Public Comments Received on Draft FIPS 186-5: Digital Signature Standards (DSS)

(January 29, 2020 deadline)

updated 3/19/2021

Adinolfi, Shailee

From: Shailee Adinolfi <shailee.adinolfi@consensys.net>
Sent: Monday, January 27, 2020 4:50 PM
To: fips186-comments <fips186-comments@nist.gov>
Cc: Oliver Terbu <oliver.terbu@consensys.net>; Delak, Katya M. (Fed) <katya.delak@nist.gov>
Subject: SP 800-186 and FIPS 186-5 Comments

To whom it may concern,

Please find attached joint comments from ConsenSys, Decentralized Identity Foundation, Enterprise Ethereum Alliance, W3C Credentials Community Group, Hyperledger and individual W3C member companies regarding the SP 800-186 and FIPS 186-5. Thank you for your consideration.

Best, Shailee Adinolfi

Joint Comments from ConsenSys, Decentralized Identity Foundation, Enterprise Ethereum Alliance, W3C Credentials Community Group, Hyperledger and individual W3C member companies

Comments on SP 800-186 and FIPS 186-5

Applicable Documents

SP 800-186, Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters: https://csrc.nist.gov/publications/detail/sp/800-186/draft

FIPS 186-5, Digital Signature Standard (DSS): https://csrc.nist.gov/publications/detail/fips/186/5/draft

Joint Comments to the Applicable Documents by the following Organizations:

- ConsenSys
- Decentralized Identity Foundation
- Enterprise Ethereum Alliance
- W3C Credentials Community Group
- Hyperledger
- Individual W3C Member Organizations

Executive Summary

Since the Bitcoin Genesis block in January 2009, Blockchain and more broadly Distributed Ledger Technology (DLT), has seen exponential growth in its usage and applications. While DLT applications were initially only available on public networks that anyone could join, enterprise applications with their own requirements for security and privacy have become more prominent, and there are now thousands of both public and enterprise projects, directly or indirectly touching the lives of hundreds of millions of people.

One of the key technology foundations of DLT is public-key cryptography, in particular, elliptic curve cryptography. The most widely adopted elliptic curve in the DLT space by far is secp256k1 and the hash function keccak-256. Unfortunately, neither secp256k1 nor keccak-256, are endorsed in SP 800-186 and FIPS 186-5. This is despite the fact that there are no significant security differences between for example the NIST endorsed secp256r1 and secp256k1 or the sha3-256 hash versus keccak-256.

The current decision by NIST will have a significant impact on business in this space. Since any effort to cater to both the large global market for DLT applications based on secp256k1 and keccak-256 and customers who require NIST-compliance in their systems necessitates a far more complex programming effort in order to maintain multiple approaches to the same problem. The likely outcome of the resulting competing business requirements and continued technology uncertainty is increased slower development, reduced and delayed investment, and increased cost in order to reach some level of industry convergence. Furthermore, these developments could lead to either of two undesirable outcomes, market fragmentation into technology silos or technology stack and, consequently, vendor monopolies. Either outcome would lead to higher costs.

Most importantly, existing deployments of DLTs based on secp256k1 and keccak-256 already affect hundreds of millions of people, and those currently under development will affect even more, as detailed in the section on Industry Adoption and Impact below.

To minimize the damage to innovation and markets caused by the difference between the standards being adopted by the world and those currently endorsed by NIST, we request that NIST include the secp256k1 curve as part of the endorsed ECDSA schemes, and the use of keccak-256 in the secp256k1 signature schemes.

About secp256k1 and keccak

The curve secp256k1 is an elliptic curve of Koblitz type, defined in the Standards for Efficient Cryptography paper [SECG2]. It is currently used together with the ECDSA signature algorithm in order to create digital signatures. Other signature types like Schnorr can also be used with this curve, but these have not been widely deployed.

The security of general Koblitz type elliptic curves is covered in [SECG1], and the secp256k1 curve has a security level of 256 bits, which is considered secure.

The secp256k1 curve has been used extensively in the blockchain space, starting with the launch of Bitcoin in 2009 and also used as a core feature of Ethereum, which enables applications far beyond cryptocurrency. The security of this curve continues to be relied upon for billions of dollars worth of blockchain transactions daily.

The core reference library libsecp256k1 [libsecp256k1] has been tested extensively and has undergone thorough optimizations, which leads to the signature algorithm being very fast. Ethereum uses a hash function called keccak-256 which is used with secp256k1 signatures. This hash function was chosen due to the fact that it was the winner of the SHA3 competition. However, the final version of the SHA3 standard included an extra padding byte to the message before applying the keccak hash, which means that the keccak-256 hash function has a different output than the FIPS-approved SHA3. The only difference between these functions, however, is the extra padding of the message.

Industry Adoption and Impact

Below we will be delineating the extent and importance of the usage of secp256k1 and keccak-256 across all industry verticals by detailing current and expected (2020) usage patterns and user basis for the currently predominant industry verticals and cross-industry functions.

Decentralized Identity

Decentralized identity has the potential to become the first universal digital identity for individuals, legal entities and things. It dramatically increases the user's privacy while creating new revenue channels for companies and government, and reducing costs for consumers of digital identities. Incubated over a period of years and tested in numerous companies (including Fortune 500) and consortia across many industry verticals around the world, the recently approved W3C Verifiable Credentials standard [W3C.VC] paves the way for major adoption in production systems.

We are very pleased that the United States, e.g., NIST [NIST], Department of Homeland Security (DHS) [DHS], recognizes the great value of this new identity management paradigm based on the W3C Verifiable Credentials standard [W3C.VC]. Other governmental and public sector organizations/ initiatives are investing a lot of effort to explore these new technologies, including the European Commission [EC][ESSIF], Spain's national Alastria network [Alastria], The UK's Financial Conduct Authority [FCA] and the Government of British Columbia [VONX].

In addition, existing trust anchors such as the Global Legal Entity Identifier Foundation (GLEIF) are partnering with decentralized identity platform providers to issue W3C Verifiable Credentials to legal entities and their corporate officers [GLEIF]. Amongst others, some platforms anchor

DIDs on the Ethereum or Quorum network which is based on secp256k1/ keccak-256 cryptography.

Generally speaking, secp256k1 is very popular in the decentralized identity community for authentication purposes. For this reason, support for secp256k1 is crucial to stay interoperable in this open standards-driven ecosystem. Many decentralized identity projects use the decentralized and immutable nature of blockchains in order to add integrity protection to decentralized identifiers and their associated public keys. These projects mainly use hash functions SHA2-256, RIPEMD-160 and keccak-256 as hash functions.

Without official endorsement, public sector applications will not be able to make full use of the above efforts and systems.

Trade and Supply Chain

The trade industry is moving quickly to leverage blockchain for trade finance, shipping and freight, digitization of documents, and maintaining expansive networks. One example of a platform in production leveraging the Ethereum-based Quorum blockchain infrastructure is Komgo, a decentralized commodity trade finance network. Investors and shareholders of the company include Citi, ING, Credit Agricole CIB, BNP Paribas, Societe Generale, ABN Amro, Macquarie, MUFG, Natixis, Rabobank, Gunvor, Mercuria, Koch, Shell, and SGS, which has already channeled more than $1 billion of financing on the platform.

Within supply chain management - retail, manufacturing, and logistics - many companies have begun using blockchain solutions for traceability, transparency, and efficiency in their processes. Treum, which leverages the ethereum blockchain, builds asset and industry agnostic supply chain solutions, including Food, Consumer Products, Oil & Gas, Healthcare, Luxury Goods, Energy, Land, and Art. Companies that have tested supply chain solutions include Glaxo Smith Kline, Proctor and Gamble, Johnson and Johnson, Mars, and many others.

Financial Services

In Deloitte’s 2018 global blockchain survey, which drew responses from 1,053 executives across seven countries, 74 percent reported that their organizations see a “compelling business case” for using blockchain technology. In 2019, JP Morgan created their stable coin, Fidelity launched its digital asset custody service, and aims to roll out a crypto trading service for its clients, State Street Bank is investing in research and development for digital assets, stablecoins, custody, and the USC initiative [the Utility Settlement Coin being developed by bank consortium Fnality. These are a few of the many banks globally working on solutions for capital markets, investment management, payments and remittances, treasury liquidity and foreign exchange, and insurance.

Government: Access Control and Credential Management

The US, UK, Canadian, the United Nations, and international non-governmental organizations such as the World Bank and the Inter-American Development Bank are evaluating the use of decentralized identity solutions for credential management, access control, and track and trace of government-issued payments.

Telecommunication

A consortium of global telecommunications carriers comprising roughly 80% of global voice and data traffic is creating a global DLT network in 2020 comprised of several DLT stacks including Enterprise Ethereum which utilizes secp256k1 and keccak-256. The DLT network is to financially settle inter-carrier voice and data transactions[1] of their several billion clients and provide an identity, compliance, and reputation layer for participating carriers and their authorized delegates. Besides improving inter-carrier voice and data-on-demand settlement speeds saving billions of dollars for carriers globally, the applications will allow for the 1st time to introduce carrier reputation, battling global carrier fraud which impacts not only carrier bottom lines globally to the tune of several billion dollars a year but also virtually every telecom customer through dropped or not completed calls. While carrier customers are not directly using secp256k1 and keccak-256, the carriers do so on behalf of their customers during inter-carrier voice and data-on-demand settlements when utilizing voice and data-on-demand settlement solutions.

