Public Comments Received on Draft NIST SP 800-186:
Recommendations for Discrete Logarithm-Based Cryptography:
Elliptic Curve Domain Parameters
(January 29, 2020 deadline)

Bernstein, Dan/Lange, Tanja

From: D. J. Bernstein
Sent: Wednesday, January 29, 2020 11:58 PM
To: fips186-comments
Cc: Tanja Lange
Subject: Comment on FIPS 186

Please see attached PDF for comments from Daniel J. Bernstein and Tanja Lange.

Failures in NIST's ECC standards, part 2

Daniel J. Bernstein[1,2] and Tanja Lange[3]

[1] Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607–7045, USA
[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
djb@cr.yp.to
[3] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org

Abstract. NIST has recently proposed an update to its ECC standards. This update does not adequately account for (1) the long and continuing history of real-world security failures in ECC implementations and (2) analyses showing how next-generation ECC reduces the risk of failures.

1 Introduction

A ZDNet article in October 2019 [25] said "Minerva attack can recover private keys from smart cards, cryptographic libraries". Minerva exploited timing leaks to break the implementations of NIST's ECDSA standard in a FIPS-certified Athena IDProtect card and in four software libraries: libgcrypt, MatrixSSL, JDK, and Crypto++.

Three of these libraries (libgcrypt, MatrixSSL, and Crypto++) also include implementations of EdDSA, specifically Ed25519. But Minerva—despite being introduced in [38] as an attack against "implementations of ECDSA/EdDSA"—did not break any of these EdDSA implementations. See [10] for an analysis of how the differences between ECDSA and EdDSA led directly to EdDSA holding up better than ECDSA against Minerva.

In November 2019, the TPM-FAIL attack [42] announced exploits of ECDSA timings in "trusted platform modules" from ST and Intel, and stated that "the certification has failed to protect the product against an attack that is considered by the protection profile"—unlike the Minerva FIPS-certified target, which turned out to have excluded side-channel attacks from consideration.

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported by the U.S. National Science Foundation under grant 1913167, by the Cisco University Research Program, and by DFG Cluster of Excellence 2092 "CASA: Cyber Security in the Age of Large-Scale Adversaries". "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" (or other funding agencies). Permanent ID of this document: 21bee85333e1b25850ce93dfde2c83f53c38ea4a. Date: 2020.01.29.

In January 2020, CVE 2020-0601 revealed that ECDSA signature verification in Windows 10 allowed forgeries. This turned out to be the result of (1) support for a broad run-time choice of elliptic-curve parameters and (2) inadequate authentication of those parameters; see, e.g., [4] and [5]. For comparison, our paper "Failures in NIST's ECC standards" [15] four years earlier had said that "unnecessary complexity in ECC implementations" creates "ECC security failures", that allowing run-time curve choices causes "obvious damage to implementation simplicity", and that one must "securely authenticate this choice" along with authenticating the ECC key.

Later in January 2020, Aldaya and Brumley [2] announced a single-trace software side-channel attack against the mbedTLS implementation of ECDSA. This attack exploits a leak inside the mbedTLS implementation of modular inversion in ECDSA signing, combined with a seemingly minor bug in surrounding mbedTLS code that was intended to blind the inversion. For comparison, EdDSA signing skips modular inversion.

Some ECC software has been formally verified, guaranteeing that the software works correctly and is immune to broad classes of timing attacks. This body of work has made some progress for NIST's ECC standards but much more progress for next-generation ECC: see, e.g., [49]. We had already commented in [15] that "this work is targeting X25519 implementations precisely because those implementations are so simple", and that "unnecessary complexity ... interferes with verification".

1.1. Moving NIST ECC beyond NSA ECC. On 31 October 2019, four weeks after the Minerva attack was announced, NIST published draft FIPS 186-5 [45], draft SP 800-186 [23], and a formal "Request for Comments" [46] regarding these drafts. If these drafts are adopted then they will update some of NIST's ECC standards.

The release of these documents was an opportunity for NIST to quote Rivest's comment

> The poor user is given enough rope with which to hang himself—something a standard should not do.

from 1992 [51] regarding the NIST/NSA "DSA" proposal; to cite the Sony PS3 hanging itself [22]; to admit that this was a mistake in DSA; to admit that this was also a mistake in ECDSA, which copied the relevant technical details from DSA; and, more broadly, to take responsibility for the way that NIST's ECC standards have predictably produced real-world security failures. Explicitly recognizing the security impact of

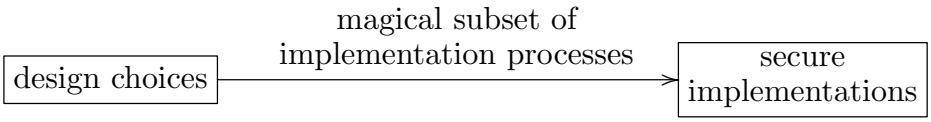
$$\boxed{\text{design choices}} \xrightarrow{\text{implementation processes}} \boxed{\text{implementations}}$$

would have helped motivate the adoption of next-generation ECC in updated standards, and would have helped guide decisions regarding the details.

Unfortunately, the Request for Comments categorically denies responsibility for any security failures:

> NIST is not aware of any vulnerabilities to attacks on these curves when they are implemented correctly and used as described in NIST standards and guidelines.

Minerva and older security failures are not cited and are not even acknowledged. There is no indication that predictable implementation security failures were considered as a design issue in draft FIPS 186-5 and draft SP 800-186. In other words, NIST appears to have limited its scope to



without even asking whether the "poor user is given enough rope with which to hang himself".

The Request for Comments goes on to state the following description of next-generation ECC:

> Advances in the understanding of elliptic curves within the cryptographic community have led to the development of new elliptic curves and algorithms, and their designers claim that they offer better performance and are easier to implement in a secure manner than previous versions.

The Request for Comments does not cite any of the literature demonstrating the performance benefits and security benefits of next-generation ECC compared to NIST's ECC standards. Instead it downgrades the benefits to a mere "claim" by the "designers". Furthermore, the description of security benefits is limited to the *ease of secure implementation*; there is no mention of the likelihood and consequences of insecure implementation.

1.2. Transparency and auditability. NIST made a series of changes from FIPS 186-4 to draft FIPS 186-5 (plus draft SP 800-186). Standard editing tools make it easy to keep logs of all changes, and if NIST had posted such logs then readers familiar with FIPS 186-4 would have been able to efficiently see all of the changes, rather than painfully comparing the two documents.

Unfortunately, NIST does not seem to have made any such logs available. There is a half-page list of revisions at the end of draft FIPS 186-5, but a reader seeing, e.g., "Aonstructing [sic] primes with congruence conditions mod 8 are allowed" has to figure out which lines have been edited and what the edits were to those lines.

Presumably NIST also had a *rationale* for each of its changes. NIST could have kept a log of this rationale along with the log of each change. This would have taken some work to type but would have saved far more work for reviewers trying to figure out *why* NIST did what it did. This in turn would have helped reviewers identify and correct mistakes. Again no such logs are available.

NIST has posted [44] various submissions that it received in response to a previous request for comments regarding FIPS 186-4. For example, a submission from BSI claimed, incorrectly, that some of the Brainpool curves had been

"standardised in RFC 5639". Did this incorrect information regarding previous standardization contribute to NIST's decision to include the Brainpool curves in draft SP 800-186? If NIST had posted a rationale for each of its changes then we would already have the answer.

More broadly, it is not clear how NIST handled the inputs it received. Consider, for example, the following claim in FIPS 186-4: "For efficiency reasons, it is desirable to take the cofactor to be as small as possible." This claim was originally from [1], distributed by NIST and reportedly written by NSA. Our own input to NIST disputed this claim.[4] The same claim turns out to appear in draft SP 800-186. Did NIST not take the time to evaluate our input? Or did NIST evaluate our input and decide to continue endorsing NSA's claim for some reason? If so, what is that reason?

Our own review of these NIST documents has not been comprehensive. Our lack of comment on any particular aspects of the documents should not be taken as endorsing those aspects.

1.3. Previous analyses of NIST's ECC standards. We incorporate our January 2016 paper "Failures in NIST's ECC standards" [15] by reference into these comments to NIST. "Incorporate by reference" means that we ask NIST to read that paper in the context of the "Request for Comments on FIPS 186-5 and SP 800-186", just as if we were repeating the contents as comments to NIST now.[5]

That paper incorporated the following further documents by reference:

- "High-speed high-security signatures" [13].
- "How to design an elliptic-curve signature system" [7].
- "EdDSA for more curves" [14].
- "Break a dozen secret keys, get a million more for free" [9].
- "How to manipulate curve standards: a white paper for the black hat" [11].
- "SafeCurves: choosing safe curves for elliptic-curve cryptography" [16].
- "Things that use Curve25519" [20].
- "Things that use Ed25519" [21].

We incorporate the latest versions of these documents by reference here. Note that the web site [16], the web page [20], and the web page [21] were updated after [15].

[4] "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."

[5] We had also sent an earlier version of the paper as comments to NIST in December 2015 regarding a previous request for comments. The final paper has extra references, clarifications, etc.

2 Nonce attacks

Write r for the secret nonce used in ElGamal signatures, Schnorr signatures, DSA, ECDSA, EdDSA, etc. Write n for the prime group order. There have been several lines of attack exploiting visible non-randomness in $r \bmod n$:

- ElGamal pointed out in [29] that it is disastrous for a signer to reuse r for another signature. This reuse is what happened in [22] and in various Bitcoin attack papers cited in [19].
- Visible biases in $r \bmod n$ enable various attacks surveyed in, e.g., [19] and [10]. For example, Bleichenbacher broke the original version of DSA using a "workfactor of 2^{64}" and "2^{22} known signatures", according to [43, page 72]. The problem is that DSA chose $n \approx 0.7 \cdot 2^{160}$ and generated r as a uniform random 160-bit integer; $r \bmod n$ is then biased towards small values.
- Even if $r \bmod n$ is generated uniformly at random, side-channel attacks can leak bits of $r \bmod n$, creating visible biases and reenabling the attacks. This is what happened in, e.g., the recent Minerva and TPM-FAIL timing attacks.

Regarding the third problem, one can blame implementors for leaking information through timing, but it is also useful to design signature systems so that implementors are less likely to leak information through timing. See [7] for a detailed comparison of the Ed25519 design to the ECDSA-P-256 design from this perspective. Minerva illustrates the success of this approach: case studies #2 through #9 in [10] are eight crypto libraries that claim to use constant-time Ed25519 implementations, while ECDSA-P-256 implementations are less likely to be designed to be constant-time.

Regarding the second problem, two obvious defenses are

- to choose n very close to a power of 2 (as in Curve25519 and P-256, but not the original DSA and not Brainpool) and
- to obtain r as a double-length hash.

A double-length hash r makes it more likely that implementors will use the entire hash—why would an implementor bother reducing r modulo n if applications have not asked for the extra performance?—and then a timing leak of the top bits of r does not compromise security. Case study #1 in [10] is a variable-time Ed25519 implementation that was saved from Minerva by the double-length r in this way. This scenario had been described in [8] five years earlier, illustrating the predictability of implementation problems.

Implementors might ignore the specification of r and substitute their own "random" r, perhaps a biased r. The analysis in [19] shows that visible biases occurred in thousands of Bitcoin signatures. One can blame implementors for this, but it is also useful to specify an easily testable r-generation process, increasing the chance that deviations from the process will be caught and fixed.

The remaining problem is the possibility of r being reused. This is not necessarily the fault of the signature implementor: sometimes RNGs fail catastrophically. One can try to dismiss this by saying "there is no hope of security when

the RNG fails", but sometimes a key is generated on a master device with a good RNG and then a signature is generated on a slave device with a bad RNG.

2.1. Hashing output from a stateful PRNG. Starting with (presumably) good randomness from a key-generation device, one can build a PRNG running in the signing device. Use, e.g., AES-256 or SHA-512 to expand a secret PRNG seed into a block of randomness; use part of the block as the hash input to create r, and a separate part of the block to overwrite the PRNG seed for the next signature.

The problem with this approach is that it is stateful. This limits deployability in some environments. More importantly, it raises security concerns regarding the possibility of state updates failing (e.g., because virtual machines are restarted). Langley [39] described stateful signatures as a "huge foot-cannon". Stateful signatures are also somewhat more difficult to test than stateless signatures.

From an engineering perspective, rather than designing a signature system with a PRNG and then another signature system with a PRNG, one should design a central device PRNG and have both signature systems use it. This factorization relabels the state problems as being the responsibility of the central device PRNG; however, it does not make the problems disappear. The security system remains stateful, and failed state updates will compromise security.

2.2. Hashing the message and a secret seed. As pointed out by Barwood [3] and Wigley [56], a stateless signing system can avoid reusing r:

- Include the message being signed as a hash input. The hash output r will not repeat unless the message repeats—and in that case the entire signature will repeat, so the repetition of r is not a problem.
- Include a secret seed as a hash input. Each hash output r is then secret, as required.

