBN 978-3-642-23950-2. See [14].

[49] Emmanuel Prouff, Patrick Schaumont (editors), *Cryptographic hardware and embedded systems—CHES 2012—14th international workshop, Leuven, Belgium, September 9–12, 2012, proceedings*, Lecture Notes in Computer Science, 7428, Springer, 2012. ISBN 978-3-642-33026-1. See [20].

[50] Ronald L. Rivest, Martin E. Hellman, John C. Anderson, John W. Lyons, *Responses to NIST's proposal*, Communications of the ACM **35** (1992), 41–54.

URL: https://people.csail.mit.edu/rivest/pubs/RHAL92.pdf. Citations in this document: §1.

[51] Pascal Sasdrich, Tim Güneysu, *Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices*, in ARC 2014 [**35**] (2014), 25–36. URL: https://www.hgi.rub.de/media/sh/veroeffentlichungen/2014/03/25/paper_arc14_curve25519.pdf. Citations in this document: §12.1.

[52] John Wigley, *Removing need for rng in signatures*, message 5gov5d$pad@wapping.ecs.soton.ac.uk posted to sci.crypt (1997). URL: https://groups.google.com/group/sci.crypt/msg/a6da45bcc8939a89. Citations in this document: §2.

[53] Hugh C. Williams (editor), *Advances in cryptology: CRYPTO '85*, Lecture Notes in Computer Science, 218, Springer, Berlin, 1986. ISBN 3–540–16463–4. MR 87d:94002. See [43].

[54] Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin (editors), *Public key cryptography—9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings*, Lecture Notes in Computer Science, 3958, Springer, 2006. ISBN 978-3-540-33851-2. See [9].