In addition, telecom regulatory authorities around the world are starting to mandate the usage of blockchain/DLT technology in their regulatory frameworks such as the Telecom Regulatory Authority of India (TRAI) mandating the usage of DLT technology to prevent text messaging spam in 2018 [TRAI].This directly impacts over 1 billion Indian mobile customers. In fact, the Tech Mahindra implementation of the Anti-spam TRAI requirement currently reaching about 300 million indian mobile users is based on the Nexledger which is an Ethereum-compatible Blockchain using secp256k1/ keccak-256.

Mobility

Similar to efforts in the telecom industry vertical, there is an effort underway in the mobility industry vertical by members of the Mobility On the Blockchain Initiative (Mobi) to create a global DLT network in 2020 consisting of global vehicle manufacturers such as GM, Ford, BMW, Honda, etc. and vendor organizations such as Accenture as consultancies or Microsoft as product companies. The global DLT network is intended to be comprised of several DLT stacks including Enterprise Ethereum which utilizes secp256k1 and keccak-256. The network will first provide verifiable identities and credentials of vehicles as well as an identity, compliance, and reputation layer for participating carriers and their authorized delegates. This will enable

[1] A voice call or data connection between a customer's device and an endpoint such as a smartphone or a website or mobile app server might traverse several carrier networks and incur charges on each leg of the voice or data journey which

real-time registration and verification of vehicles saving billions of dollars in manual processes globally. In addition, the DLT network intends to use utility tokens such as asset-backed stable coins for service payments by a vehicle or tokens issued by municipalities representing access rights for things such as neighborhood parking or congestion pricing.This will require vehicle buyers to use secp256k1 and keccak-256 directly through tokens and indirectly through verifiable vehicles identities and associated credentials. Given that there are over 1.2 billion vehicles globally, 64 million connected cars are to ship in 2019 and mobility IoT services such as Lime, Bird, Ofo or Blue Bike are rapidly increasing in popularity and thus the size of both fleet and customer base at a global level -- Lime reached the 50 million trip mark in significantly less than half the time (~ 2 years) than Uber did -- the DLT network is expected to reach over 100 million vehicle identities and several million token transactions in 2020.

Consumer Products - Entertainment, Music, Sports, Fashion, CPG and other Retail

Endconsumer focused products (B2C or B2B2C) are different to the B2B verticals discussed above because of the very different problems they solve: Customers or Fans of brands demand personalized and unique experiences any time, anywhere, on any device. In addition, we have an increasingly fragmented and saturated advertising landscape which together with siloed customer systems prevents brands from effectively reaching, engaging, and understanding their target audience. New end consumer focused, blockchain enabled solutions such as Sorare, Socios or Kapture are starting to address this need, albeit in very different ways though typically it involves combining several technologies such as Augmented Reality, Machine Learning and Social Media with DLT technologies. With very large brands in different verticals such as Sports -- the NBA, the Los Angeles Dodgers, the Sacramento Kings, Juventus Turin, Manchester United, FC Barcelona -- or Entertainment -- Warner Brothers, Capitol Records -- or retails brands such as Anheuser Busch engaged in this area, the number of consumers directly touched by these products, in particular through social media with influencer marketing, is expected to reach 100M+ in 2020. For example, one anticipated pilot in India around a well-known sports franchise can easily reach a few hundred thousands per mobile media event through social media sharing, and, thus, the pilot could easily engage over a million sports fans.

Given that most of the above mentioned end consumer products are built on either Enterprise Ethereum, public Ethereum or Ethereum-like chains, the situation in terms of impact of the usage of secp256k1 and keccak-256 is very similar in terms of impact on end users as for the above mentioned, primarily B2B verticals; in particular mobility, given the required usage of wallets for digital assets such as stable coins, utility tokens, loyalty tokens or digital collectibles.

Industry Standards Adoption

The following is a non-exhaustive list of standards and specifications that recognize secp256k1:

- As a proof algorithm in W3C Verifiable Credentials Standard [W3C.VC]
- As a proof algorithm for DID for W3C Decentralized Identifiers [W3C.DID] (future standard)

- As the signature algorithm of authenticators in the FIDO 2.0/ W3C WebAuthn Standard [WebAuthn]
- Signature algorithm for the COSE/ JOSE family [JOSE]
- JSON-LD Linked Data Signature specification based on secp256k1 [JSON-LD]
- EEA Ethereum Enterprise Client Specification V4.0 [EEA.Client]
- EEA Off-Chain Trusted Compute Specification V1.1 [EEA.TC]
- ...

Independent Implementations

The following is a non-exhaustive list of independent cryptography and related libraries with support for secp256k1:

- OpenSSL CLI tool and Open Source cryptography library [OpenSSL]
- Bouncy Castle cryptography library for JAVA applications [BouncyCastle]
- Node.js native cryptography library [Node.Crypto]
- Secp256k1 reference implementation in Bitcoin [libsecp256k1]
- “jose” which is a Node.js JOSE library [Node.JOSE]
- Nimbus JWT library for JAVA applications [Nimbus]
- JWT library based on Decentralized Identifiers in JavaScript [DID.JWT]
- JSON-LD Linked Data Signatures in JavaScript [JSON-LD.Lib]
- …

Please note that many of these libraries have significant industry adoption and use.

Organizations supporting this Letter

Consensys

Web: https://consensys.net/
ConsenSys is solving real-world problems with Ethereum blockchain solutions for organizations of all sizes, from the local community to the global enterprise.

Decentralized Identity Foundation (DIF)

Web: https://identity.foundation/
DIF has approximately 90 member companies such as Consensys/ uPort, Microsoft, Mastercard, Accenture, and many more. DIF and a lot of their members adopted secp256k1 and keccak-256 in their specifications in the area of decentralized identifiers, authentication and verifiable credentials exchange. Endorsing secp256k1 and keccak-256 officially by FIPS 186-5

and SP 800-186 will allow many decentralized identity solutions to be adopted by the public sector and it will ensure the public sector will be able to interact with decentralized identity applications in the private sector in the future.

Enterprise Ethereum Alliance (EEA)

Web: https://entethalliance.org/
The Enterprise Ethereum Alliance is a member-driven standards organization, with approximately 200 organizations, whose charter is to develop open blockchain specifications that drive harmonization and interoperability for businesses and consumers worldwide. The global community of members is made up of leaders, adopters, innovators, developers, and businesses who collaborate to create an open, decentralized web for the benefit of everyone.

W3C Credentials Community Group

Web: https://www.w3.org/community/credentials/
With approximately 320 members, the mission of the W3C Credentials Community Group is to explore the creation, storage, presentation, verification, and user control of credentials. The W3C CCG serves an important role in the incubation of new specifications and reference implementations in the decentralized identity space.

Hyperledger

Web: https://www.hyperledger.org/
Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology.

Other W3C Member Organizations

Microsoft Web. https://www.microsoft.com
Transmute Web: https://transmute.industries
ArcBlock Web: https://www.arcblock.ioTBD

References

[SECG1]	http://www.secg.org/sec1-v2.pdf
[SECG2]	http://www.secg.org/sec2-v2.pdf

[NIST]	A Taxonomic Approach to Understanding Emerging Blockchain Identity Management Systems, NIST: https://csrc.nist.gov/publications/detail/white-paper/2019/07/09/a-taxonomic-approach-to-understanding-emerging-blockchain-idms/draft
[DHS]	News Release: DHS Awards $198K for Raw Material Import Tracking Using Blockchain, DHS: https://www.dhs.gov/science-and-technology/news/2019/11/08/news-release-dhs-awards-198k-raw-material-import-tracking
[TRAI]	Tech Mahindra launched Blockchain solution to curb spam calls in India: https://telecom.economictimes.indiatimes.com/news/tech-mahindra-launched-blockchain-solution-to-curb-spam-calls-in-india/69147376
[EC]	Blockchain and Digital Identity, European Blockchain Observatory and Forum: https://www.eublockchainforum.eu/sites/default/files/report_identity_v0.9.4.pdf?width=1024&height=800&iframe=true
[ESSIF]	European Self-Sovereign Identity Framework: https://www.eesc.europa.eu/sites/default/files/files/1._panel_-_daniel_du_seuil.pdf
[GLEIF]	https://medium.com/uport/uport-partners-with-the-gleif-network-to-launch-decentralized-corporate-identity-management-2a7a20be3354
[W3C.VC.UseCase]	Verifiable Credentials Use Cases, W3C VC WG: https://www.w3.org/TR/vc-use-cases/#legal-identity
[W3C.VC]	Verifiable Credentials Data Model, W3C VC WG: https://www.w3.org/TR/vc-data-model/
[W3C.DID]	Decentralized Identifier Specification, W3C DID WG: https://www.w3.org/TR/did-core/
[Alastria]	Alastria ID: https://alastria.io/en/id-alastria/
[komgo]	What is komgo? | Commodity Trade Finance Meets Blockchain https://www.tradefinanceglobal.com/posts/what-is-komgo-commodity-trade-finance-meets-blockchain/
[VONX]	Verifiable Organizations Network, Government of British Columbia: https://vonx.io/about/