From a mathematical security perspective, what one needs here is that r is the output of a strong PRF applied to the message. Standard hash functions appear to be massive overkill as PRFs, but it is simplest to reuse the hash function used elsewhere in signing, and it is difficult to find verifiable examples where message-hashing time inside signing is a cost issue for the ultimate user.

2.3. Hashing further inputs. More generally, one can take r as a hash of a string that *begins* with the secret seed and the message. The rest of the string consists of additional hash inputs.

Examples of additional inputs mentioned in [7] include PRNG output and a counter of the number of messages signed. These inputs are stateful, limiting deployability and complicating tests, but failed state updates (or other PRNG failures) no longer produce catastrophic failures: as above, the secrecy of the seed ensures the secrecy of r, and r will not repeat unless the message repeats. Similarly, on a device with a supposedly stateless RNG, including the RNG output as an additional hash input will not break the system if the RNG gets stuck.

Of course, in environments that support the writable state for a PRNG, one can merge that state with the state used for the initial seed, using some cipher or hash output to overwrite the seed as before. One can also merge this update with the message processing: hash the seed together with the message, use part of the output for r, and use another part of the output to overwrite the seed. As a concrete example, for Ed25519, one can hash a 32-byte seed together with the message using SHA-512, take the first 32 bytes of output as a new seed, and hash the other 32 bytes of output to generate a 64-byte r.

2.4. Evaluating security tradeoffs between options. There appears to be no dispute that r should be computed as a hash.

Having the hash input *include* the message being signed, along with a secret derived from key-generation randomness, is essential for maintaining security when state updates (including the device RNG) fail. This is not very complicated, and it is justified by its undisputed ability to stop real-world security failures.

Having the hash input *limited to* a long-term seed and the message being signed has advantages and disadvantages. The main objection is that this makes the long-term seed an attractive target of, e.g., power attacks in environments where power consumption is visible to the attacker. See, e.g., [**52**]. The bigger picture is that there is an extensive literature

- breaking a wide range of cryptographic computations via power attacks, electromagnetic attacks, etc., and
- designing countermeasures to protect against these attacks.

There are, for example, papers advertising protections for SHA-2, and papers advertising lower-cost protections for SHA-3. General protection strategies include randomization and state updates. In particular, one can try to stop attacks against the long-term seed in signatures by including separate randomness in the hash or updating the seed, as in Section 2.3. On the other hand, this complicates testing, and it needs to be accompanied by further complications to protect (e.g.) arithmetic modulo n. In environments that have other protections against side-channel attacks, the simplicity of Section 2.2 is preferable. Similar comments apply to fault attacks.

This simplicity argument justifies standardizing an easily testable stateless signature-generation method that hashes just two inputs: a long-term seed and the message being signed. This does not prevent subsequent standardization of an alternative signing procedure that uses randomization and/or state updates as countermeasures against side-channel attacks. Note that arbitrary variations in the signer's computation of r are compatible with the specified procedure for signature verification.

2.5. Using the message input to promote good practices. Section 7.8.3 of draft FIPS 186-5, "Differences between EdDSA and HashEdDSA", complains about EdDSA's requirement to have a long message "either buffered or read from storage twice" during signing (once during the generation of r and once later). It is correct to conclude that HashEdDSA "will have better performance" since

it allows long messages to be streamed through signing. This is what prompted the development of HashEdDSA in the first place.

However, this section of draft FIPS 186-5 fails to note the *security* impact of applications signing long messages:

- Signatures on long messages put verifiers under performance pressure to support streaming interfaces.
- These streaming interfaces generally allow attackers to pass forged message prefixes directly to unwitting applications, before the verification procedures have been invoked.

The conclusion of [**14**, "Security notes on prehashing"] is that it is "safest for protocol designers to split long messages into short messages to be signed; this splitting also eliminates the storage issue". Of course, each signed message needs to explicitly state enough of its context to avoid being taken out of context, but this is true whether or not messages are split.

We recommend adding the following text to the section:

> EdDSA **should** be used in preference to HashEdDSA, except in applications that cannot afford EdDSA.

Our rationale for this text is as follows. First, some applications will have no problem buffering and hashing messages for EdDSA, and in these applications EdDSA is preferable to HashEdDSA for two reasons:

- EdDSA is collision-resilient while HashEdDSA is not.[6]
- EdDSA is slightly simpler than HashEdDSA.

Second, some applications with limited buffer sizes will nevertheless be able to use EdDSA by splitting long messages into short messages to be signed. The recommendation to use EdDSA encourages these applications to limit message lengths being signed, and helps discourage dangerous streaming interfaces. Third, applications that really cannot afford EdDSA are better with HashEdDSA than with nothing (or with ECDSA). Fourth, we have written "**should**" rather than "**shall**" to accommodate the possibility of other reasons for HashEdDSA. The text does not prohibit HashEdDSA; it merely specifies EdDSA as the default and asks for documentation of deviations from the default.

[6] The current text in draft FIPS 186-5 disputes the value of collision resilience: it claims that "the risk of collisions using either SHA-512 or SHAKE256 is considered negligible". Who exactly considers the risk to be "negligible"? No citations are given to the relevant cryptanalytic literature. NIST's call for SHA-3 submissions in 2007 used very different language: "Although there is no specific reason to believe that a practical attack on any of the SHA-2 family of hash functions is imminent, a successful collision attack on an algorithm in the SHA-2 family could have catastrophic effects for digital signatures. NIST has decided that it is prudent to develop a new hash algorithm to augment and revise FIPS 180-2." How much of the cryptanalytic community since then has been studying SHA-512, rather than splitting effort across >100 candidates for a series of competitions in symmetric cryptography?

3 Selecting fields and curves

The selection of fields and curves in SP 800-186 is different from the selection in FIPS 186-4. The changes are clear but not all of the changes have clear rationales.

3.1. Removing multiplicative groups. One change in SP 800-186 is that now only *elliptic* curves are allowed: multiplicative groups (DSA) are removed. This change has a clearly stated rationale:

> Industry adoption of DSA was limited, and subsequent versions of FIPS 186 added other signature algorithms that are in broad use within products and protocols, including ECDSA and RSA-based signature algorithms. At this time, NIST is not aware of any applications where DSA is currently broadly used. Furthermore, recent academic analysis observed that implementations of DSA may be vulnerable to attacks if domain parameters are not properly generated. These parameters are not commonly verified before use.

This rationale has two components: one regarding the unpopularity of DSA, and another regarding the vulnerability of DSA "if domain parameters are not properly generated".

It is interesting to observe that these arguments are not tied to multiplicative groups. The unpopularity of DSA seems to be connected to DSA's inefficiency, which in turn has been created by attacks exploiting the structure of multiplicative groups,[7] but there are also inefficient elliptic-curve groups with similarly limited industry adoption.

As for improper generation of domain parameters: Elliptic curves are *also* vulnerable to attacks if domain parameters are not properly generated. This was spectacularly illustrated by CVE 2020-0601: Microsoft added code to support many different elliptic curves, and a bug in this code turned out to allow forged signatures using forged domain parameters. Properly tying domain parameters to certificates would have prevented this particular ECDSA disaster, but in general elliptic-curve parameters are more complicated to verify than DSA parameters. This component of NIST's rationale does not seem to be a valid argument for elliptic curves compared to multiplicative groups; instead it is an argument for standardizing a limited selection of well-vetted parameters.

3.2. Deprecating curves over binary fields. Another change in SP 800-186 is that elliptic curves over binary fields are deprecated. This also has a clearly stated rationale:

> Finally, based on feedback received on the adoption of the current elliptic curve standards, the draft standards deprecate curves over binary fields due to their limited use by industry.

[7] A separate issue is that these attacks are very complicated, with a long history of improvements and many unexplored avenues for further improvements. We would recommend against multiplicative groups for this reason even if multiplicative groups were as fast as elliptic-curve groups.

Again this is a statement regarding the level of deployment.

3.3. Evaluation of adoption of added curves. The draft SP 800-186 includes Edwards coordinates for Curve25519 and Curve448, as stated in the Request for Comments. It also includes Weierstrass coordinates for Curve25519 and Curve448 to use in ECDSA. Not mentioned in the Request for Comments is that the draft also allows Brainpool curves "to be used for interoperability reasons".

Given the prominent role that "industry adoption"/"use by industry" has played in NIST's decisions regarding multiplicative groups and binary fields, presumably it is also important to understand the levels of adoption of Curve25519, Curve448, and the Brainpool curves. For example, if the Brainpool curves are less popular than DSA and binary curves, then why are the Brainpool curves being added?

From this perspective, it is troubling to see, within the standardization process, misinformation regarding levels of adoption. For example:

- Draft SP 800-186 claims that "The most widely used curves are usually expressed in short-Weierstrass format". The NIST curves are usually expressed in short-Weierstrass format, but no evidence is provided for the implicit claim that the NIST curves are "the most widely used curves" at the time of writing. The next sentence refers to other curves as having "garnered academic interest" without mentioning (e.g.) more than a billion users of WhatsApp. See generally [**20**] and [**21**] regarding usage of Curve25519.
- Draft SP 800-186 refers to "other standards-setting organizations, such as the Crypto Forum Research Group (CFRG) of the IETF". IETF is a standards-setting organization but CFRG is not.
- As noted in Section 1, BSI claimed, as part of official input to NIST, that some Brainpool curves were "standardised in RFC 5639". This claim is false. RFC 5639, like RFC 7748, is an Informational RFC, not a standards-track document. Furthermore, the review process for RFC 5639 was minimal—whereas RFC 7748 was the conclusion of an open two-year CFRG analysis occupying thousands of email messages, and says "This RFC represents the consensus of the Crypto Forum Research Group of the Internet Research Task Force (IRTF)." TLS 1.3, which *is* on the IETF standards track, allows the RFC 7748 curves and not the RFC 5639 curves.

More broadly, the lack of a stated rationale and supporting data regarding the addition of curves makes it unnecessarily difficult to evaluate and comment upon NIST's choices.

3.4. Evaluation of verification of added curves. Given NIST's statement that DSA "parameters are not commonly verified before use", it is also troubling to see that the 160-bit, 192-bit, 224-bit, 256-bit, 320-bit, and 384-bit Brainpool curves published in October 2005 were *not* generated by the Brainpool curve-generation procedure that was published in the same document and that was claimed to have been used to generate those curves. This observation appears to

Core	Xeon	Microarchitectures	P-256	Curve25519
1		Nehalem (2008), Westmere (2010)		226872
2	v1	Sandy Bridge (2011)	311000	159068
3	v2	Ivy Bridge (2012)	313000	157192
4	v3	Haswell (2013)	228000	156052
5	v4	Broadwell (2014)	177000	120000
6	v5	Skylake (2015)	171000	116000
7	v6	Kaby Lake (2016)	171000	116000

Table 3.6. Variable-base-point scalar-multiplication times, in cycles (smaller is better), for P-256 and Curve25519 on various Intel Core microarchitecture generations. The "Core" column lists the generation number in Intel Core CPU numbers. The "Xeon" column lists the generation number in Intel Xeon CPU numbers. The year listed is the year when the microarchitecture was introduced; e.g., Intel introduced its first Haswell CPUs in June 2013, so Haswell is listed as "2013" in the table. For P-256: 311000, 313000, 228000, 177000, 171000, and 171000 are measured by OpenSSL 1.1.1; 228000 is an improvement over the 291000 from [35]. For Curve25519: 226872 is reported in [13]; 159068, 157192, and 156052 are measurements from SUPERCOP of the software from [24]; 120000, 116000, and 116000 are measured by OpenSSL 1.1.1. Measurements are on an Intel Xeon i3-2310M, Intel Xeon E3-1275 v2, Intel Xeon E3-1220 v3, Intel Xeon E5-2609 v4, Intel Xeon E3-1220 v5, and Intel Xeon E3-1220 v6 respectively.

have been first published in [11] ten years later, along with further observations regarding the Brainpool curves.

Furthermore, like FIPS 186-4, draft SP 800-186 requires that ECDSA curves be generated using SHA-2 or SHA-3. To justify making an exception for the NSA curves, [23, Footnote 1] says that "SHA-1 was considered secure at the time of generation" of the NSA curves. This exception does not appear to apply to the Brainpool curves: Wang's attack against SHA-1 appeared at Crypto 2005 in August 2005 and was already publicly announced, e.g., in Schneier's blog post "SHA-1 broken" in February 2005.

3.5. Evaluation of implementation properties of added curves. It is also troubling to see misinformation regarding implementation properties such as simplicity and efficiency: these properties have an impact upon adoption, and adoption plays a role in NIST's decisions.

Consider, e.g., the submission from Adalier to NIST included in [44]. This submission claims that "recent high performance implementations of ECDSA P-256 (OpenSSL—S. Gueron, taraEcCRYPT(tm)—M. Adalier) show that ECDSA can be implemented as fast and securely as the other schemes."

What Gueron and Krasnov actually reported in [35] for the software that they contributed to OpenSSL was 291000 Haswell cycles for P-256 variable-base-point scalar multiplication. For comparison, [13] had already reported just 226872 Westmere cycles for Curve25519 variable-base-point scalar multiplication. Westmere is three processor generations older than Haswell; see generally Table 3.6.

P-256 implementations improved after [35]. For example, OpenSSL 1.1.1 takes 228000 Haswell cycles or 171000 Skylake cycles for P-256 variable-base-point scalar multiplication. However, for Curve25519 variable-base-point scalar multiplication, the software from Chou [24] (optimized for an older microarchitecture, Sandy Bridge) takes 156052 Haswell cycles or 135157 Skylake cycles, and newer software in OpenSSL takes just 116000 Skylake cycles.

More broadly, Curve25519 implementations have consistently outperformed P-256 implementations ever since the introduction of Curve25519. See, e.g., [6], [31], [26], [13], [17], [40], [41], [53], [24], [27], and [37].

These performance comparisons between ECDH-P-256 and X25519 illustrate the slowness of P-256 field operations and P-256 curve operations. It is safe to predict that similar implementation effort will produce similar cost ratios between ECDSA-P-256 and Ed25519, although not identical cost ratios (e.g., ECDSA has inversions that are not needed in EdDSA). Finally, there does not appear to be any publicly verifiable evidence that "taraEcCRYPT" outperforms OpenSSL.

Regarding implementation security, the original software from Gueron and Krasnov used a "scatter-gather" table-lookup method that is a few percent faster than the table-scanning method used in [13] but that also violates the security rules stated in [6] and [13]. Scatter-gather lookups were later exploited by the "CacheBleed" RSA attack [57], and presumably are also exploitable in the ECDSA context.

It is of course *possible* to write variable-time Ed25519 software, and in particular scatter-gather Ed25519 software, which presumably would be exploitable in the same way as scatter-gather ECDSA-P-256 software. But this does not end the risk analysis. Because Curve25519 is faster than P-256, Curve25519 implementors are under less pressure than P-256 implementors to apply minor optimizations such as scatter-gather lookups. Furthermore, in many applications, a simple ladder provides acceptable speed for Ed25519 key generation and signing, and the implementor does not need table lookups at all—whereas a P-256 ladder is slower and more likely to be rejected for speed reasons. To summarize, insecure Ed25519 implementations are *possible*, but insecure ECDSA-P-256 implementations (such as the implementations exploited by Minerva and TPM-FAIL) are *more likely*.

The same issues are even more severe for ECDSA with the 256-bit Brainpool curve, which appears to be much slower than Ed25519. See also the Brainpool-specific implementation problems exploited in [55] related to the primes not being close to powers of 2.

3.7. Interoperability impact. In [15] we explained how a profusion of standardized curves damages implementation simplicity and creates interoperability problems. We concluded the analysis as follows:

> 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?"

It is worrisome to see so many curves in draft SP 800-186 without a clear explanation, for each curve, of how the benefits outweigh the costs.

4 Further comments

See Appendices A, B, C, D, and E.

We repeat a warning from Section 1: "Our own review of these NIST documents has not been comprehensive. Our lack of comment on any particular aspects of the documents should not be taken as endorsing those aspects."

References

[1] — (no editor), *Recommended elliptic curves for federal government use* (1999). URL: https://web.archive.org/web/20080917124637/http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf. Citations in this document: §1.2.

[2] Alejandro Cabrera Aldaya, Billy Bob Brumley, *When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA* (2020). URL: https://eprint.iacr.org/2020/055. Citations in this document: §1.

[3] 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: §2.2.

[4] Tal Be'ery, *Win10 crypto vulnerability: cheating in elliptic curve billiards 2* (2020). URL: https://medium.com/zengo/win10-crypto-vulnerability-cheating-in-elliptic-curve-billiards-2-69b45f2dcab6. Citations in this document: §1.

[5] Tal Be'ery, *CurveBall's additional twist: the certificate comparison bug* (2020). URL: https://medium.com/zengo/curveballs-additional-twist-the-certificate-comparison-bug-2698aea445b5. Citations in this document: §1.

[6] Daniel J. Bernstein, *Curve25519: new Diffie-Hellman speed records*, in PKC 2006 [**58**] (2006), 207–228. URL: https://cr.yp.to/papers.html#curve25519. Citations in this document: §3.5, §3.5.

[7] Daniel J. Bernstein, *How to design an elliptic-curve signature system* (2014). URL: https://blog.cr.yp.to/20140323-ecdsa.html. Citations in this document: §1.3, §2, §2.3.

[8] Daniel J. Bernstein, *Re: Mishandling twist attacks* (2014). URL: https://mailarchive.ietf.org/arch/msg/cfrg/8z3ZcujGRxFSGEBI-uE7C1tjw4c. Citations in this document: §2.

[9] Daniel J. Bernstein, *Break a dozen secret keys, get a million more for free* (2015). URL: https://blog.cr.yp.to/20151120-batchattacks.html. Citations in this document: §1.3.

[10] Daniel J. Bernstein, *Why EdDSA held up better than ECDSA against Minerva* (2019). URL: https://blog.cr.yp.to/20191024-eddsa.html. Citations in this document: §1, §2, §2, §2.

[11] 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: §1.3, §3.4.

[12] 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 [**13**]. URL: https://eprint.iacr.org/2011/368.

[13] 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 [**12**]. URL: https://eprint.iacr.org/2011/368. Citations in this document: §1.3, §3.5, §3.6, §3.6, §3.5, §3.5, §3.5.

[14] 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: §1.3, §2.5.

[15] Daniel J. Bernstein, Tanja Lange, *Failures in NIST's ECC standards* (2016). URL: https://cr.yp.to/papers.html#nistecc. Citations in this document: §1, §1, §1.3, §1.3, §3.7.

[16] Daniel J. Bernstein, Tanja Lange, *SafeCurves: choosing safe curves for elliptic-curve cryptography* (2017). URL: https://safecurves.cr.yp.to. Citations in this document: §1.3, §1.3.

[17] Daniel J. Bernstein, Peter Schwabe, *NEON crypto*, in CHES 2012 [**50**] (2012), 320–339. URL: https://cr.yp.to/papers.html#neoncrypto. Citations in this document: §3.5.

[18] G. R. Blakley, David Chaum (editors), *Advances in cryptology, proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19–22, 1984, proceedings*, Lecture Notes in Computer Science, 196, Springer, Berlin, 1985. ISBN 3-540-15658-5. MR 86j:94003. See [28].

[19] Joachim Breitner, Nadia Heninger, *Biased nonce sense: lattice attacks against weak ECDSA signatures in cryptocurrencies*, in FC 2019 [**34**] (2019), 3–20. URL: https://eprint.iacr.org/2019/023. Citations in this document: §2, §2, §2.

[20] Nicolai Brown, *Things that use Curve25519* (2020). URL: https://ianix.com/pub/curve25519-deployment.html. Citations in this document: §1.3, §1.3, §3.3.

[21] Nicolai Brown, *Things that use Ed25519* (2020). URL: https://ianix.com/pub/ed25519-deployment.html. Citations in this document: §1.3, §1.3, §3.3.

[22] "Bushing", Hector Martin "marcan" Cantero, Segher Boessenkool, Sven Peter, *PS3 epic fail* (2010). URL: https://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf. Citations in this document: §1.1, §2.

[23] Lily Chen, Dustin Moody, Andrew Regenscheid, Karen Randall, *SP 800-186 (draft): recommendations for discrete logarithm-based cryptography: elliptic curve domain parameters* (2019). URL: https://csrc.nist.gov/publications/detail/sp/800-186/draft. Citations in this document: §1.1, §3.4, §B, §B, §B, §B, §B, §B, §B.

[24] Tung Chou, *Sandy2x: new Curve25519 speed records*, in SAC 2015 (2015). URL: https://tungchou.github.io/papers/sandy2x.pdf. Citations in this document: §3.6, §3.6, §3.5, §3.5.

[25] Catalin Cimpanu, *Minerva attack can recover private keys from smart cards, cryptographic libraries* (2019). URL: https://www.zdnet.com/article/minerva-attack-can-recover-private-keys-from-smart-cards-cryptographic-libraries/. Citations in this document: §1.

[26] 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: §3.5.

[27] 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: https://link.springer.com/article/10.1007/s10623-015-0087-1/fulltext.html. Citations in this document: §3.5.

[28] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, in Crypto '84 [**18**] (1985), 10–18; see also newer version [**29**]. MR 87b:94037.

[29] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), 469–472; see also older version [**28**]. ISSN 0018-9448. MR 86j:94045. Citations in this document: §2.

[30] Pierrick Gaudry, *Variants of the Montgomery form based on Theta functions* (2006); see also newer version [**31**]. URL: https://cr.yp.to/bib/2006/gaudry-toronto.pdf.

[31] Pierrick Gaudry, *Fast genus 2 arithmetic based on Theta functions*, Journal of Mathematical Cryptology **1** (2007), 243–265; see also older version [**30**]. URL: https://hal.inria.fr/inria-00000625/file/arithKsurf.pdf. Citations in this document: §3.5.

[32] Benedikt Gierlichs, Axel Y. Poschmann (editors), *Cryptographic hardware and embedded systems—CHES 2016—18th international conference, Santa Barbara, CA, USA, August 17–19, 2016, proceedings*, Lecture Notes in Computer Science, 9813, Springer, 2016. ISBN 978-3-662-53139-6. See [57].

[33] 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 [53].

[34] Ian Goldberg, Tyler Moore (editors), *Financial cryptography and data security—23rd international conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, revised selected papers*, Lecture Notes in Computer Science, 11598, Springer, 2019. See [19].

[35] Shay Gueron, Vlad Krasnov, *Fast prime field elliptic curve cryptography with 256 bit primes*, Journal of Cryptographic Engineering **5** (2013), 141–151. URL: https://eprint.iacr.org/2013/816. Citations in this document: §3.5, §3.6, §3.6, §3.5.

[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 [37].

[37] 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: §3.5.

[38] Jan Jancar, Petr Svenda, Vladimir Sedlacek, *Minerva* (2019). URL: https://minerva.crocs.fi.muni.cz. Citations in this document: §1.

[39] Adam Langley, *Hash based signatures* (2013). URL: https://www.imperialviolet.org/2013/07/18/hashsig.html. Citations in this document: §2.1.

[40] 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: §3.5.

[41] 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: §3.5.

[42] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, Nadia Heninger, *TPM-FAIL: TPM meets timing and lattice attacks*, in USENIX 2020, to appear (2019). URL: https://tpm.fail/. Citations in this document: §1.

[43] National Institute of Standards and Technology (NIST), *FIPS 186-2 (+Change Notice): Digital signature standard (DSS)* (2001). URL: https://csrc.nist.gov/CSRC/media/Publications/fips/186/2/archive/2001-10-05/documents/fips186-2-change1.pdf. Citations in this document: §2.

[44] National Institute of Standards and Technology (NIST) (editor), *Public comments received on FIPS 186-4: digital signature standard (DSS)* (2015). URL: https://csrc.nist.gov/csrc/media/publications/fips/186/4/final/documents/comments-received-fips186-4-december-2015.pdf. Citations in this document: §1.2, §3.5.

[45] National Institute of Standards and Technology (NIST), *FIPS 186-5 (draft): Digital signature standard (DSS)* (2019). URL: https://csrc.nist.gov/publications/detail/fips/186/5/draft. Citations in this document: §1.1, §A, §A, §A, §A, §A, §A, §A, §A, §A.

[46] National Institute of Standards and Technology (NIST), *Request for comments on FIPS 186-5 and SP 800-186* (2019). URL: https://www.federalregister.gov/documents/2019/10/31/2019-23742/request-for-comments-on-fips-186-5-and-sp-800-186. Citations in this document: §1.1.

[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 [26].

[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 [12].

[49] Jonathan Protzenko, Bryan Parno, Aymeric Fromherz, Chris Hawblitzel, Marina Polubelova, Karthikeyan Bhargavan, Benjamin Beurdouche, Joonwon Choi, Antoine Delignat-Lavaud, Cedric Fournet, Natalia Kulatova, Tahina Ramananandro, Aseem Rastogi, Nikhil Swamy, Christoph Wintersteiger, Santiago Zanella-Beguelin, *EverCrypt: a fast, verified, cross-platform cryptographic provider*, in IEEE S&P 2020, to appear (2019). URL: https://eprint.iacr.org/2019/757. Citations in this document: §1.

[50] 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 [17].

[51] 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: §1.1.

[52] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, Ruggero Susella, *Breaking Ed25519 in WolfSSL*, in CT-RSA 2018 [**54**] (2017), 1–20. URL: https://eprint.iacr.org/2017/985. Citations in this document: §2.4.