[FCA]	Regulatory sandbox - cohort 5: https://www.fca.org.uk/firms/regulatory-sandbox/cohort-5
[WebAuthn]	Server Requirements and Transport Binding Profile: https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-server-v2.0-rd-20180702.html
[JOSE]	https://tools.ietf.org/html/draft-ietf-cose-webauthn-algorithms-03
[EEA.Client]	https://entethalliance.org/wp-content/uploads/2019/11/EEA_Enterprise_Ethereum_Client_Specification_V4.pdf
[EEA.TC]	https://entethalliance.org/wp-content/uploads/2019/11/EEA_Off-Chain_Trusted_Compute_Specification_v1.1.pdf
[OpenSSL]	Command Line Elliptic Curve Operations, OpenSSL: https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations
[BouncyCastle]	The Legion of the Bouncy Castle: https://www.bouncycastle.org/
[Node.Crypto]	https://nodejs.org/api/crypto.html
[Node.JOSE]	https://www.npmjs.com/package/jose
[libsecp256k1]	https://github.com/bitcoin-core/secp256k1
[Nimbus]	https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-es256k-signature
[DID.JWT]	https://github.com/decentralized-identity/did-jwt
[JSON-LD]	https://w3c-dvcg.github.io/lds-ecdsa-secp256k1-2019/
[JSON-LD.Lib]	https://github.com/digitalbazaar/jsonld-signatures

Received: January 28, 2020
Status: Posted
Posted: January 29, 2020
Tracking No. 1k4-9epa-4f8i
Comments Due: January 29, 2020
Submission Type: API

Alexander, Ken

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-2019-0004-0007
Comment on FR Doc # 2019-23742

Submitter Information

Name: Ken Alexander
Email: ken.alexander@faa.gov
Organization: Federal Aviation Administration

General Comment

In June 2016, the FAA submitted the attached comment on Draft FIPS 186-4 requesting the addition of Schnorr 384. There is still considerable interest within the international civil aviation community in adding authentication to SBAS broadcast services and interest in specifying a signature defined by a recognized cryptographic standards body. We respectfully request that (EC) Schnorr 384 again be considered for addition into FIPS 186-5.

Attachments

U.S. Department
of Transportation
**Federal Aviation
Administration**

Dr. Lily Chen
Project Leader, Computer Scientist
Computer Security Division
Cryptographic Technology Group
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, MD 20899

Re: Please consider the Schnorr 384 digital signature as part of FIPS-186

Dr. Chen:

Thank you for your interest in cryptographic signatures for aviation broadcast systems that support air (and maritime) navigation. Indeed, some of these systems, like the Wide Area Augmentation System (WAAS), broadcast safety critical navigation at 250 bits per second from geostationary satellites to aircraft spread over North America. Today, WAAS is carried by over one hundred thousand aircraft. Similar systems exist in Europe, India, and Japan. China, Russia, and several other States are also developing or planning such systems. Taken together, these systems are known as satellite based augmentation systems (SBAS), because they augment satellite navigation systems, like the Global Positioning System (GPS), with extra data needed to compensate for GPS errors and ensure flight safety.

As mentioned above, SBAS broadcasts data at 250 bits per second. More specifically, SBAS break the data into 250-bits messages that broadcast once per second. The data payload in each of these messages is 212 bits. Two successive message carry 424 bits of data. If a bad actor were to replace the data in one of these messages with malicious data, then it could introduce potentially hazardous navigation data into the avionics.

For this reason, we have worked with Professor Dan Boneh to identify a best-of-class cryptographic signature for WAAS and SBAS. Since some of the SBASs are already operational, our choices are limited. Introduction of new data must be done with great care, given any change cannot affect the performance of existing avionics. We feel that we could insert two security messages every 100 seconds with signatures limited to a length 424 bits or less. For this reason, Professor Boneh recommended the length 384 Schnorr sequence. He points out that it could fit within two WAAS messages. He also points out that the Elliptic Curve Digital Signature Algorithm (ECDSA) of equal strength would require 384+128 bits. In short, Schnorr would occupy 2 percent of our broadcast capability, while ECDSA would occupy 3 percent of our broadcast capacity. We much prefer the Schnorr sequence if a decision is made to implement Authentication on WAAS.

In addition, Professor Boneh points out that the generation of the Schnorr key can be distributed over several servers. Such a strategy compels an adversary to access several servers to obtain (‘steal’) the key. ECDSA does not appear to have this property. This multiple server opportunity exists in WAAS and therefore this capability is of interest to the FAA.

Please understand neither the FAA nor the international civilian air community has made a final decision on the introduction of cryptographic signatures into the SBAS data streams. Even so, our interest is strong and the Schnorr 384 sequence is well designed for our WAAS SBAS application. Hence, we ask NIST to consider Schnorr 384 as part of FIPS-186.

Sincerely,

eSigned Jun 28, 2016

Deborah Lawrence
Manager, Navigation Programs

eSigned Jun 28 2016

Ken Alexander
Navigation Team Lead
Aircraft Certification Service
Navigation and Flight Deck Technologies

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

$$\begin{tikzcd}\boxed{\text{design choices}} \arrow[r, "\text{implementation processes}"] & \boxed{\text{implementations}}\end{tikzcd}$$

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

$$\begin{tikzcd} \boxed{\text{design choices}} \arrow[r, "\substack{\text{magical subset of}\\ \text{implementation processes}}"] & \boxed{\substack{\text{secure}\\ \text{implementations}}} \end{tikzcd}$$

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 "message 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 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.
```

Boyle, Vincent

From: Boyle, Vincent M <vmboyle@nsa.gov>
Sent: Thursday, March 5, 2020 10:58 AM
To: fips186-comments
Subject: Comment on Draft FIPS 186-5

Hi,
Sorry we are so late. If possible, please accept the following comments from NSA's Center for Cybersecurity Standards.

Thanks,
Mike Boyle

Comments on FIPS 186-5

Technical Comments:

5.4 step (h), first and second bullets and accompanying footnote. Per RFC 8017 (PKCS #1 v2.2) the hash is immediately preceded by the digest algorithm identifier. FIPS 186-5 should require confirming that the encoded digest algorithm identifier is correct *in addition* to confirming that the hash is located in the least significant bytes. Recommend the following rewording:

For RSASSA-PKCS-v1.5, when the hash value is recovered from the encoded message EM during the verification of the digital signature^1, the extraction of the ASN.1 value of the DigestInfo data structure shall be accomplished by either:

1) Selecting the appropriate number of rightmost (least significant) bits of EM, as determined by the size of a PKCS#1-defined ASN.1 DER value corresponding to the expected hash function's algorithm identifier and output length, regardless of the length of the padding,

or (if the DigestInfo is selected by its location with respect to the last byte of padding),

2) checking that a byte string of the length expected for the ASN.1 DER value of DigestInfo fills the remaining rightmost (least significant) bytes of EM (i.e., no other information follows the DigestInfo data structure in the encoded message).

Only if the extracted DigestInfo has the appropriate form shall the signature verification process continue. Assuming that this is the case, the following two checks should be performed:
The algorithm identifier extracted from DigestInfo shall be examined to verify that the expected (approved) hash function has been identified. The length of the digest value that is extracted from DigestInfo shall be determined and verified to be equal to the length of hash values output by the expected hash function. Only upon successful verification of both the algorithm identifier and the length of the digest value, shall the extracted digest value be used as the recovered hash value during the verification of the digital signature.

Footnote 1: PKCS #1, v2.2 (Section 8.2.2) provides two methods for comparing the DigestInfo values: by comparing the messages EM and EM' or by applying (a not specified) decoding operation. Step (h) above applies to the latter case.

5.4 item (g). Item (g) imposes the constraint 0 <= sLen <= hLen. This inequality should also be checked during the signature verification process, where hLen is determined by the expected (approved) hash function and sLen is the actual byte length of the byte string following the leftmost (most significant) nonzero byte (which should be 0x01) in the recovered DB.

A.1.1. condition 2(b) and (c) on the primes p and q. The most significant bit will always be set. Do you mean to say, the 2nd most significant bit may be set? Further algorithms don't appear to address when the second most significant bit may be set... care needs to be taken as setting this bit may destroy needed congruence conditions used in the primality proofs.

A.1.2.2 step 4 checks that len(seed) != 2*security strength, so the seed must have the high bit set, else the algorithm throws an exception. Consider setting the high bit of the seed in the Get the Seed generation process (A.1.2.1) so as to avoid half of the generated seeds from being tossed.

Appendix B4 - Checking for a Perfect Square, p. 61 (used in B.3.3 - General Lucas Probabilistic Primality Test)

Comment 1. The given algorithm uses rational arithmetic and testing of rational inequalities. It may be preferable to drop the implicit requirement that devices adhering to the DSS Standard be designed to perform such operations.

Comment 2. In the sequence {x_i} of rational approximations of sqrt(C), the bit length of the numerator of x_i doubles at each iteration as a result of the squaring operation in step 5.2. For example, starting with an input C with len(C) = 1024, growth such as

len(numerator(x_i)) = 512, 1024, 2046, 4092, 8184, 16368, 32736, 65472, 130943

is typical. It is even common for one more step to be needed, in which case the length is like 260000. This makes the algorithm impractical.

We suggest replacing the algorithm with newtonl, which is essentially the same as (1) the algorithm proposed for Standard X9.80, and (2) Algorithm 1.7.1, page 38, in H. Cohen, "A Course in Computational Algebraic Number Theory," Springer, 2000.

newtonl(n):

```
    m = len(n)
    b = (m + 1) div 2
    x = 2^b
    y = (x + (C div x)) div 2

    while y < x do
        x = y
        y = (x + (C div x)) div 2

  if x^2 = C
        status = PERFECT SQUARE
    else
        status = NOT A PERFECT SQUARE
    return status.
```

B.7 trial division step 2 the statement "For example, rather than preparing a table of primes, it might be more convenient to divide by all integers except those divisible by 3 or 5." Note still necessary to check if c is divisible by 3 or 5. Also, do you want to check divisibility by composites or do you really want to take GCDs?

General Comments:

A.1 inconsistent treatment of seeds/entropy:
Seeds are treated differently in the generation of provable primes and probable primes. For provable primes, A.1.2.1 Get the Seed is invoked where the seedlen is set to be twice the security strength. Further, there is no mention of entropy. For the probable methods, section A.1.3 requires the entropy to be at least the security strength of the DRBG and that the minimal length of the seed shall be at least the number of bits in the entropy; section A.1.6 makes no mention of entropy or seed lengths. Suggest the written requirements on entropy and seedlength be uniform across the different prime generation methods. This might be easiest done by adding a statement to the last paragraph of section 5.1 RSA Key Pair Generation.

Editorial Comments:

2.1 pg. 5 "Public Key"
Change "digital signature that was signed" to "digital signature that was generated."

"Security strength"
Consider stating what it means to "break a cryptographic algorithm or system"

"Signature validation"
Since this is different from "Signature verification," rewrite definition without using the phrase "verification of the digital signature"

"Signature "verification"
The definition "The process of using a digital signature algorithm and a public key to verify a digital signature on data" is circular.

"Trusted third party" XXXXXX
Consider expanding the phrase "other than the owner and verifier" to specify "owner of ..." and "verifier of ..."

2.3 p. 7 "n"
Change "the bit length on n" to "len(n)," which is defined three items before.

3.1 p. p. 10, paragraph 2
Consider replacing "Anyone can verify a correctly signed message using the public key" with "Anyone can verify a correctly signed message using the message digest and the public key." Suggested change reflects the diagram on p. 9.

3.1 p. 10, paragraph 4 - There is a syntax error in the sentence,

"A verifier requires assurance that the public key to be used to verify that a signature belongs to the entity that claims to have generated a digital signature (i.e., the claimed signatory)."

Replace with

"A verifier requires assurance that the public key that is used to verify a signature actually belongs to the entity that claims to have generated a digital signature (i.e., the claimed signatory)."

3.1 p. 10, paragraph 4 - The next sentence might not be as intended. Maybe replace "public/private key pair used to generate and verify a digital signature" with "private key used to generate a digital signature."

3.1 p. 10, Item 1 - The third sentence, "If a verifier does not..." is 70 or 71 words long, and is difficult to understand. Consider replacing with "That is, a verifier has assurance of the data's integrity, but does not have assurance of source authentication."

3.1 p. 10, Item 2 - Replace "used to verify that a signature is not mathematically valid" with "used to verify a signature is not mathematically valid." This seems to be the same kind of miswording as in p. 10, paragraph 4, mentioned above.

----

5 p. 15 Later in Chapter 6 (ECDSA), it is explained that a "signature" is a certain pair of integers (r, s), and the algorithm to generate (r, s) is included. Chapter 5 contains no such information. Perhaps mention what an RSA signature is, and include a sentence "The RSA Algorithm can be found in..." The opening sentence, "The use of the RSA algorithm..." appears to address other concerns.

5.2 item 1 states that the public key shall be used only for signature verification. Public key should be switched to public encryption exponent as the public modulus is also needed in signature generation.

5.2 item 5. Remove this item as there are no domain parameters for RSA. Note section 3.1 specifically states "Note that the RSA algorithm does not use domain parameters."

5.4 item (b) May want to ensure use of SHAKE128 and SHAKE256 is consistent with draft-ietf-lamps-cms-shakes-13 "Use of the SHAKE One-way Hash Functions in the Cryptographic Message Syntax (CMS)". The draft notes in section 4.2.1 that MGF1 is not used as the MGF, but that SHAKE is used naively as the MGF. Perhaps this should be specified in FIPS 186-5. This may require specifying additional parameters such as output length, etc.

5.4 footnote 1 should refer to step (h) and not (f).

6.3 p. 22 first paragraph

"Both k and k^(-1) may be pre-computed since knowledge of the message to be signed is not required from the computations"

The statement holds only for (non-deterministic) ECDSA, whereas this section addresses per-message secret number generation for both ECDSA and Deterministic ECDSA. Consider adding a note that statement applies only to non-deterministic ECDSA.

6.4.1, p. 23, Step 1
Replace ceiling(log_{2} n) with len(n).

Rationale: len(n) is a "defined mathematical symbol" on p. 7. Also, in practice, len(n) can be accessed directly, whereas log_{2}(n) is a computation, requiring care with decimal precision.

6.4.1 p. 23 item 11
If the rare event r = 0 or s = 0 is triggered for Deterministic ECDSA, consider creating a mechanism to extract additional ephemeral values out of the in technique A.3.3 rather than return a failed message.

A.1.2.2 uses as input the seed obtained from method in B.3.2.1; this section does not exist. Should it be A.1.2.1?

A.1.2.2 step 6.1 refers to C.10, this should be B.10.

A.2.1 process item 3

Change reference Table B.2 to Table A.2.

A.2.2 process item 4

Reference to B.6.2 is incorrect. Should it be A.4.2?

B.10 step 6 should reference B.6 and not C.6

Celi, Christopher

From**:** Celi, Christopher T. (Fed) <christopher.celi@nist.gov>
Sent: Friday, November 15, 2019 3:24 PM
To: fips186-comments fips186-comments@nist.gov
Subject: Comment on FIPS 186

Hello,

Thank you for publishing this draft. Here are my comments as an implementor for the CAVP. Naturally the CAVP must cover all possible implementations of NIST cryptographic standards.

5.1 RSA Key Pair Generation

- 4096 and 8192 bit keys are specified for KAS-IFC in SP800-56B but their security strengths do not appear in the corresponding table within SP800-57 Part 1. I recommend including those values here or specifying that their effective security strengths correspond to the next lowest key size that appears in Table 2 in SP800-57 Part 1.

- Hash function for both key generation and signature generation MUST be have at least the security strength of the key size. This is not currently in ACVP testing for FIPS186-5.

5.4 PKCS

- This allows a wide enumeration of possibilities for ACVP testing. The "Hash" function can differ than the function used for MGF1 within PSS.

- In requirement (f) specify SP800-90A for an approved DRBG.

7.4 EdDSA Key Pair Generation

- In step (1) specify SP800-90A for an approved DRBG.

Appendix A.1.3

- For probable prime generators, perhaps provide guidance on how to proceed when the conditions within step (4.7) and step (5.8) are met.

Appendix A.3.3

- Are there any limitations on the Hash used within the HMAC-DRBG construction outside of the guidance of SP800-90A? How does this affect testing within ACVP when multiple (different) hash functions could be needed to generate a Det-ECDSA Signature with an existing key.

Appendix B.3

- Are the different tables (Table B.1 and Table B.2) used in practice? Can this be consolidated into a single table with only one set of values? I.E. always take the stronger number of iterations, or always take the faster number of iterations.

- If it is stated that "Tables B.1 and B.2 list the minimum number of iterations of the Miller-Rabin tests that shall be performed," then why not just include a single table of values being the minimum between Tables B.1 and B.2?

- The chance that a value is found that would pass a Miller-Rabin test from one table and fail from the other is still (much, much) less than the chance that a false-positive is found from the weaker iterated Miller-Rabin test.

- The Lucas test is not required if the Miller-Rabin test is performed. The language in this section implies but does not state otherwise.

> As stated in Appendix E, if the definition of the error probability that led to the values of the number of Miller-Rabin tests for p and q in Tables B.1 and B.2 is not conservative enough, the prescribed number of Miller-Rabin tests can be followed by a single Lucas test. Since there are no known non-prime values that pass the two-test combination (i.e., the indicated number of rounds of the Miller-Rabin test with randomly selected bases followed by one round of the Lucas test), the two-test combination may provide additional assurance of primality over the use of only the Miller-Rabin test. However, the Lucas test is not required when testing the p1, p2, q1, and q2 values for primality when generating RSA primes.

Appendix E

- Please correct "Aonstructing primes with congruence conditions mod 8 are allowed."

Thanks,
Chris Celi

Received: January 18, 2020
Status: Posted
Posted: January 29, 2020
Tracking No. 1k4-9eip-3krn
Comments Due: January 29, 2020
Submission Type: API

Chalkias, Konstantinos

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-2019-0004-0006
Comment on FR Doc # 2019-23742

Submitter Information

Name: Konstantinos Chalkias
Email: kostascrypto@fb.com
Organization: Calibra / Facebook

General Comment

There is a confusing symbol inconsistency around the signature components R and S of the EdDSA scheme.