[53] Pascal Sasdrich, Tim Güneysu, *Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices*, in ARC 2014 [**33**] (2014), 25–36. URL: https://www.hgi.rub.de/media/sh/veroeffentlichungen/2014/03/25/paper_arc14_curve25519.pdf. Citations in this document: §3.5.

[54] Nigel P. Smart (editor), *Topics in cryptology—CT-RSA 2018—the Cryptographers' track at the RSA Conference 2018, San Francisco, CA, USA, April 16–20, 2018, proceedings*, Lecture Notes in Computer Science, 10808, Springer, 2018. ISBN 978-3-319-76952-3. See [52].

[55] Mathy Vanhoef, Eyal Ronen, *Dragonblood: analyzing the Dragonfly handshake of WPA3 and EAP-pwd*, in IEEE S&P 2020, to appear (2019). URL: https://eprint.iacr.org/2019/383.pdf. Citations in this document: §3.5.

[56] 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: §2.2.

[57] Yuval Yarom, Daniel Genkin, Nadia Heninger, *CacheBleed: a timing attack on OpenSSL constant time RSA*, in CHES 2016 [**32**] (2016), 346–367. Citations in this document: §3.5.

[58] 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 [6].

A Further comments from author 1 on draft FIPS 186-5

Given the level of interest in post-quantum cryptography, it would be useful to say explicitly that this standard is not designed to protect against quantum computers.

Algorithms expressed in Python, Sage, hacspec, etc. would be more useful than algorithms expressed in pseudocode.

[**45**, page 3] defines "hash function" to map a "bit string of arbitrary length" to a "fixed length bit string". But FIPS 180-4 says that SHA-256 hashes only a message with "a length of ℓ bits, where $0 \le \ell < 2^{64}$"; this is not an "arbitrary length".

[**45**, page 9, Figure 1] indicates that signing takes only a hash of a message, rather than the message itself: there is a "hash function" that provides a "mes-

sage digest" to "signature generation". This is wrong for EdDSA.[8] There is also an important difference between

- "signature generation" functions that rely on the separate "hash function" for security—these functions are broken if the attacker has the ability to provide fake message digests for signing—and
- "signature generation" functions that are secure by themselves.

Labeling functions of the first type as "signature generation" functions, and exposing those functions to the user, will mislead implementors and users into thinking that the functions meet the standard definition of signature security in the literature.

The same problem also occurs in the narrative description of signature generation in [45, Section 3.2]. This description says that a message digest is generated "prior to the generation of a digital signature".

[45, Section 4] says that it "no longer approves DSA for digital signature generation" but says that DSA "may be used to verify signatures generated prior to the implementation date of this standard". This appears to mean that it is not compliant with the standard to verify DSA signatures generated after the implementation date of the standard.[9] But how is a DSA verifier supposed to know that it is in this situation? If this text is approved as a standard, and a DSA signer (e.g., software built for compliance with the previous standard) then generates a signature, and a DSA verifier checks the signature, is the verifier violating this standard?

"PCKS" is a typo for "PKCS".

[45, Section 7.1] claims that "attacks such as side-channel attacks and fault attacks" are of "particular concern" for "deterministic signature schemes". This is not an accurate summary of what is known on this topic. See Section 2.

[45, Section 7.2] switches from notation "$\bullet$" to notation "$*$", making the reader wonder whether the switch is between two different types of multiplication. The same section refers to an "encoding of $\mathrm{GF}(p)$" but never defines this encoding. More broadly, if the goal of this section is to specify the same details as in RFC 8032, then why does this section not simply cite RFC 8032 and say that encodings are defined there? Having each standardization organization write its own specification is a denial-of-service attack against reviewers.

[45, Section 7.6] violates the standard definition of signatures in the literature as producing a signature from a private key and a message (and whatever random bits are consumed by the signing algorithm). The public key is not an input in

[8] Mathematically, since SHA-256 is defined only for messages shorter than 2^{64} bits, one could describe EdDSA with SHA-256 as first applying a fake "hash function" that outputs a fixed-length encoding of the input message as slightly more than 2^{64} bits. Algorithmically, however, computing EdDSA in this way would be intolerably slow. Furthermore, Figure 1 is presented as a general description of signatures, not merely as a description of signatures using hash functions with limited input lengths.

[9] Otherwise, why didn't the text simply say without further restrictions that DSA "may be used to verify signatures"?

the standard definition, and should not be an input in [45, Section 7.6]. There are two options for fixing this: recompute the public key as part of the signing process; or cache the public key as part of the private key.

[45, Section 7.8.3]: See Section 2.5 of this document.

B Further comments from author 1 on draft SP 800-186

"This recommendation includes two newly specified Montgomery curves, which claim increased performance, side-channel resistance, and simpler implementation when compared to traditional curves.": See above regarding "claim". For comparison, [23] unskeptically repeats NSA's "For efficiency reasons, it is desirable to take the cofactor to be as small as possible" claim, which (1) was always incorrect for ECDH and (2) has been incorrect for signatures since the advent of Edwards curves.

"These curves are only to be used with the EdDSA digital signature scheme in FIPS 186-5.": This is ambiguous. Does it mean that the curves are not to be used with, e.g., ECDH? Is NIST trying to suggest that there is some sort of problem for the billions of people already using X25519 (ECDH using Curve25519 with Montgomery x-coordinates)? This usage is not NIST-standardized today, but if NIST subsequently decides to standardize X25519 then presumably this will be in separate documents such as an update of SP 800-56A. An underlying document on "Elliptic curve domain parameters" should be written in a way that it is not forced to change.

Presumably NIST's intent was instead to say that, *within the scope of FIPS 186-5*, these curves are to be used with EdDSA (or HashEdDSA) rather than ECDSA. This will confuse readers who understand that Curve25519—in short Weierstrass coordinates—can be plugged into ECDSA:

- Should the reader think that plugging Curve25519 into ECDSA causes problems avoided by plugging NIST P-256 into ECDSA? What are these problems supposed to be?
- Or should the reader think that plugging Curve25519 into ECDSA causes problems avoided by plugging Curve25519 into EdDSA? This is easy to justify, but should be covered in the signature standard rather than in the curve specification.

Later the document defines a short Weierstrass curve W-25519, suggesting that NIST's intent is to draw a syntactic distinction between short Weierstrass curves to be plugged into ECDSA and other curves to be plugged into EdDSA. This could be more clearly expressed as follows: "These curves are approved for use with the EdDSA and HashEdDSA digital signature schemes in FIPS 186-5. Each curve has an equivalent short Weierstrass curve approved for use with ECDSA."

Brainpool curves "are allowed to be used for interoperability reasons": What does "allowed" mean here? Does it mean something weaker than "recommended" in a document that "specifies the set of elliptic curves recommended for U.S. Government use"? What reasons qualify as "interoperability reasons"?

The draft defines "morphism" as "mapping from a first group to a second group that maintains the group structure". The following changes should be made for clarity. First, "morphism" should be renamed "group morphism", to avoid confusion with other types of morphisms. Second, "maintains the group structure" should be written as "maps addition to addition". Otherwise the reader could understand "group that maintains the group structure" as "group that has the same group structure as the first group".

The draft defines "isogeny" as "morphism from a first elliptic curve to a second elliptic curve". Given the definition of "morphism", this does not match the standard definitions[10] of "isogeny" in the literature. An elliptic curve E over a finite field $\mathbf{F}_q$ carries more information than the group $E(\mathbf{F}_q)$: it also defines, for example, the group $E(\mathbf{F}_{q^2})$. An isogeny from E to E' correspondingly carries more information than a group morphism from $E(\mathbf{F}_q)$ to $E'(\mathbf{F}_q)$. For example, the Frobenius endomorphism $(x, y) \mapsto (x^q, y^q)$ from E to E is not the identity endomorphism even though it induces the identity morphism from the group $E(\mathbf{F}_q)$ to $E(\mathbf{F}_q)$. Considering $E(K)$ for an algebraic closure K of $\mathbf{F}_q$ would not fix the problem, since it would allow *more* group morphisms than isogenies; this would also be in conflict with subsequent text that implicitly restricts attention to $E(\mathbf{F}_q)$, such as text regarding the number of curve points. Furthermore, a group morphism from $E(\mathbf{F}_q)$ to $E'(\mathbf{F}_{q'})$ with $q \neq q'$ is an "isogeny" in the draft but not in the standard definitions.

The draft defines "l-isogeny" as "isogeny with kernel of size l". A reader who views an isogeny from E to E' over $\mathbf{F}_q$ as a group morphism from $E(\mathbf{F}_q)$ to $E'(\mathbf{F}_q)$ (see above) will assume that the "kernel" here is the subgroup of $E(\mathbf{F}_q)$ mapping to 0; but this again does not match the standard definitions. For example, if $q = 2^{255} - 19$ and E is Curve25519, then multiplication by 3 on E is a 9-isogeny under the standard definitions and not a 1-isogeny, even though its kernel in $E(\mathbf{F}_q)$ has size 1. Even if the definition is adjusted to consider kernel elements defined over extensions of $\mathbf{F}_q$, the definition will not match the standard definition for l divisible by the field characteristic.

"The operation addition" in [**23**, Section 3.1.3] is unclear. Saying "the Edwards addition law" would be clear.

The definition of the points of $E_{a,d}$ in [**23**, Section 3.1.3] does not match the standard definition without further assumptions. The statement that the set forms a group under the Edwards addition law is also incorrect without further assumptions. An easy fix is to restrict attention to the case that a is a square and that d is not, which is done anyway later in the paragraph to guarantee that the Edwards addition law is complete. The document does not seem to have any use of the incomplete case.

[**23**, Section 4.1.2] incorrectly says "this appendix".

[**23**, Table 1] is incorrect for (e.g.) Curve25519.

[10] Regarding "definitions" vs. "definition": Some authors allow the zero map as an isogeny while others do not. All of the standard definitions of nonzero isogenies are equivalent.

[23, Section 4.1.5] says that users can "generate their own base points to ensure a cryptographic separation of networks". No definition is provided for this "cryptographic separation", and it is not clear what security properties are being claimed here. Allowing users to generate their own base points makes some types of tests more difficult, and gives users opportunities to shoot themselves in the foot.

The statement that "one cannot reuse an implementation for elliptic curves with short-Weierstrass form that hard-codes the domain parameter a to -3 to implement Curve25519" is incorrect. One can apply a suitable isogeny to transform a to -3. The similar statement in B.2 is incorrect for the same reason.

"P-385" is a typo for "P-384".

[23, Appendix C.2.2.1] uses two different notations for the twist cofactor.

C Further comments from author 1 regarding NIST's Request for Comments

"No longer referenced in FIPS 185-5": This appears to be a typo for 186-5.

"Working in collaboration with the NSA, NIST included three sets of recommended elliptic curves in FIPS 186-2 that were generated using the algorithms in the American National Standard (ANS) X9.62 standard and Institute of Electrical and Electronics Engineers (IEEE) P1363 standards.": What exactly is NIST's justification for making claims regarding the method that NSA used to generate these curves? The fact that a hash matches is publicly verifiable, but the distribution of "random" inputs is not. I have heard NSA employees claiming that the "random" inputs were actually generated as hashes of English text chosen (and later forgotten) by Jerry Solinas.

D Further comments from author 2 on draft SP 800-186

```
Comments on NIST SP 800-186 draft.pdf
For
"Specification of new Montgomery and Edwards
curves, which are detailed in Elliptic
Curves for Security [RFC 7748]. These curves
are only to be used with the EdDSA
digital signature scheme in FIPS 186-5."
it makes more sense to refer to
https://tools.ietf.org/html/rfc8032
but probably the whole phrasing should change

Section 1.2 is bouncing between different scopes and does not fit with the
summary above.

p.12
```

"Appendix B: Relationship Between Curve Models" ->
"Appendix B: Relationships Between Curve Models"

"prime curves": term is not defined

The glossary is not accurate; at the very least include fields of definition, rational maps, and include "nonzero" in the definition of order. Some definitions are very vague; some don't match.

The glossary uses 0 as identity, section 2.2 uses \emptyset.

Writing "tr" in italics is confusing as this is canonically interpreted as t*r. Textbooks use "t" to denote the trace of Frobenius of a curve.

3.1.3
It is not correct that the described set of points forms a group, that only holds for complete curves, else there are points at infinity. This needs to be rephrased.
I don't understand the restriction of Edwards curves to EdDSA, but they sure are suitable for that.

l.394 for binary fields, GF(q) needs to include a representation of the field

l.396 Given that you use (u,v) as coordinates for Montgomery curves, writing "G=(G_x,G_y)" is not proper. The "=(G_x,G_y)" part can be omitted here without problem.

l. 397 - 402
Does this paragraph mean that FIPS accepts user defined curves?
I don't think that this is a good idea.

l.404 This needs to say that the points are defined over GF(q)

l.405/417 and others: As stated before, do not use "tr" as variable

l.408/409/413 Use the same symbol for the group generated by P (currently you use \langle and \rangle as well as < >)

l.414 "P" -> "P"

l.426 "as small as possible." is not accurate, change to "small."

l.427 "below" is non specific

l.427 Curve25519 has cofactor 8, so include 8 in the listing

l.449 "this appendix" is not defined

l.444 The bit length of n should be smaller or equal to the bit length of p, not the other way around. This table does not fit with the curves recommended in this draft

l.467 the choice of pentanomials is not described correctly. "t^a has the lowest degree m" would say "t^a has the lowest degree of all irreducible polynomials of degree m"

l.468 Replace "of degree m and the second term t^a" by "of degree m with the second term t^a"

l.477 The method does not guarantee what it claims.

l.484 Earlier you called this a form, not a name of the curves. Being special is independent of the form the curves are given in.
This is in contrast to Koblitz curves which are special by nature.

l.486 Are you sure about this? Does that mean that you can exclude other conditions?

Section 4.1.5
As the recent vulnerability in windows (CVE-2020-0601) showed, it is dangerous to leave the choice of base point to the user. The attack using a basepoint so that the DLP for the public key becomes 1 has a valid basepoint (yet no seed).
I would recommend against user-chosen curves or base points. However, I do not see a benefit of provably random base points over choosing the smallest point for some definition of smallest, or any other deterministic way.

l.501-503 I don't think anybody advertises these features for the _Weierstrass_ form of these curves, so omit "the curves W-25519 and W-448 may provide improved performance of the elliptic curve operations as well as increased resilience against side-channel attacks while allowing for ease of integration with existing implementations."

l.524 All values except for the seed are provided in decimal and hex, so update this sentence

I did not check the values provided

l.697 in analogy with the previous statements the prime should not be specified here, or it should be specified for the other curves as well.

l.698 n_1 is not in math mode. None of the other curve descriptions mentions the quadratic twist of the curve, I would skip it here as well.

W-25519 Since you are defining isogenies anyways, you may also define a curve with a=-3 that is isogenous to Curve25519 and then provide the isogeny rather than the simple isomorphism.

W-448
Same comments regarding the mention of p and the twist as for W-25519 as well as a=-3.

l.786 I disagree with the "Similar to W-25519 and W-448" part of this statement. The statement is also missing that these formulas are simpler to implement.

4.2.2.1 and 4.2.2.2
Throughout these sections, A and B are not stated in italics. The twists are mentioned but neither n1 is stated.

For Curve448 the respective Edwards curve is stated while this is missing for Curve25519. These sections should be kept in parallel

4.2.3 The feature that addition on Edwards curves is complete is missing but is what makes them easier to implement

Decide on whether to call the curves E-,,, or Edwards...

l.894 The curve is not isomorphic but birationally equivalent to Curve25519

l.919 n1 is not in italics

4.2.3.1 and 4.2.3.2
The twists are mentioned but neither n1 is stated.

4.3.1
It would make more sense to have the type be "Koblitz" to indicate that the Frobenius endomorphism can be used

l.1043 Earlier the irreducible polynomial was called p(t) rather than f(z); f is not a good name as that is typically used for the right-hand side of a Weierstrass equation y^2 + h(x)y = f(x). Stick with p(t) or change to yet another letter; make sure to also adjust the variable name.
This comment applies to all binary curves

l.1060-1062 The normal basis is not given, so this is not defined. State T.
This comment applies to all binary curves

l.1319 -P is not in italics

l.1323 and 1326 There is no reason not to state these as explicit definitions of x= ... and y= ... rather than implicitly. Yes, it's a simple transformation but this is not friendly to the user

A.1.2 Again A and B are not in italics

l.1330 -P is not in italics

l.1334 and 1337 State u and v explicitly in terms of the input values

l.1349 -P is not in italics

l.1350 replace \emptyset with (0,1)

In analogy with Weierstrass and Montgomery curves it would make sense to define the identity and negation before the addition formula, but unlike there it is not necessary here.

l.1345 Q is not in italics

l.1355 -P is not in italics

l.1359 and 1362 State x and y explicitly in terms of the input values

B.1 again, A and B are not in italics (except for one B)

l.1372 This map is not an isomorphism of curves but a birational equivalence. It is an isomorphism of the finite groups of points over GF(p), but not a curve isomorphism

l.1373 "thereby showing that the discrete logarithm problem in either curve model is equally hard." is correct and should be stated as motivation

somewhere -- but this does not make sense here in the appendix. Move this to the body of the text where the appendix is announced

The maps stated here are the standard maps between Edwards and Montgomery curves, but these do not match the specific maps given earlier, which include an extra parameter alpha. The definitions should match

l.1389 This map is actually an isomorphism of cures, so keep as is

l.1397 "recommendation" -> "Recommendation" (at least in line with previous typesetting, else fix there)

To make B2 useful for implementations using a=-3 you should include the isogeny version -- or skip W-25519 and W-448 completely

B.4
This section is not comprehensible as is. People who don't understand what an isogeny is will miss that this is not a one-to-one mapping and that this is OK to use only for the prime-order subgroup

l.1426 should come before l.1424 (by grammar)

l.1433 "prime number" -> "prime field"

l.1437 what is the motivation of allowing such a large range for h? The given curves have h <= 8.

l.1439/1440 you probably want to make sure that q is large (and not 2), else you're in trouble with the Koblitz curves

l.1442/1443 Once you fix the notation for trace to be t choose a different letter for the embedding degree; k is the typical letter.

l.1444 why do you choose such a low bound on the embedding degree? This is typically on the order of n, so much much larger, and I don't see any reason to stay small. For smallish values one might need to worry about the number of subfields.

l.1462 remove "The curve parameters a and b are:" as this does not fit what comes next

l.1474 append "from Curve25519 and Curve448".

l.1485 and 1501 The group is cyclic, not the curve

l.1487 and 1503 the cofactor is called h1 above, not h'

l.1511 and 1513 the map is a birational equivalence, not an isomorphism

l.1522 remove "wiz."

l.1524 append that this is about "chosen to satisfy the following"; but again the following is about more than a and b

l.1534 remember to adjust f(z) if you change variable names earlier.

In the curve parameters you also mention normal bases, this is missing here.

l.1543 again the text after this covers more than a and b
The specification of the field is missing.

In the curve parameters you also mention normal bases, this is missing here.

l.1572, 1573, 1579, and 1580 h is already used for the cofactor, choose a different letter, e.g. H

Step 9: for all curves you have a=-3, so this should be mentioned here. None of your curves have a=b; worse, he verification process is specific to a=-3, so other choices of a would fail there.

l.1590 the title is misleading; this is not a test of pseudorandomness but a test that a and b match seed. The text following the section title is accurate

l.1594 The inputs need to include b, or b and a.

Adjust the variable names to match the generation procedure

Either include a in the test (and input) as b^2c=a^3 or adjust the generation procedure

C3.3 same comments on h as above; now there is also a clash for z, which is the variable in f(z), the irreducible polynomial

C3.4 the input needs to include b, make clear the representation for the normal basis is used

l.1662 elsewhere HASH is in italics; it actually should be like here

in upright font everywhere

l.1686 append "given by the domain parameters ..." The same applies to the tests for other curve shapes and the complete tests

l.1687 and 1690 these lines are not compatible as Q=(x,y) implies that Q is not the point at infinity, remove "=(x,y)" in l.1687. This requires defining x and y in l.1691 using "Q=(x,y)"

l.1693 "point on the" -> "point on"

If the curve can be specified by the user you also need to verify that n and p are prime and that the curve is elliptic. The same applies to the tests for other curve shapes

l.1712 remove "=(u,v)" as above

l.1716 include definition of u and v as above

l.1718 "point on the" -> "point on"

l.1754 "If Q is the point at identity element (0,1)" -> "If Q=(0,1)"

l.1795 is missing an underline for y

l.1805 do not use GF(2) to describe the set {0,1}

D.2.2 normally B is in italics

l.1822 remove "(mod 2)"

l.1824 remember to check this if you change z to a different letter

D.2.2 normally B is in italics

l.1822 remove "(mod 2)"

l.1824 remember to check this if you change z to a different letter

D.3 is specific to Weierstrass curves (binary or prime fields); Montgomery and Edwards curves are not covered. I would prefer not to have this section at all and use standardized base points

l.1855 and 1860 "F_q" is not defined here; change to GF(q) as elsewhere

l.1855 This does not match the domain parameters; adjust to fit the other descriptions

l.1858 what do you mean by "or its equivalent"? This is too vague for a standard; the same applies to the following sections

I would replace this with a procedure that finds a point on the curve, rather than hiding this requirement in a comment; once that's done, the comment can be deleted

l.1862 E should be in italics

l.1868 there is no "verifiably random nature" if the x and y are not generated from seed an no method is specified here, so remove this comment

l.1870 You use a different symbol for the identity here than normally, same for the following sections

l.1877 this needs to specify that n is a large prime. Also, this statement relies on (x,y) being valid, while the selection routines for Curve25519 and Curve448 were incrementing u

l.1889 do you want \not= here?

l.1892 and 19082 change F_q to GF(q)

l.1935 n is already used for the order of G, use a different letter

l.1937 upper case letters denote points, so use other variable names instead of Q and S

l.1940 I suggest to specify a quadratic non-residue as input and skip this step

l.1966 I don't see any reason for this line

l.1968 The section is describing a general procedure to compute square roots, but now mixes this with u/v. This is useful for decompression for Edwards curves but needs a proper subsection title.

l.1970 and 1976 There is no mention of "decoding" earlier, thus remove or adjust the rest.

l.1983 This is not the best inversion method for SCA protection, so this should not be stated as a "should" condition. Using Fermat's little theorem is easier; there exist other methods that are constant time.

l.1998 for the avoidance of doubt, say that this is computed over Z, not modulo 2

G.1 starts by covering both pseudo Mersenne and Crandall primes, but in l.2044 only pseudo Mersenne numbers are mentioned, this should mention both

l.2055 italicize B

I did not check the numbers in this appendix; executable code would be more useful for testing

Either adjust the text in L.2048/2049 or add a matching text before l.2108.

Everywhere else 25519 is handled before 448, it would make sense to match this order here as well.

Efficient implementations of Curve25519 use radix 25.5, why do you present a different radix here?

G.2
Why do you present the Lucas sequences for the Koblitz curves? This makes the algorithms unnecessarily imposing. Computing a representation to the base of Frobenius and then turning that into a tau-NAF is shorter, see Solinas' paper, no need to do this as a complicated 1-pass algorithm.

l.2230 the change in endianess is confusing; if you do change it, make sure to adjust l.2146 as well (from right to left shift)

l.2350 are you sure about the statement "These curves were pseudorandomly generated"?
What protocols can these curves be used for? Would the implementation be FIPS certified?

E Further comments from author 2 on draft FIPS 186-5

Notes on NIST FIPS 186-5 draft

I did not review the RSA part.

2.3
Unify notation -- this uses a mod n while SP 800-186 uses a (mod n); similarly [n]X vs. nX for scalar multiplication
The ECDSA and EdDSA parts use the mod notation from SP 800-186, so adjust here; the use of [ ] in scalar multiplication is incompatible.

At least one of p and q should mention
"2. size of the finite field GF(p) (or GF(q)"

The base point of an EC should be included in the list

6.1
"if the elliptic curve was randomly generated in a verifiable fashion": same comment as for SP 800-186, this does not prove it.

Table 1: what is the justification for allowing so large cofactors? Why is this included if the curves are fixed by SP 800-186?

Do not use "GF_p" or "GF_{2^m}" to denote finite fields.
In line with SP 800-186 I suggest using "GF(p)" and "GF(2^m)".

I recommend against user-generated curves, but I don't see a reason for having G generated randomly rather than by a deterministic method finding the smallest or first valid point in some sequence.

6.1
The "(successfully)" part here does not make sense
"could be accidentally used (successfully) for another purpose"

6.4
Item 5 does not make sense for the deterministic ECDSA version

6.4.2
The algorithm does not check that Q is on the curve and has the correct order.

7.1
It is good to highlight the importance of protecting against side-channel and fault attacks, however, this is not a feature unique to EdDSA and should be included in the sections on ECDSA and RSA as well. The same holds for the verification of implementation, which is even more important for ECDSA because of the more complicated arithmetic, so make sure to include the same comment, or a strengthened version, there.

The notation is inconsistent
Given that you use (a,d) as the curve parameters for Edwards curves in SP 800-186, the use of d as the private key is not good. Parameters b and c are not mentioned in SP 800-186. H is called Hash before, while H is the output of the hash function. Here H is a function that is not exactly the stated hash function as some strings are appended to the input.

7.2 This uses \bullet and * in a single line, both to indicate integer multiplication; make the notation consistent

7.2 / 7.3 Change to "Point Encoding" and "Point Decoding" or leave out "Point" in both.