Note that the algorithm spec description for signature output components are denoted with different symbols between the two schemes. More specifically, for ECDSA signature (r, s) are used, while for EdDSA it is (R, S). However, across the document, EdDSA's signature components are sometimes mistakenly denoted with lowercase r and s, which can cause confusion, especially because the lowercase s happens to also denote the private key part of the signer, while lowercase r is also a secret value in EdDSA.

Examples:

a) Section 2.3 Mathematical Symbols (page 7)
"An ECDSA, or EdDSA digital signature, where r and s are the
digital signature components."

Recommendation: Either use R, S for both algorithms across the document or distinguish between ECDSA and EdDSA.

b) Section 7.7 EdDSA Signature Verification (page 30) in the first bullet point of Process.
"Decode the first half of the signature as a point R and the second half of the signature as an

integer s. Verify that the integer s is in the range of 0 s < n."

Recommendation: change s to S.

c) Section 7.8.2 HashEdDSA Signature Verification (page 32) in the first bullet point of Process. "Decode the first half of the signature as a point R and the second half of the signature as an integer s. Verify that the integer s is in the range of 0 s < n."

Recommendation: change s to S.

Costa, Graham

From: COSTA Graham <graham.costa@thalesgroup.com>
Sent: Wednesday, January 29, 2020 7:50 AM
To: fips186-comments <fips186-comments@nist.gov>
Cc: GOUGET Aline <aline.gouget@thalesgroup.com>; SecurityCertifications@gemalto.com <SecurityCertifications@gemalto.com>; BURNS Robert <robert.burns@thalesgroup.com>
Subject: Comment on Draft FIPS 186-5

Hi,

Please find below 2 comments on the proposed draft of FIPS PUB 186-5.

Independent of the comments, Thales is supportive of the draft and is keen for its final publication in order to allow for its inclusion in FIPS 140-2 Annex A. This will allow edDSA and deterministic ECDSA to be permitted for use as approved security functions in FIPS 140-2 certified cryptographic modules.

Comment 1: In relation to DSA the following statements are made in the current 186-5:

- Section 1: *"(1) The Digital Signature Algorithm (DSA) is no longer specified in this standard and may only be used to verify previously generated digital signatures. Complete specifications may be found in Federal Information Processing Standard (FIPS) 186-4."*
- Section 4: *"Prior versions of this standard specified the DSA. This standard no longer approves DSA for digital signature generation. DSA may be used to verify signatures generated prior to the implementation date of this standard."*
- Appendix E: *"DSA is no longer approved for digital signature generation. DSA may be used to verify signatures generated prior to the implementation date of this standard."*

Although not opposed to the removal of DSA from 186-5, Thales is concerned as to how any transition in related standards is managed. i.e. Whether 186-4 will continue to be listed in FIPS 140-2 Annex A following publication of FIPS PUB 186-5 and for how long?

In general – HSM's, tokens and cryptographic modules support DSA and as such it 'may' still be being used by some customers albeit we are unaware of any standards where DSA is the only signature option available.

In light of above – we would recommend softening statements in section 1, 4 and E to simply say that 'DSA is no longer defined in 186-5' and to refer to 186-4 for its definition. This then allows other standards such as SP800-131A2 as the formal reference for transitions to set out the conditions as and when DSA can be used without creating a conflict between 186-5 as an cryptographic specification and SP 800-131Ar2 as the formal transition roadmap.

Although not strictly relevant to FIPS PUB 186-5, we would separately recommend that when the standard is published, any immediate transition is both reflected in SP800-131Ar2 and where existing certified modules such as those approved under FIPS 140, are continued to be allowed to use DSA as defined in 186-4.

Note: Flip-side to not supporting an extended transition for DSA is that in practice, many federal users requiring DSA will stagnate on the firmware release they use in order to be operating in a FIPS compliant configuration (rather than shifting to later FIPS approved firmware but where a required algorithm has been dropped). This is counter to the goals of them adopting industry best practice and deploying the latest firmware where in many cases this may also include security fixes.

Comment 2: In relation to removal of the definition for the NIST binary curves – similar to above, whereas Thales don't oppose this change, we strongly suggest the text in the current draft is softened. In particular, the current draft includes the following statements:

- Appendix E: *"Elliptic curves defined over binary curves (specified in SP 800-186) are now deprecated"*

Referring across to terms used in SP800-131Ar2 – "deprecated" in particular is taken to mean that any "deprecated" algorithm is no longer allowed to be supported or used.

In this case, whilst most of the NIST ECC standards allow the use of curves beyond those specified in the NIST standards (provided at least one option from the NIST specification is supported) it seems inappropriate to use the term 'deprecated'.

As an alternative – it is recommended that Appendix E simply states that 'Elliptic curves defined over binary curves are no longer formally defined by NIST'.

The primary concern here is that reference to the curves being 'deprecated' in FIPS 186-5 could lead to certifications such as FIPS 140-2 from disallowing the certification of modules (as Level 3) where the optional use of binary curves is supported.

Should you have any comments or follow-on questions on above – please do not hesitate to contact us.

Kind Regards,

Graham Costa
SECURITY AND CERTIFICATIONS MANAGER
Tel.: +44 (0) 1276 608001
Mob.: +44 (0) 774 131 3657

Gemalto is now part of the Thales Group.
Please note that my new email address is graham.costa@thalesgroup.com

THALES

Rivercourt, 3 Meadows Business Park,

Blackwater, Camberley, Surrey, GU17 9AB.

www.thalesgroup.com

Dieguez, Ignacio

From: Dieguez, Ignacio <ignacio.dieguez@ncipher.com>
Sent: Friday, January 24, 2020 6:18 AM
To: fips186-comments <fips186-comments@nist.gov>
Subject: Comment on Draft FIPS 186-5

Dear NIST colleagues,

Please find attached nCipher Security comments on FIPS 186-5.

Kind regards,
Ignacio.

nCipher Security comments on FIPS 186-5 (Draft)

#	Type	Line#	Comment (Include rationale for comment)
1	Ed		FIPS 186-5 s7.2 typo, "•" (bullet) for "*" (asterisk)
2	Ge		• nShield HSMs use DSA (with L=3072, N=256) for firmware verification and to authenticate the HSM, users and applications, when operating in approved modes. • All nShield HSM users who require FIPS 140 or FIPS 186 compliance will be impacted by the removal of DSA. • I therefore call for a deprecation plan rather than immediate removal.
3	Te		FIPS 186-5 s7.2 In "For Ed25519, the most significant bit of the final octet is always zero, while for Ed448, the final octet is always zero.", I suggest the equivalent "most significant octet is always zero", to avoid a confusing inconsistency in language.
4	Ed		FIPS 186-5 A.2.2 step 4 refers to appendix B.6.2, but there is no such section. It probably means A.4.2.
5	Te		FIPS 186-5 A.2.2 and A.3.2 no longer include an explicit repetition when x > n-2 (or c > n-2 in the FIPS 186-4 language). This should be restored.
6	Ed		Typo in Appendix E: "Aonstructing primes" should be ""Constructing primes"
7	Ed		Typo in Appendix E. It references tables B.1, C.1 and C.2, but no such tables exist. It probably means Table A.1, B.1 and B2.
8	Ge		Given that FIPS 186-5 no longer specifies DSA, but SP 800-131rev2 doesn't include any transition for DSA, what is the deprecation plan / timeline for DSA in the future?

Giessmann, Ernst

From: Ernst G Giessmann <ErnstG.Giessmann@t-online.de>
Sent: Thursday, January 30, 2020 8:23 AM
To: fips186-comments
Subject: Comments on Draft FIPS 186-5

Ooops,
I missed the deadline.
Sorry for being late.
Kind regards,
/Ernst.

Remarks to the 186-5 Draft

Ernst G. Giessmann
giessman@informatik.hu-berlin.de

January 30, 2020

Please excuse for sending a $\LaTeX$ document instead of using a standardized form. Some of the editorial remarks require a clear font selection.

Technical Remark to Annex A.4

The text in clause A.4.1 announces an different algorithm than presented. It says that the output of an approved RBG in the interval $[0, N-1]$ will be converted into an integer in the interval $[0, n-1]$ "by simply reducing this output modulo n". The algorithm described later uses the reduction modulo $n-1$ together with a addition of one. The outputs of these algorithms are certainly different, but both are acceptable if the first is accompanied by a zero test. From an implementors point this is even a bit better, because it needs a cheap flag checking only, whereas the adding requires an extra access to the output data. The probability of an invalid output is negligible.

Proposal: Keep the text in the beginning and add an description for this algorithm, including, e.g.:

4. Set $x = x \bmod n$;
5. If $x = 0$, output INVALID;
6. Output x.

Almost Editorial Remarks

p. 8

⌊-2.1⌋ $= -3$

Comment: wrong font for the minus character, appears also later, e.g., page 27, use instead $\lfloor -2.1 \rfloor = -3$

$[n]X$

Comment: formally $[1]X$ is not defined hereby, add $[1]X = X$ for completeness

p. 17

“then either”

Comment: replace by “set either”

SHAKE128

Comment: check the font here, and also the font for $emLen$, $hLen$ and $dbMask$

p. 20

GF_p

Comment: replace it by $\mathrm{GF}(p)$ as elsewhere in the doc

p. 21

a new secret number k

Comment: add the restrictions $0 < k < n$ (the same applies on p. 23)

$1 = (k^{-1}k) \bmod n$

Comment: it is not an assignment, therefore the equivalence sign is needed here, use $1 \equiv (k^{-1}k) \bmod n$

p. 23

$(1/k)(\mathrm{mod} n)$

Comment: $(1/k)$ is not defined mod n, use $k^{-1}(\mathrm{mod} n)$

p. 25

$r\text{1}$ and r_1

Comment: missing subscript

p. 27

$p \equiv 3 \pmod{4}$

Comment: wrong boldface for **mod**

p. 28

$\bullet, {}^*$

Comment: different multiplication symbols, use a lowered $*$ or maybe a centered dot $\cdot$ instead (cf. p. 24)

p. 29

S is a little-endian encoded value