7.3 requires a and d as curve coefficients: a is not defined and d is used to denote the secret key (see above)

Remove
"Square roots can be computed using the Tonelli-Shanks algorithm (see NIST SP 800-186, Appendix E)."
as this is not used for either of the two curves.
Instead include that x_0 selects the correct root for x.

Remove "*" in the second condition for b)

Unify
"Otherwise, no square root exists, and the decoding fails."
and
"Otherwise, no square root exists for modulo p, and decoding fails."

The text
"For both cases, if x=0 and x_0=1, point decoding fails. If x (mod 2) = x_0 , then the x-coordinate is x. Otherwise, the x-coordinate is p - x."
Should be part of item 2. Given the algorithm continues with 3, I strongly suggest to move the routines for computing squareroots outside of this environment and to put a forward reference in 2.

7.4 and following sections
This is back to using d as private key and H instead of Hash, make sure to unify this. Note that H is not used as defined here, as H(d) includes extra inputs for Ed448. This needs fixing.

7.6

2.1 and 2.2 need to state that r is turned into an integer < n
4. should include a reference to section 7.4

4.1 and 4.2 miss how the hash outputs turn into integers;
remove "*" as multiplication is not denoted by any symbol in other places.

7.7.
Input 3. "that is valid for domain parameters D." This is checked in
1; so skip here that is is valid

Process 1 and 4 should have R in italics

Process 2 is not analogous to the signing procedure; unify

Process 4: remove "*", change "S" to "s", change "(2^c*t)" to "[2^ct]"

7.8 I suggest to also include reasons for using EdDSA over HashEdDSA.

The same comments regarding use of d and H apply as above.
Again, H is not used as such.

Process 2 needs to define s

3.1 and 3.2 need to turn r into an integer

5.1 and 5.2 remove "*" and turn the outputs of the hash functions
into integers

7.8.2
Input 3. Same comment as above on validity

Process 1 and 4 should have R in italics

Process 2 is not analogous to the signing procedure; unify

Process 3: refer to the signing procedure for the definition of dom2
and dom4.

Process 4: remove "*", change "S" to "s", change "(2^c*t)" to "[2^ct]"

7.8.3
What do you mean by "believed"; be concrete and state the requirements
on the hash function.
I would skip "Note that the risk of collisions using either SHA-512 or
SHAKE256 is considered negligible.' as this language is not compatible

with the rest of the standard.

Appendix A
"math" -> "mathematics" (this sentence is still very colloquial)

A.2.1
Note that this matches d in ECDSA not in EdDSA
In SP 800-186 l is the length of n, here it is N; Q=dG should be Q=[d]G; same for the next section.

Table A.2 does not match the curves in SP 800-186.

A.4.2 what is the difference between rejection sampling and the "Discard Method"

B.2.1/2 does not match the representations elsewhere which start at _0 and have x_i belong to 2^i.

Giessmann, Ernst

From: Ernst G Giessmann
Sent: Thursday, January 30, 2020 8:22 AM
To: SP800-186-comments
Subject: Comment on Draft NIST SP 800-186

Dear editors,
thanks for good job. The included remarks are almost editorials and I guess, that they are found already by others. If there are new ones, let me know ;-)
One remark on the "(mod n)" notation. You defined "mod" as the operation reducing an integer "modulo n" to the remainder. Therefore "6 mod 5" is defined 0 mod 4 as well. And if you write "0 (mod 4)", then it can't be the modulo reduction. Quite often it is clear that you had the equivalence relation in GF(n) in mind. But, as it is not defined explicitly, you must use only one symbol for it, either two bar equal sign or three bar equivalence sign (line 506). Therefore many of the remarks can be resolved by using "(mod n)" in brackets with the equivalence symbol or "mod n" without brackets.

Kind regards,
/Ernst.

Received: December 18, 2019
Status: Posted
Posted: January 29, 2020
Tracking No. 1k3-9dy4-dnfu
Comments Due: January 29, 2020
Submission Type: API

Hartog, Kyle

Docket: NIST-2019-0004
Request for Comments on FIPS 186-5 and SP 800-186

Comment On: NIST-2019-0004-0001
Request for Comments on FIPS 186-5 and SP 800-186

Document: NIST-2019-0004-0004
Comment on FR Doc # 2019-23742

Submitter Information

Name: Kyle Den Hartog
Email: kyle.denhartog@mattr.global
Organization: MATTR

General Comment

At MATTR we're concerned about the exclusion of the now popularly used curves secp256k1 and Curve25519 missing from these documents. The prominent use throughout the blockchain space should be an encouraging factor for FIPS 186-5 and SP 800-186 to additionally support these curves. We believe that FIPS 186-5 and SP 800-186 should add these curves to make it more likely that FIPS compliant hardware emerges. The inclusion of these curves would support blockchain solutions which would impact many different industries in a positive manor. As more and more capabilities in finance, Identity and access management, cybersecurity, governments agencies, and other major industries began working with Ethereum and Bitcoin it will be especially advantageous to support the use of these curves. We urge the authors of FIPS 186-5 and SP 800-186 to support the addition of secp256k1 and Curve25519 for key agreements to make compatibility and support of strong software implementations and hardware support more likely.

Ireland, Marc

From: Marc Ireland marc.ireland@nxp.com
Sent: Monday, January 27, 2020 9:08 AM
To: fips186-comments fips186-comments@nist.gov
Subject: Comment on Draft FIPS 186-5

Hello,

Attached please find comments from NXP Semiconductors on draft FIPS 186-5. Please confirm receipt as soon as is convenient.

Thank you,

Marc Ireland
Certifications Expert
NXP Semiconductors

FIPS 186-5

Physical attacks

The draft writes (Section 7.1) “Care must be taken to protect implementations against attacks such as side-channel attacks and fault attacks”.

In order to aid the practitioners who care about e.g. fault resistance why not follow the advice from Section 4.2 of [7]? By having the choice to randomize the signature algorithm many of the presented attacks are prevented or at least getting much harder.

This means including additional random nonce in the hash computation (Step 2, Section 7.6 of the NIST FIPS 186-5 draft). Adding some randomness does not change the proposed verification algorithm, does not weaken security and one can still do unit testing by using a constant value. Moreover, noise from a poor random number generator will not harm the security of the signature scheme. When no such protection is needed this additional random nonce can be constant or omitted.

This does mean, however, that the scheme loses the deterministic signature property.

Same remark holds for the ECDSA deterministic signature .

Cofactorless EdDSA Signature Verification

In [EdDsaCofVer], the authors propose a cofactorless verification of the EdDSA signature. Shouldn't this cofactorless verification be an option proposed in FIPS 186-5?

[EdDsaCofVer]: Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe and Bo-Yin Yang, “EdDSA for more curves", 2015.07.04 (https://pure.tue.nl/ws/portalfiles/portal/3850274/375386888374129.pdf)

Small subgroup attack

Small subgroup attacks are applicable to curves with a cofactor > 1. Such curves are referenced in SP800-185 (e.g. W-25519), therefore a small sub-group check shall be performed in the ECDSA algorithm described in FIPS 186-5.

E448

It is not clear what the curve E448 specified in SP800-186 shall be used for. If it shall be used in the EdDSA scheme, then additional information needs to be specified (choice of hash function, point encoding mechanism, etc.)

SP800-186

Correspondence between curves (Appendix B)

It should be made clear that the correspondence between twisted Edwards curves and Montgomery curves does not hold for all curves, only in the case a is a square and d a non-square (l1372).

Besides the correspondence between some curves is missing (e.g. correspondence between Curve25519 and W25519, between Edwards448 and Curve448)

Typo

In FIPS 186-5:

- Page 10, paragraph 2: "the public key needs"
- Remove DSA from document (e.g. figure 2)
- Section 5.2 RSA Key Pair management. Point 5: remove domain parameters
- Section 5.4.1, correct "defination"
- Section 6.4.2, ECDSA Signature Verification Algorithm, step 4.
  Compute s−1 = (1/s) (mod n) using the routine in Appendix C.1. To be replaced by Appendix B.1
- Section 7.7: [2c * S]G = [2c]R + (2c * t)Q should be [2^c * S]G = [2^c]R + [2^c * t]Q
- Section 7.2 Encoding: The multiplication of 2^8 and h[1] and 2^248 and h[31] seem to use different notation. The first operator is not defined in Section 2.3.
- Section A.1.2.2 : reference to C.10 should probably be B.10
- We suggest the Section 3 to be reworked. For instance page 10, "For both the signature generation and verification processes, the message (i.e., the signed data) is converted to a fixed-length representation of the message by means of an approved hash function. Both the original message and the digital signature are made available to a verifier."
  It is not completely correct, since for the pure EdDSA there is no hashing of the message.

- Reference [7]:
  Ambrose C, Bos JW, Fay B, Joye M, Lochter M, Murray B (2017) Differential Attacks on Deterministic Signatures. Cryptology ePrint Archive preprint. https://ia.cr/2017/975
  was actually published, better to reference
  Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter and Bruce Murray: Differential Attacks on Deterministic Signatures. RSA Conference Cryptographers' Track - CT-RSA, Lecture Notes in Computer Science 10808, pp. 339–353, Springer, 2018.
- "Section 7.7, first step of "Process": The variable "s" conflicts with the integer "s" in Step 4 of "Process" in Section 7.6. This conflict should be resolved.

Markowitz, Michael

From: Michael Markowitz <markowitz@infoseccorp.com>
Sent: Tuesday, January 28, 2020 1:24 PM
To: SP800-186-comments <sp800-186-comments@nist.gov>
Subject: question on E448 in SP800-186 draft

Folks: I can't find comments on the draft posted online, so please pardon me if this has been previously discussed …

First I want to note the typo in the value for the y-coordinate of the generator of E448 in section 4.2.3.3: there an extraneous 'L' at the end of line 983.

Second, I want to ask whether the order, n, of the subgroup generated by (Gx,Gy) has been validated: when I compute n(Gx,Gy), I get the point (0,-1) of order 2, rather than the identity element (0,1). I find this very strange… Is my code (which works fine with the parameters for Edwards25519 and Edwards448) flawed, or is the value of n in the draft (and in RFC7748) incorrect?

```
n =
181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779
  = 0x3fffffffffffffff ffffffffffffffff ffffffffffffffff ffffffff7cca23e9
c44edb49aed63690 216cc2728dc58f55 2378c292ab5844f3
  = 2^446 - 0x8335dc163bb124b65129c96fde933d8d723a70aadc873d6d54a7bb0d
```

Thanks in advance for your response!

Regards,
Michael


```
==========
Michael J. Markowitz, Ph.D.        Email: markowitz@infoseccorp.com
Vice President R&D                 Office: 708-445-1704
Information Security Corporation   Direct: 708-872-0962
1011 Lake Street, Suite 425        Fax:   708-445-9705
Oak Park, IL  60301                WWW:   http://www.infoseccorp.com
```

Received: November 13, 2019
Status: Posted
Posted: January 29, 2020
Tracking No. 1k3-9dag-qwp5
Comments Due: January 29, 2020
Submission Type: Web

Mattsson, John

Docket: NIST-2019-0004
Request for Comments on FIPS 186-5 and SP 800-186

Comment On: NIST-2019-0004-0001
Request for Comments on FIPS 186-5 and SP 800-186

Document: NIST-2019-0004-0002
Comment on FR Doc # 2019-23742

Submitter Information

Name: John Prue Mattsson
Email: john.mattsson@ericsson.com

General Comment

Dear NIST,

Thanks for your continuous efforts to produce well-written open-access security documents. These are two very important and well-written document. Comments submitted in two parts as there is a limit of 5000 characters.

High level comments:

- Excellent that NIST is adding Montgomery and Edwards curves from RFC 7748. Because of their excellent performance, these curves have already found substantial use in various industries for key exchange, ECIES, and signatures in deployed systems (TLS), new standards (IMSI protection in 5G), and upcoming standards like TLS ESNI, Group OSCORE, and EDHOC.

- We are fine with NIST deprecating binary curves and DSA. While they are quite many libraries that support them, we do not know of any deployment that are actually using them.

Comments on SP 800-186

- Line 160: Elliptic curves over binary field are deprecated, but still included in the document. What does this mean in practice? When is use allowed and when is it not allowed? This should be explained.

- Line 164: Specification of new Montgomery and Edwards curves, which are detailed in Elliptic

Curves for Security [RFC 7748]. These curves are only to be used with the EdDSA

This make it seems like the Weierstrass curve W-25519 can be used for anything, the Edwards curve Edwards25519 can be used for EdDSA and the Montgomery curve cannot really be use for anything. Are Montgomery curves specified only as a way to use a Montgomery code library for EdDSA? For industry use cases, we would like to use the Montgomery curve Curve25519 as much as possible for key exchange (e.g. in TLS) and for hybrid encryption like in ECIES.

- Line 286: have garnered academic interest.

These curves already have substantial deployment in various industries.