Comment: To deal with leading zeros it should clearly stated that S is not an integer value nor the octet encoding of an integer but an octet string of a fixed given length: "S is an octet string of a given length of a little-endian encoded integer."

The octet string S is the encoding of the resultant integer.

Comment: please add here "The octet string S of given length is the encoding of the resultant integer filled with zero bytes".

p. 30

$[2c * S]G = [2c]R + (2c * t)Q$

Comment: replace the uppercase S and the round brackets: $[2c \cdot s]G = [2c]R + [2c \cdot t]Q$ or at least $[2c * s]G = [2c]R + [2c * t]Q$ (applies also to p. 32)

p. 34

len(p_1) and len(p_1)

Comment: harmonize the font in the text

p. 58

Key = 0x00 0x00...0x00

Comment: too many 0x's here, use "Key = 0x00 00 00 ... 00"

p. 87

Aonstructing primes

Comment: misspelled "Constructing"

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.

Johnston, Anna

From: Anna Johnston <amj@juniper.net>
Sent: Wednesday, January 8, 2020 4:21 PM
To: fips186-comments <fips186-comments@nist.gov>
Cc: seth-staff <seth-staff@juniper.net>
Subject: Comments on FIPS 186-5

Please find comments on FIPS 186-5 in the attached document.

Thank you,

A. Johnston

Comments on FIPS 186.5: Digital Signature Standard

Anna M. Johnston
Juniper Networks
amj at juniper dot net

January 8, 2020

- Finite field based DSA should not be removed as a signature option. It is by far the more flexible and security agile of the existing choices:
  - Many attacks are based on the base field, be it in finite field cryptography or elliptic curves (NFS, index calculus for EC (verify if individual point based only), or the first attacks on binary elliptic curves)
  - For this reason the base field/EC/Ed) should be changed with some regularity;
  - Provable primes (Pocklington's theorem based) are much easier to generate than new elliptic/edwards curves;
  - Flexibility with group order in finite field based cryptography allow for a wide variation of prime types:
    - "Safe" primes: These have the maximal order subgroup with relation to the field size, with primes of the form $p = 2q + 1$ where q is also a prime (Sophie Germain) integer. Note that these primes are far less common than more randomized primes. They are harder to generate due to their rarity, and may be less secure due to the lack of entropy in each prime.
    - Security requirements for the base field size are generally much larger than the requirements for the subgroup size. Primes may be generated with cryptographically 'packed' finite field. A cryptographically packed finite field contains multiple subgroups of the required security size, allowing for the storage of multiple public keys in a single prime integer.
  - Primes generated using Pocklington's theorem are easy to validate, and the primality proof can be transmitted for regular, verifiable updates.
  - Finite field based cryptography is more resilient against quantum attacks than elliptic curve [5] and base field sizes can be increased (upping the cost of a

quantum attack) without as much of a computational hit as increasing elliptic curve base fields.

  - The mathematics behind EC/Ed cryptography is far more complicated than the mathematics behind finite field. This increases the probability of implementation and user errors.

- Pocklington's theorem [4] (1914) is at the core of most (if not all) existing provable prime generation algorithms, such as [1] (1975), [3] (1982), [6] (1986), [2] (1995).

- The attack based on the factorization of $(p+1)$ only works for factoring, not discrete logarithms. There are no discrete logarithm attacks based on the factorization of $(p+1)$. Adding the requirement that $(p+1)$ contains a large prime factor, therefore, is not required when generating primes for discrete logarithm based systems. In fact, in can be argued that adding this requirement is a security risk as it reduces the entropy in the prime.

References

[1] John Brillhart, D.H. Lehmer, and J.L. Selfridge, *New primality criteria and factorizations of* $2^m \pm 1$, Mathematics of Computation **29** (1975), no. 130, 620–647.

[2] Ueli M. Maurer, *Fast generation of prime numbers and secure public?? key cryptographic parameters*, Journal of Cryptology **8** (1995), 123–155.

[3] William J. Miller and Nick G. Trbovich, *Rsa public-key data encryption system having large random prime number generating microprocessor or the like*, 1982, US Patent assigned to Racal-Milgo Inc; expired 2000.

[4] Henry C. Pocklington, *The determination of the prime or composite nature of large numbers by fermat's theorem*, Proceedings of the Cambridge Philosophical Society, no. 18, University of Cambridge, 1914–1916, pp. 29–30.

[5] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter, *Quantum resource estimates for computing elliptic curve discrete logarithms*, 2017.

[6] J. Shawe-Taylor, *Generating strong primes*, Electronics Letters **22** (1986), 875–877.

Laing, Thalia May

From: Laing, Thalia May <thalia@hp.com>
Sent: Wednesday, January 29, 2020 2:57 PM
To: fips186-comments <fips186-comments@nist.gov>
Cc: Belgarric, Pierre <pierre.belgarric@hp.com>; Schiffman, Josh <joshua.ser.schiffman@hp.com>
Subject: Comments on FIPS 186-5 draft

Hello,

Please find below our comments on the NIST FIPS 186-5 draft.

In the draft, ECDSA can be implemented with either a random or a deterministic nonce (see Section 6.3, "The secret number k may be generated either randomly […] or in a deterministic way"). EdDSA is only implemented with a deterministic nonce (see Section 7.6, "EdDSA signatures are deterministic").

Ensuring the generation and use of good quality random nonces can be challenging from a practical implementation perspective. Deterministic nonces address these problems but may be vulnerable to a number of known physical implementation attacks, including side channel and fault injection attacks, such as in [1,2].

Is there room to include the standardisation of nonces that are deterministic but allow for some additional randomness or noise in both ECDSA and EdDSA? Such solutions could reduce the reliance on good quality randomness generation somewhat, whilst also addressing some physical attacks. Potential solutions along these lines have been proposed in the literature.

[1] Ambrose, C., Bos, J. W., Fay, B., Joye, M., Lochter, M., & Murray, B. (2018, April). Differential attacks on deterministic signatures. In Cryptographers' Track at the RSA Conference (pp. 339-353). Springer, Cham.
[2] Samwel, N., Batina, L., Bertoni, G., Daemen, J., & Susella, R. (2018, April). Breaking Ed25519 in WolfSSL. In Cryptographers' Track at the RSA Conference (pp. 1-20). Springer, Cham.

From,

Pierre Belgarric
Thalia Laing
Joshua Schiffman

Thalia Laing, PhD
Security Researcher
HP Labs

thalia@hp.com
T +445601090928
HP Inc.
2nd Floor
1 Redcliff Street
Bristol
BS1 6NP
UK

Received: November 13, 2019
Status: Posted
Posted: January 29, 2020
Tracking No. 1k3-9dah-ecx2
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-2019-0004-0003
Comment on FR Doc # 2019-23742

Submitter Information

Name: John Preu 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.

Comments on FIPS 186-5
- Section 1. Instead of writing Three techniques are approved and then listing 4 bullets, we would suggest that bullet (1) about DSA is made into a note and placed after the list.

- Section 1. The reference [31] does not exist in the Reference section. Even if the document is republished online, it may disappear again. Maybe only use RFC 8017 [15] as the only normative reference for PKCS#1 2.3.

- Section 1. The introduction only points to references for RSA, ECDSA, and EdDSA and says that they are approved. It would be good if the introduction also states that the document provides specification for the algorithms compatible with the specifications in the references. Especially for ECDSA where the reference is a withdrawn specification that used to be behind a paywall.

- The documents takes DSA as an example in several places (2.1, 3.3, etc.). I assume DSA should be removed.

- It would be good if NIST could provide a little more information and discussion regarding RSASSA-PKCS-

v1.5. IETF is currently working hard to deprecate all use of PKC#1v1.5 padding and BSI (Germany) is only allowing use of the PKC#1v1.5 padding until 2025. FIPS 186-5 only states that it applies additional restriction to RFC 8017 so the requirement RSASSA-PSS is REQUIRED in new applications. RSASSA-PKCS1-v1_5 is included only for compatibility with existing applications. still apply. Harmonization between various government organization and SDOs would be good for global industries implementing these standards. It would be valuable with more information on how NIST looks at the future for RSASSA-PKCS-v1.5.
https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/tr02102/index_htm.html

- We think it good that NIST is discussing side-channel and fault injection attacks on deterministic ECDSA. The discussion is however very focused on deterministic ECDSA. We think NIST should also shortly discuss side-channel and fault injection attacks on RSASSA-PKCS1-v1_5, RSASSA-PSS, and randomized ECDSA.

- Good that SHAKE128 and SHAKE256 is approved for use in different signature algorithms.

- We think it is good that NIST introduces deterministic ECDSA. Instead of listing deterministic ECDSA as a variant of the whole ECDSA algorithm, we think deterministic ECDSA should be listed as an alternative method for generating the number k.

- In addition to specifying a random k = rand() and deterministic k = f(M) ways to generate k, we would like NIST to specify a deterministic with noise k = f(M, rand()) method to generate k, i.e. a method that depends on the message but introduces some noise into the calculation. This is suggested in section 4.2 of https://eprint.iacr.org/2017/975.pdf and implemented in several places such as https://signal.org/docs/specifications/xeddsa/