- Section 4.1.2: The document lets the reader calculate the security strengths themselves from the curve parameters and SP 800-57. It would be easier for the reader if the document listed the security strengths of the curves instead of forcing the reader to calculate them.

- Section 4.1.2: Following the security strength calculation, W-25519, Curve25519 and Edwards25519 has a security strength of only 112 as n is 255 bits. Algorithms with a 112 bit security strength are only approved to be used beyond 2030 minus the number of year of protection needed. We strongly suggest that NIST changes the security strength calculations and approve W-25519, Curve25519, and Edwards25519 as having 128 bit security strength.

- Line 481: For each curve size range, the following curves are given

This is not true as Edwards and Montgomery curves are not given for all ranges. Also the Weierstrass curves W-25519 and W-448 seem special rather than pseudorandom.

Best Regards,
John Preu Mattsson, Senior Specialist, Ericsson

Received: January 29, 2020
Status: Posted
Posted: January 30, 2020
Tracking No. 1k4-9epy-d584
Comments Due: January 29, 2020
Submission Type: Web

Patil, Harsh

Docket: NIST-2019-0004
Request for Comments on FIPS 186-5 and SP 800-186

Comment On: NIST-2019-0004-0001
Request for Comments on FIPS 186-5 and SP 800-186

Document: NIST-2019-0004-0008
Comment on FR Doc # 2019-23742

Submitter Information

Name: Harsh Kupwade Patil
Email: harsh.patil@lge.com

General Comment

See attached file(s)

Attachments

January 29, 2020

Information Technology Laboratory
ATTN: FIPS 186-5 and SP 800-186 Comments
Docket ID: NIST-2019-0004
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
sp800-186-comments@nist.gov

Re: Federal Register Notice (FRN) Request for Comments on Draft Special Publication (SP) 800-186 - Docket no. NIST-2019-004

Dear Docket Manager,

LG Electronics appreciates the opportunity to provide comments in the above-referenced proceeding. LG has a long history of developing and producing world-class electronics products in a range of consumer and business areas including home entertainment, home appliances, mobile communications, air solutions, renewable energy and vehicle components. LG was one of the first companies to introduce cellular phones using CDMA technology and is a leading developer of LTE technology. LG also is a leading supplier of Electronic Control Units (ECUs) such as Audio/Video Navigation, LCD clusters, and telematics modules to vehicle OEMs worldwide.

LG understands the importance of cybersecurity and is spearheading research in this area. For some time, LG has been addressing privacy and cybersecurity in various consumer electronics devices such as Smart TVs and mobile phones for our global consumers. And, as one of the leading Tier-1 automotive suppliers, LG understands the security issues in the vehicle ecosystem, and has established a globally-focused, automotive, Internet of Things (IoT) cybersecurity team.

LG Electronics also has been an active contributor in the University of Michigan's public-private partnership M-City working groups (Pillar1/2/3, Legal/Insurance, and Cybersecurity). We have actively contributed to a number of key global standards for security and privacy in the North America and Europe. Furthermore, our proposed cryptographic applications have been adopted by major OEMs in the connected-vehicle paradigm.

We are pleased to respond to the request for comments on NIST's proposal to update its standards on digital signatures and Elliptic Curve Cryptography (ECC) to align with existing and emerging industry standards. In addition to updating ECC standards, NIST is proposing the removal of Digital Signature Algorithms (DSA), noting recent security analysis against DSA implementation and increased industry adoption of ECDSA.

We commend NIST for its continued study into the advances of applications of elliptic curves in the cryptographic community. As highlighted in the comments below, LG generally supports the continued development and analysis of the application of protocols and approaches to security, including the use of Elliptic Curve Cryptography.

In summary, LG believes that there are many advantages to the proposed curves; however, there are multiple limitations that NIST should consider. First, the proposed curves may be limited in application. Not all industry segments will benefit from potential implementations based on the need for an increased number of memory resources and processing overhead.

Although NIST recommends the use of unsaturated limb representations for high-level and portable software implementations of Curve25519 modular arithmetic to minimize the effects of carry computations, this approach requires more registers to accommodate intermediate operands in low-level Assembly implementations. In addition, the conversion of finite-field elements from an unsaturated limb representation to a saturated one, and vice-versa, incurs in processing overhead. On the other hand, hardware architectures of modern embedded processors have incorporated application-domain instructions to maximize chip functionality, performance and efficiency. Some of these instructions were designed to enable faster carry computations.

Indeed, LG believes that saturated limb representations are most suitable for low-level Assembly implementations targeting embedded platforms having constrained register use, but that can leverage instruction-level carry handling mechanisms.

As a complement to the proposed unsaturated 26-bit representation from SP 800-186 (Draft), we propose reference algorithms, along with the corresponding upper bound analysis, for the implementation of the reduction modulo $2p = 2^{256} - 38$

using saturated limb representations. We performed an analysis of all the steps required in order to ensure that all carry values are properly handled during the reduction process.

More-detailed LG comments may be found on the following pages. LG Electronics applauds NIST's important focus on cybersecurity, and we welcome any additional related discussions.

If you have any questions regarding the comments, please contact harsh.patil@lge.com or henrique1.ogawa@lge.com.

Respectfully Submitted,

Harsh Kupwade Patil, Ph.D.

Principal Research Engineer
Edge Security Team,
Advanced AI
LG America R&D Center
Santa Clara, CA
214-801-7318
harsh.patil@lge.com

Peter (Seungyoon) Song, Ph.D.

Vice President
Advanced AI
LG Electronics
America R&D Center
Santa Clara, CA
speter.song@lge.com

John I. Taylor
Senior Vice President, Government Relations
LG Electronics North America
Washington, DC
202-719-3490
john.taylor@lge.com

Comments of LG Electronics Inc.
on Draft Special Publication (SP) 800-186 - Docket no. NIST-2019-004

Introduction/Background

Key exchange protocols and digital signature schemes are cryptographic building blocks for many applications using key distribution schemes and secure software updates based on code signing. These protocols are fundamental to preserve the integrity of software running in embedded devices, and to establish symmetric cryptographic keys for encrypted communication between different parties.

A. Elliptic Curve Cryptography (ECC)

From this perspective, Elliptic Curve Cryptography (ECC) is a set of cryptographic primitives based on mathematical foundations of special elliptic curves. Recently, the efficiency and security being achieved by ECC over legacy RSA solutions has promoted ECC-based digital signature schemes and key exchange protocols inside standardization initiatives involving academy, industry and governmental institutions.

In detail, an elliptic curve E over a field $\mathbb{F}_q$ is the set of coordinates $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, which satisfy the Weierstrass equation:

$$E/\mathbb{F}_q : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad \text{(Eq. 1)}$$

In Equation 1, $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and the curve discriminant is such that $\Delta \neq 0$. Special attention is given to curves defined over prime fields which can be represented in the Montgomery (or Twisted Edwards) model. These models facilitate implementations with faster and unified arithmetic formulas[1,2,3].

The main advantages of ECC over RSA are the reduction in the key sizes and processing times. ECC provides smaller keys and faster processing by relying

[1] D. Bernstein; P. Birkner; M. Joye; T. Lange; C. Peters, "Twisted Edwards Curves", in International Conference on Cryptology in Africa, pages 389–405. Springer, 2008.
[2] D. Bernstein; T. Lange, "Analysis and Optimization of Elliptic-Curve Single-Scalar Multiplication", in Contemporary Mathematics, 461(461):1, 2008.
[3] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", in Mathematics of Computation, 48(177):243–264, 1987.

on the complexity of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), conjectured to be exponential. Therefore, ECC is an efficient, yet conservative option for deploying public-key cryptography in embedded systems, and with less memory requirements.

B. Curve25519

An efficient, yet conservative instance of the ECDLP can be obtained by selecting prime curves of near-prime order without supporting any non-trivial endomorphism. Curve25519, defined in IETF RFC7748[4], is a popular curve offering a 128-bit security level, which is represented through the Montgomery model

$$\text{Curve25519: } y^2 = x^3 + Ax^2 + x, \qquad \text{(Eq. 2)}$$

where A = 486662. This Montgomery curve is defined over the prime field $p = 2^{255} - 19$. Curve25519 is ideal for ECC-based key exchange protocol (namely, X25519), since it allows the scalar multiplication to be computed using x-coordinates only.

C. Ed25519 Digital Signatures

Using a birational equivalence, Curve25519 can be also represented in the twisted Edwards model using full coordinates to allow instantiations of secure signature schemes:

$$\text{Edwards25519: } -x^2 + y^2 = 1 - \frac{121655}{121666}x^2y^2 \qquad \text{(Eq. 3)}$$

The Edwards25519 curve enables the use of Edwards-curve Digital Signature Algorithm (EdDSA), which is a signature scheme variant of Schnorr signatures based on elliptic curves represented in the Edwards mode. When instantiated using Edwards25519, the EdDSA scheme is called Ed25519, and is defined in IETF RFC8032[5].

Due to the improvements over ECDSA, Ed25519 has been adopted into well-known technologies across different application domains: SSH connection

[4] A. Langley; M. Hamburg; S. Turner, "RFC 7748: Elliptic Curves for Security", in Internet Engineering Task Force (IETF), 2016.
[5] S. Josefsson; I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", in Internet Research Task Force, Crypto Forum Research Group, RFC, volume 8032, 2017.

protocol (Secure Shell) adopted Ed25519 as an additional signature scheme (alongside RSA, DSA, and ECDSA) for client authentication to a server[6]. Ed25519 was also incorporated to the Transport Layer Security (TLS) Version 1.3 cryptographic suite for secure client/server applications[7]. In addition, Apple Inc.'s recently introduced CryptoKit framework for common cryptographic operations on iOS platform also includes the support for Ed25519 signature scheme[8]. Moreover, modern Hardware Security Modules (HSMs) have incorporated tamper-resistant technologies and hardware acceleration for the execution of the Ed25519 signature generation/verification algorithms[9,10].

D. F2255-19 Finite Field Arithmetic

Modular arithmetic operations in Curve25519 are defined over the $\mathbb{F}_p$ prime field[11], where $p = 2^{255} - 19$. Hereby, the results of addition, subtraction and multiplication operations are given as modulo the prime p. Since the reduction modulo p requires bit-level manipulations, the reduction modulo $2p = 2^{256} - 38$ is proposed in order to make intermediate results of the $\mathbb{F}_p$ arithmetic operations to fit within 256 bits[12]. As opposed to the reduction modulo p, the reduction modulo $2p$ does not require bitwise operations, and can be performed using only additions with carry. In this way, the reduction modulo $2p$ can be applied throughout groups of finite-field arithmetic operations (e.g. scalar inversion, X25519, point addition, point doubling), allowing the postponing of the reduction modulo p until the calculation of the final result.

For notation purposes, let us consider the arithmetic operations f, such that $R = f(X, Y)$. Moreover, being R up to 512 bits in length (in the case of f being a multiplication), let us also consider R_H and R_L as the higher and lower 256-bit limbs of R, respectively. The reduction modulo $2p$ consists in

[6] S. Moonesamy, "RFC7479 – Using Ed25519 in SSHFP Resource Records", in Internet Engineering Task Force (IETF), 2015

[7] E. Rescorla, "RFC8646 – The Transport Layer Security (TLS) Protocol Version 1.3", in Internet Engineering Task Force (IETF), 2018

[8] Apple Inc., "Apple CryptoKit – Perform cryptographic operations securely and efficiently", in Apple Developer, 2019. Apple and CryptoKit are registered trademarks of Apple Inc., in the U.S. and other countries.

[9] Infineon Technologies AG, "AURIX™ Security Solutions", 2019

[10] Thales Group, "SafeNet Luna Network HSM", 2019

[11] D. Bernstein, "Curve25519: New Diffie-Hellman Speed Records", in International Workshop on Public Key Cryptography, pages 207–228. Springer, 2006.

[12] M. Düll; B. Haase; G. Hinterwälder; M. Hutter; C. Paar; A. Sánchez; P. Schwabe, "High-Speed Curve25519 on 8-bit, 16-bit, and 32-bit Microcontrollers", in Designs, Codes and Cryptography, 77(2-3):493–514, 2015.

successively adding the value congruent to the carry modulo $p = 2^{255} - 19$ to the lower R_L limb, until the result fits in 256 bits.

Technical comment

On page 59, NIST SP 800-186 (Draft) states that "*The modulus for this curve (Curve25519) is* $p = 2^{255} - 19$. *Each integer A less than* p^2 *can be written*

$$A = A_1 \cdot 2^{256} + A_0 \qquad \text{(Eq. 4)}$$

where each A_i *is a 256-bit integer. As a concatenation of 256-bit words, this can be denoted by*

$$A = (A_1 || A_0)$$

The expression for B *is*

$$B = (38 \cdot A_1 + A_0)(\text{mod } 2p),$$

where all computations are carried out modulo $2p$ *rather than modulo* p.

This allows efficient modular reduction and finite field operations that try and minimize carry-effects of operands if each integer X less than 2p is represented as

$$X = X_9 \cdot 2^{234} + X_8 \cdot 2^{208} + X_7 \cdot 2^{182} + X_6 \cdot 2^{156} + X_5 \cdot 2^{130} + X_4 \cdot 2^{104} + X_3 \cdot 2^{78} + X_2 \cdot 2^{52} + X_1 \cdot 2^{26} + X_0, \qquad \text{(Eq. 5)}$$

where each X_i *is a 26-bit integer and where* X_9 *is a 22-bit integer. Note that in this case, multiplication by the small constant 38 does not lead to overflows if each* X_i *is stored as a 32-bit word. It turns out that the cost of occasional resizing of X, represented this way, is outweighed by savings due to the possibility of postponing 'carry' operations."*

<u>LG Comment</u>

NIST SP 800-186 (Draft) proposes the unsaturated limb representation shown above in order to avoid carry computations when multiplying 38 by the higher A_1 word (an operation equivalent to add 38 repeatedly for A_1 times into A_0). Since a multiplication of a 26-bit unsigned integer by 38 (6 bits) results in a 32-bit unsigned integer, the value resulting from this operation fits in a 32-bit register, eliminating the need for propagating the carry value to another register. Following a similar logic, an unsaturated 51-bit limb representation was proposed for implementations on 64-

bit platforms[13]. LG believes that these unsaturated limb representations are suitable for software implementations in order to facilitate code auditability and portability across different target platforms, to prevent mistakes when manually implementing carry propagation logic, and to avoid incorrect compiler's machine-code extraction from high-level carry logic implementations. Therefore, when adopting an unsaturated representation, the implementer neither needs to keep track of overflow conditions nor how/when to process the carry values.

Nevertheless, we believe that the register usage limitations in low-level Assembly implementations targeting embedded platforms might impose restrictions to the use of unsaturated representations, since they require an increased number of registers to accommodate finite-field operands. For instance, whereas eight (8) 32-bit registers are required for the representation of a single 256-bit integer, ten (10) registers are necessary when using the proposed 26-bit limb representation (Equation 5). Moreover, it is worth mentioning that most modern embedded processors implement carry handling mechanisms (e.g. overflow flags, carry instructions, multiply-and-accumulate instructions, implicit registers), which can be used along with a saturated limb representation in order to allow more efficient register usage schemes. However, when adopting a saturated representation, it may be tricky for the implementer to take care of all overflow conditions, and to make sure that all possible carry values are processed accordingly.

Therefore, as a complement to the NIST's suggested unsaturated 26-bit limb representation, LG proposes reference algorithms for the implementation of finite-field operations in Curve25519 using saturated limb representations. Along with the algorithms, we demonstrate a corresponding step-by-step upper-bound analysis throughout the reduction modulo $2p$ process in order to ensure that all carry values are handled during the reduction process. For our algorithms, we consider the constant-time implementation practices (aiming to thwart side-channel attacks), and that all the variables and intermediate results are represented in radix-2^{256}, as in Equation 4.

A. Modular addition and subtraction

Algorithm 1 describes the implementation of the reduction modulo $2p$ for the modular addition. Since field elements are up to 256-bit long, the sum S

[13] Y. Chen; C. Hsu; H. Lin; P. Schwabe; M. Tsai; B. Wang; B. Yang; S. Yang; "Verifying Curve25519 Software", in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 299–309. ACM, 2014.

resulting from the addition of the field elements X and Y can only carry at most one bit to c_{add}, as described in line 6 of Algorithm 1. Therefore, the value to be added back to S will be either 0 or 38.

If c_{add} is equal 1, the addition with carry must be performed again over S, since there might still exist intermediate carry values. The addition of 38 to S results in S', as depicted in line 7 of Algorithm 1. Although very unlikely to happen, if this addition step still carries a bit in c'_{add}, it is necessary to add 38 again to the S', which gives the final result S'' (line 8 of Algorithm 1).

Even considering that the probability of c'_{add} to be equal 1 is very low, a constant-time implementation requires the execution of all the aforementioned steps without any conditional statements.

```
1: input: X,Y ∈ {0,1}^256 ⇒ 256-bit integers
2: output: S'' ∈ {0,1}^256 ⇒ 256-bit integer

3: S,S' ∈ {0,1}^256 ⇒ intermediate addition variables, 256-bit integers
4: c_add, c'_add ∈ {0,1}^1 ⇒ carry variables, 1-bit integers

5: procedure ADD_MOD_2P (X,Y):
6:   2^256 · c_add + S ← X + Y
7:   2^256 · c'_add + S' ← S + 38 × c_add
8:   S'' ← S' + 38 × c'_add
9: end procedure
```

Algorithm 1 - Addition modulo $2p$, constant-time implementation

To demonstrate that no extra additions by 38 are required, Equation 6 describes an upper bound analysis for all the steps shown in Algorithm 1. Even considering the largest possible input values that can be represented by X and Y, the upper bound analysis show that the final sum S'' still fits in 256 bits. Therefore, when c_{add} is equal zero (0), the final sum S'' is equal S', which can be up to 256 bits in size. On the other hand, when c_{add} is equal one (1), the final sum S'' can be up to seven (7) bits in size. Thus, by repeating the addition by $38 \times carry$ twice, we ensure that no carry bits are left aside during the reduction process.

Algorithm 1, line 6:

$X_{max} + Y_{max} = 2^{256} - 1 + 2^{256} - 1 = 2^{257} - 2$

$\Rightarrow \therefore \lfloor log_2(X_{max} + Y_{max}) \rfloor + 1 =$ 257 bits

Algorithm 1, line 7:
$S_{max} + 38 \times c_{add} = 2^{256} - 2 + 38 = 2^{256} + 36$
$\Rightarrow \therefore \lfloor log_2(S_{max} + 38 \times c_{add}) \rfloor + 1 = 257$ bits

Algorithm 1, line 8:
$S''_{max} = S'_{max} + 38 \times c'_{add} = 74 \Rightarrow \therefore \lfloor log_2(S''_{max}) \rfloor + 1 = 7$ bits

Equation 6 – Upper bound analysis of the addition modulo $2p$ for $X_{max} = Y_{max} = 2^{256} - 1$

This upper bound analysis also applies for the modular subtraction, since the subtraction of the field elements X and Y can only borrow at most one bit to c_{sub} (the equivalent of c_{add} in addition operation). Thus, in this case, the value to be subtracted back from S will be either 0 or 38 as well.

B. Modular multiplication

Algorithm 2 describes the implementation of the reduction modulo $2p$ for the modular multiplication. Following the radix-2^{256} representation, we split the product resulting from the multiplication of the field elements X and Y in two parts: H and L. Since field elements are up to 256-bit long, the multiplication $X \times Y$ can carry at most 256 bits to H, as represented by the operation in line 8 of Algorithm 2.

In this case, the value of $38 \times H$ must be added back to the lower L limb (this is equivalent to add 38 into L repeatedly for H times). However, since the $38 \times H$ product might also get more than 256-bit long, we need to first reduce it in order to make sure it fits in 256 bits: if the $38 \times H$ product results in carry bits over c_{mul}, we need to add the value of $38 \times c_{mul}$ to the lower 256-bit limb of the $38 \times H$ product. These operations are shown in lines 9, 10, and 11 of Algorithm 2.

The addition of $38 \times c_{mul}$ into the lower 256-bit limb of $38 \times H$, resulting in up to 256 bits in M', can still carry up to one bit in c_{add}. In this case, a last 38 must be added into M' in order to obtain M'', which always fits in 256 bits. Lastly, since M'' is the value of $H \bmod 2p$ (therefore being up to 256-bit long), the last step of the multiplication modulo $2p$ consists in applying the addition modulo $2p$ over the lower 256-bit limb of $X \times Y$ (namely, L) and M'', as depicted in line 13 of Algorithm 2.

It is worth mentioning again that, despite the low probabilities of some carry values to occur, a constant-time implementation requires the execution of all the aforementioned steps without any conditional statements.

```
1:  input: X, Y ∈ {0,1}^256 ⇒ 256-bit integers
2:  output: P ∈ {0,1}^256 ⇒ 256-bit integer

3: L, H, M, M', M'' ∈ {0,1}^256 ⇒ intermediate variables, 256-bit integers
4: c_mul ∈ {0,1}^6 ⇒ multiplication by 38 carry, 6-bit integer
5: r ∈ {0,1}^11 ⇒ multiplication by 38 carry, 11-bit integer
6: c_add ∈ {0,1}^1 ⇒ addition carry, 1-bit integer

7: procedure MUL_MOD_2P (X, Y):
8:   2^256 · H + L ← X × Y
9:   2^256 · c_mul + M ← 38 × H
10:  r ← 38 × c_mul
11:  2^256 · c_add + M' ← M + r
12:  M'' ← M' + 38 × c_add
13:  P ← ADD_MOD_2P (L, M'')
14: end procedure
```

Algorithm 2 – Multiplication modulo $2p$, constant-time implementation

To demonstrate that no further repeated additions by 38 are required, Equation 7 details an upper bound analysis for all the operations listed in Algorithm 2. Again, even considering the largest representable values for X and Y, the upper bound analysis shows that the final product P still fits in 256 bits.

Algorithm 2, line 8:
$X_{max} \times Y_{max} = (2^{256} - 1) \times (2^{256} - 1) = 2^{512} - 2^{257} + 1$
$\Rightarrow \therefore \lfloor log_2(X_{max} \times Y_{max}) \rfloor + 1 = 512$ bits

Algorithm 2, line 9:
$38 \times H_{max} = 38 \times (2^{256} - 2) = 38 \times 2^{256} - 76$
$\Rightarrow \therefore \lfloor log_2(38 \times H_{max}) \rfloor + 1 = 262$ bits

Algorithm 2, line 10:
$38 \times c_{mul} = 38 \times 37 = 1406 \Rightarrow \therefore \lfloor log_2(38 \times c_{mul}) \rfloor + 1 = 11$ bits

Algorithm 2, line 11:
$M_{max} + r_{max} = 2^{256} - 76 + 1406 = 2^{256} + 1330$
$\Rightarrow \therefore \lfloor log_2(M_{max} + r_{max}) \rfloor + 1 = 257$ bits

Algorithm 2, line 12:
$M'_{max} + 38 \times c_{add} = 1330 + 38 = 1368$
$\Rightarrow \therefore \lfloor log_2(M'_{max} + 38 \times c_{add}) \rfloor + 1 = 11$ bits

Equation 7 - upper bound analysis of the multiplication modulo $2p$ for $X_{max} = Y_{max} = 2^{256} - 1$

Conclusion

In response to NIST's publication of Draft Special Publication (SP) 800-186, we presented reference algorithms for the implementation of the reduction modulo $2p = 2^{256} - 38$ using saturated limb representations. The upper-bound analyses provided along the reference algorithms show that all overflow conditions are properly handled during the specified algorithms' steps, and the final results always fit in 256 bits. Despite final overflow conditions occurring for a very small set of (X, Y) pairs, we believe that secure implementations need to cover the corner cases.

Although we demonstrated our algorithms and analysis using a radix-2^{256} for the sake of simplification of the mathematical expressions in the upper-bound analysis, the algorithms, the overflow conditions, and the upper-bound analysis apply to implementations in traditional 32 and 64-bit processor platforms. Furthermore, the analysis for the multiplication modulo $2p$ also applies for the finite-field squaring operation, since it is essentially a multiplication with equal operands.

LG Electronics appreciates your consideration of our comments in this FRN. For additional information, please contact Harsh Kupwade Patil (harsh.patil@lge.com) or Henrique Ogawa (henrique1.ogawa@lge.com).

Smith, David

From: Smith, David E. <David.Smith@cyber.gc.ca>
Sent: Thursday, January 30, 2020 4:52 PM
To: SP800-186-comments <sp800-186-comments@nist.gov>
Subject: Cyber Centre comments on SP 800-186 (Draft)

Please find below our editorial and technical comments on the Draft SP 800-186 issued for comment in October 2019.

David Smith
Canadian Centre for Cyber Security

Page, section, paragraph	Type	Comment
8, 4, 7	Editorial	The formatting of angle brackets on <P> on line 409 should be to be consistent with lines 408 and 413.
10, 4.1.3, *Polynomial Basis*	Editorial	The explanation of how to choose the pentanomial states that "the second term t^a has the lowest degree m". It should read "the second term t^a has the lowest degree among all irreducible pentanomials of degree m". Also the phrase "the third term t^b has the lowest degree among all irreducible pentanomials of degree m and the second term t^a …" should read "the third term t^b has the lowest degree among all irreducible pentanomials of degree m with the second term t^a …".
26, 4.3.1.4, *f*(*z*)	Editorial/ Technical	The definition of *f*(*z*) on line 1103 is incorrect. It should be *f(z) = z^{409} + z^{87} + 1*. See ANSI X9.142 Table B.3 for reference.