- We would like to see more hash functions (or XOFs) in EdDSA. The current requirement to only use SHA-512 is a current and future problem for IoT deployments. Many IoT devices currently have support for SHA-256 but not SHA-512. In the future it would be very beneficial if these devices could use the same primitive for hash, MAC/PRF and AEAD. We suggest that NIST follows its practice for RSA and ECDSA to allow use of all approved hash and XOR functions. In particular, it would be beneficial to use SHA-256 and SHAKE128 in IoT deployments. In the future, the outcome of the lightweight IoT project should be useable in EdDSA. In the vast majority of IoT systems, even quite constrained ones, the current algorithms are not a problem, but being required to implement several algorithms is a substantial problem. NIST should strive to strive to have a set of algorithms all using a single primitive.

- It would be good if NIST explicitly states that Ed25519 is approved for use after 2030 when NIST only approves 128 bit security strength. Right now the document only states that It is noted that Ed25519 is intended to provide approximately 128-bits of security

- It is great that NIST is specifying ECDSA in FIPS 186-5 instead of relying on references behind paywalls. We think open access is very important for security specifications as history has showed over and over again that lack of analysis often lead to serious weaknesses.

Best Regards,
John Preu Mattsson, Senior Specialist, Ericsson

Moskowitz, Robert

From: Robert Moskowitz <rgm-sec@htt-consult.com>
Sent: Tuesday, November 19, 2019 9:19 PM
To: fips186-comments; Dang, Quynh H. (Fed)
Subject: Comments on FIPS186

This is a comment about the use of SHA512 for EdDSA25519.

In sec 7.1:

"For Ed25519, SHA-512 shall be used."

There are two important problems of using SHA-512 rather than SHAKE128.

First is that in highly constrained systems that use Keccak derived operations (from FIPS 202, sp800-185 and sp800-56Cr1) this may be the ONLY need of the older hash function code. Even 1K of code is of concern on many constrained systems. By this choice you are demanding inclusion of hash code that may be unique to this (EdDSA25519) function.

On the other side of the problem, inclusion of SHA512 could end up as a barrier to adopting Keccak hashes in highly constrained systems when they HAVE to support SHA512 anyway.

The second concern is perhaps more problematic. This will enshrine SHA512 use for the next 20? years (ignoring any potential impact of quatum computing and all new, to be developed, approaches to signing that work on really constrained systems). With the availability of the new sponge functions, is this the proper approach? This would appear to be the point to make a clean break in cryptographic approach.

Robert Moskowitz
HTT Consulting

Nikolaenko, Lera

From: Lera Nikolaenko <valerini@fb.com>
Sent: Tuesday, January 28, 2020 9:35 PM
To: fips186-comments <fips186-comments@nist.gov>
Cc: Kostas Chalkias <kostascrypto@fb.com>
Subject: Non-repudiation of Ed25519

Dear Author(s),

We would like to draw your attention to the fact that Ed25519 signatures (as currently specified in FIPS186-5 draft) do not provide strong non-repudiation guarantees: it is possible to construct tuples [pk, m, m', sig], such that both pairs [m, sig] and [m', sig] verify correctly under the public key pk, this happens for public keys that lie in a small order (8-torsion) subgroup.

In detail: if Q is an elliptic curve point such that 8 * Q = 0; then the verification equation 8 * S * G = 8 * R + 8 * SHA-512(R || Q || M) * Q will be equivalent to 8 * S * G = 8 * R and will not depend on the message! So any message will suit the verification equation under the public key Q and signature (R, S).

This problem can be easily avoided if the public keys lying in a small order group are rejected. This check is really inexpensive as the order of the small subgroup is 8, so there are only 8 valid points of low order.

For the FIPS 186-5 document we suggest to either explicitly state that the scheme does not provide strong non-repudiation guarantees or mandate a small subgroup membership check on the public key in the verification function.

--
Thank you
Valeria Nikolaenko and Kostas Chalkias

Olson, Ethan

From: Ethan Olson <EOlson@imagenet.com>
Sent: Wednesday, December 4, 2019 10:01 AM
To: fips186-comments <fips186-comments@nist.gov>
Subject: Comment on Draft FIPS 186-5

Clarification or text correction requested.

The document makes a definitive statement in section 7.1, which reads "For Ed25519, SHA-512 shall be used."

This language seems to indicate only one digest is to be considered acceptable, but there are two SHA-512 digests that have been defined. References are made in the beginning of the document to SHA-2 (FIPS 180) and SHA-3 (FIPS 202). Is there a preference between them in this case? Can the text be clarified to reflect such?

Ethan Olson
Solutions and Security
eolson@imagenet.com

Smith, David

From: Smith, David E. <David.Smith@cyber.gc.ca>
Sent: Thursday, January 30, 2020 4:52 PM
To: fips186-comments <fips186-comments@nist.gov>
Subject: Cyber Centre comments on FIPS 186-5 (Draft)

Please find below our general, editorial and technical comments and questions on the Draft FIPS PUB 186-5 issued for comment in October 2019.

David Smith
Canadian Centre for Cyber Security

Page, section, paragraph	Type	Comment
13, 3.3, 4	General	In the final paragraph of the page it says "However, if a verification or assurance process fails the digital signature should be considered invalid". According to the definition of "should" in Section 2.1 Terms and Definitions this is a strong recommendation but not a requirement of the standard. When would you not want to require these types of verifications. Perhaps it should be changed to "shall"?
24, 6.4.2, Step 4	Editorial	Appendix C.1 reference is incorrect. Should be Appendix B.1
26, 7.2	Editorial	Two different notations are used for multiplication when describing how to turn the octet string into an integer. Following the notation in Section 2.3 Mathematical Symbols both should be "*".
27, 7.3	Editorial	In the sentence following point "2." There is a bolded "**mod** 4". No other "mod" are bold in the document.
29, 7.6, Process	Editorial	Throughout the "process" section there is an inconsistent use of integers vs. octets in calculations. In RFC 8032 they are very specific that r and SHA512(R||Q||M) should be interpreted as little endian integers where here it is not specified. This leads to possible incompatible options in the operations in step 4.
30, 7.7, Step 4	Editorial	In RFC 8032 they use Q' in step 4, not Q
31, 7.8.1, Process step 5	Editorial	Same inconsistent use of integers vs. octets comment as for section 7.6 above. Need to specify the interpretation as integers before calculations.
32, 7.8.2, Process Step 4	Editorial	Should be Q' instead of Q.
36, A.1.2.2, Input	Editorial	Appendix B.3.2.1 reference is incorrect. Should be Appendix A.1.2.1.

37, A.1.2.2, Process Step 6.1 and 6.2	Editorial	Two references to Appendix C.10 are incorrect. Should both be Appendix B.10.
38, A.1.3, Process Step 4.7 and 5.8	General	Steps 4.7 and 5.8 have changed bounds from the previous version. 4.7 used to read i\geq 5(nlen/2); 5.8 used to read i\geq 5(neq/2). Is there an analysis justifying these changes? The algorithm doesn't exactly match the algorithm in section 5.2.2 of draft X9.80-2020.
40, A.1.4, Process steps 6.1 and 7.1	Editorial	Appendix C.10 reference in both of these steps is incorrect. Should be Appendix B.10.
45, A.2.1, Process step 3	Editorial	Table B.2 reference is incorrect. Should be Table A.2.
45, A.2.1, Process step 4	Editorial	Appendix B.6.1 reference is incorrect. I believe it should be Appendix A.4.1.
47, A.2.2, Process step 4	Editorial	Appendix B.6.2 is incorrect. Should be Appendix A.4.2.1.
47, A.2.2, Process step 4	Editorial	Step 5 of the process is accidentally included as the end of step 4. This should be a new line.
47, A.3.1, Outputs	General	The previous version (186-4) would also output k^{-1}.
48, A.3.2, Outputs	General	The previous version (186-4) would also output k^{-1}.
50, A.3.3, Process step 4.4	Technical	In RFC 6979, there is an added restriction that the generated k must not create an r=0. This condition is missing here.
54, B.1, Process step 8	General	There doesn't appear to be any cases of the use of this algorithm with non-prime modulus. Is step 8 necessary?
57, B.3, 1	Editorial	Two references to Appendix E are incorrect. Should be Appendix C.
67, B.9, Input	General	The condition on the size of r_1 and r_2 is slightly different from 186-4. This restriction does not appear in section 5.2 of X9.80.
68, B.9 Process step 7.1	Editorial	Appendix C.3 reference is incorrect. Should be Appendix B.3.
68, B.9, Process step 9	General	In 186-4, the bound on i was \geq 5(nlen/2).

69, B.10, Input point 2 and 4	Editorial	References to Table B.1 are incorrect. Should be Table A.1.
70, B.10 Process step 4	Editorial	Appendix C.6 reference is incorrect. Should be Appendix B.6.
71, B.10, Process step 14	Editorial	“algorithm of C.1” reference is incorrect. Should be “algorithm of B.1”.
78, E, point #7	Editorial	References to tables are incorrect. Should be A.1, B.1 and B.2.

Susella, Ruggero

From: Ruggero SUSELLA <ruggero.susella@st.com>
Sent: Wednesday, January 29, 2020 9:57 AM
To: fips186-comments <fips186-comments@nist.gov>
Cc: Gilles VAN ASSCHE <gilles.vanassche@st.com>; Thierry SIMON1 <thierry.simon1@st.com>; Sylvie WUIDART <sylvie.wuidart@st.com>
Subject: Comment on FIPS 186

Dear NIST,
Please find our comments on FIPS 186-5 (Draft):

- Regarding EdDSA and ECDSA deterministic signatures: we would prefer to allow an option to add random bits in the computation of the Per-Message Secret Number.
  We understand that in some systems the purely random generation of this secret number was poorly implemented and led to security failures. We like the idea of having a fail-safe design which does not rely on the random generation for its security. But we think that allowing to provide an additional optional random input to the deterministic Per-Message Secret Number generation algorithm is a low-cost way to increase resistance against side channel and fault attacks. To be effective, the random and the secret key should be the first input to the underlying hash function, as suggested in [1] and [2].
  For EdDSA, by having it as optional, it would be possible, for example, to extend the dom2 and dom4 functions to allow full compatibility with RFC 8032 when no random is used.

- For the deterministic version of ECDSA we would like to note that the procedure described in RFC 6979 is much more complex than what is done for EdDSA. In particular it requires the usage of several invocations of HMAC, which a pure signing/verifying device could not implement by default. Also, the secret number is part of HMAC's message input, while protections against side-channel attack usually protect HMAC's key input. We would favor a simpler approach more in line with EdDSA, we believe that KMAC is the correct choice for addressing what is, de facto, a keyed hash invocation. Furthermore, KMAC is much easier to protect against side-channel attacks than HMAC-SHA-2.

- We believe there is a strong need for a prehashed version of EdDSA. In embedded systems it might be infeasible to store the message in internal memory before computing the signature. An implementor might be tricked to believe that reading the message from an external memory twice is the proper way to address the issue. Unfortunately, if an attacker can tamper the external memory in between the two read operations she can recover the private key. Therefore, support of the prehashed version or constraining the input message to the internal memory size are mandatory in this use case.

- In section 7.7 the verification equation is written as [2^c * S]G = [2^c]R + (2^c * t)Q.
  Please note that the usage of different brackets is not clear in this context.
  More importantly this definition seems to disallow the use of the more commonly used cofactorless verification [S]G = R + [t]Q.
  Cofactorless verification is significantly simpler than cofactor verification, avoiding the need of decompressing R, to multiply by the cofactor and to convert [2^c]R back to affine coordinates.
  Therefore, we would prefer that FIPS 186-5 will more clearly allow, as RFC 8032 does, the usage of cofactorless verification.
  In case the intention was to remove ambiguity between the two verifications by defining only one, we would favor cofactorless verification over cofactor verification, as XEdDSA [3] did, because we do not see meaningful advantages to justify the additional cost.

- We agree with the proposal to remove DSA. The vast majority of the applications we see in our field prefer to use the elliptic curve variant and given its security concerns regarding the domain parameters generation we would prefer to avoid its usage.

- We believe that many applications will benefit from the high speed of ParallelHash128 or ParallelHash256 as approved hash functions for ECDSA and the prehashed variants of EdDSA, in particular when signing long messages.

- At beginning of Appendix B, Appendix C.1. is referenced instead of Appendix B.1.

- Appendix B.1. describes the EEA inversion algorithm as the reference algorithm for computing multiplicative inverses. We understand that this algorithm is very generic and supports composite moduli, but in the case of prime moduli, such as all the cases for which Appendix B.1. is referenced, we wonder whether it would be better to provide as reference the common technique based on Fermat's little theorem, which is better suited to avoid timing side-channel.

Kind Regards,
Ruggero Susella, Gilles Van Assche and Thierry Simon.
STMicroelectronics

[1] 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
[2] Samwel N, Batina L, Bertoni G, Daemen J, Susella R, (2017) Breaking Ed25519 in WolfSSL. *Cryptology ePrint Archive preprint*. https://ia.cr/2017/985
[3] Perrin T, (2016) The XEdDSA and VXEdDSA Signature Schemes. https://signal.org/docs/specifications/xeddsa/xeddsa.pdf

Willliamson, Michael

From: Michael Williamson <Michael.Williamson@wdc.com>
Sent: Monday, December 9, 2019 3:07 PM
To: fips186-comments <fips186-comments@nist.gov>
Subject: Comments on Draft FIPS 186-5

Western Digital has compiled three comments so far, all of which address elliptic curve portions of the specification.

1. Now that a deterministic method of per-message secret generation has been added (corresponding to RFC6979), the dangerous and error-prone randomized method of per-message secret generation is no longer needed. The randomized method should either be removed immediately OR if there are compelling applications for randomized signatures, the specification should be amended to provide additional safety. One mechanism would be to state that it must be used with an approved DRBG that accepts additional input AND both the private key and message digest should be provided as additional input on the first call to the DRBG in the per-message secret generation process. Another would be to specify only the general RFC6979 method, but allow applications to set the personalization string for the HMAC-DRBG instance used in this method, which would allow applications to randomize signatures whilst retaining the safety of the deterministic method. Additionally, section 6 should not treat deterministic ECDSA as a separate process with a separate name.
2. Appendix A should be improved:
    a. The method of key pair generation for EdDSA appears in 7.4, but should appear in Appendix A (or should at least be referenced there).
    b. The rejection sampling methods as specified are not correct. The text states that an iterative method is used, but the actual processes described fail to have any iteration in them.
    c. For both the rejection sampling methods and the deterministic ECDSA method, the process should be specified to continue until the probability of an invalid output is below 2^-N, where N is the bit length of the output, and if a valid parameter has not been generated, an ERROR indication may be returned at this point.
    d. Section A.2.2 has an incorrect reference to B.6.2 which should be A.4.2.
    e. I do not believe that A.3.3 needs to spell out how to write a HMAC-DRBG. The process should be described in the simplest and most unambiguous way, and if it is desired to allow implementations to use something less than a full HMAC-DRBG implementation, then that flexibility can be given.
3. Given the broad use and implementation of prime-order curves, and the general distrust of the process through which the NIST P-curves have been specified, it may be useful to specify an additional prime order curve for use with ECDSA. I would suggest secp256k1 for this purpose, given its broad use in new applications.

Thank you.
Michael Williamson, Program Manager, FIPS 140-2 Certifications
Western Digital®
Email: michael.williamson@wdc.com

Wooding, Mark

From: Mark Wooding <mark.wooding@trustonic.com>
Sent: Thursday, January 9, 2020 6:35 AM
To: fips186-comments
Subject: Comments on the FIPS186-5 draft

Please find my comments on the FIPS186-5 draft in the attached PDF.

Comments on the FIPS186-5 draft

Mark Wooding

4 December 2019

Contents

1 General

I strongly approve of almost all of the changes made in this revision.

2 Technical problems

2.1 §5.4: (RSA) PKCS #1

Isn't it time that we gave RSASSA-PKCS-v1.5 a decent burial?

2.2 §7.2: (EdDSA) Encoding

> The encoding of $\mathrm{GF}(p)$ is used to define "negative" elements of $\mathrm{GF}(p)$—specifically, x is negative if the $(b-1)$-bit encoding of x is lexicographically larger than the $(b-1)$-bit encoding of $-x$.

This looks like it's a paraphrase from the Bernstein, Duif, Lange, Schwabe, and Yung paper:

> We use the encoding of $\mathbf{F}_q$ to define some field elements as being negative: specifically, x is negative if the $(b-1)$-bit encoding of x is lexicographically larger than the $(b-1)$-bit encoding of $-x$. If q is an odd prime and the encoding is the little-endian representation of $\{0, 1, \ldots, q-1\}$ then the negative elements of $\mathbf{F}_q$ are $\{1, 3, 5, \ldots, q-2\}$.

However, the version in the FIPS186-5 draft has a number of problems:

- The quantity b is not defined.
- The previous paragraphs define a little-endian encoding in terms of octets. This can't be the $(b-1)$-bit encoding mentioned in the definition of 'negative' elements above unless $b \equiv 1 \pmod 8$, which would be strange.
- Ignoring that difficulty, if we compare little-endian *octet* encodings of field elements lexicographically, we end up with a very strange ordering which is certainly not that envisaged by Bernstein *et al*.
- Fortunately, nothing depends on this definition of 'negative' field elements. Bernstein *et al* use the concept in their point compression scheme, but the FIPS186-5 draft has already defined compressed points without recourse to any notion of 'negative' field elements.

I suggest that this paragraph be deleted.

2.3 §7.3: (EdDSA) Point Decoding

The suggested computation for square-roots-of-ratios in $\mathrm{GF}(2^{255}-19)$, taken as it is directly from the original Bernstein *et al* paper, is unnecessarily complicated. Instead, to calculate a square root of x/y, first let $v = xy$ and $w = xv^{3(p-5)/8+1}$. Notice that $v = y^2(x/y)$, so if x/y is square then so is v; and in this case we have $v^{(p-1)/2} = 1$. Then, assuming x/y is indeed a square, we have

$$\begin{aligned}
w^4 &= x^4 v^{3(p-5)/2+4} \\
&= x^4 v^{(3p-7)/2} \\
&= x^4 v^{3(p-1)/2-2} \\
&= x^4 v^{-2} (v^{(p-1)/2})^3 \\
&= x^4/v^2 = x^4/(x^2y^2) = x^2/y^2
\end{aligned}$$

Now, if $w^2 y = x$ then w is the square root we seek; otherwise choose some square root i of -1 and use iw.

Similarly, square-root-of-ratios in $\mathrm{GF}(2^{448} - 2^{224} - 1)$ can be calculated as follows. Again, suppose we want the square root of x/y. Let $v = xy$ and $w = xv^{(p-3)/4}$; then, assuming x/y is indeed a square, we have

$$w^2 = x^2 v^{(p-3)/2} = x^2 v^{(p-1)/2-1} = x^2/v = x/y$$

as required.

I don't know where the complicated versions came from originally. I explain how I derived my simpler versions in https://vox.distorted.org.uk/mdw/2017/05/simpler-quosqrt.html.

2.4 §7.6: EdDSA Signature Generation

This procedure doesn't permit the Ed25519ctx construction from RFC8032. The context parameter can be provided to the Ed25519ph scheme defined in §7.8, and to Ed448, but not to the non-prehashed variant of Ed25519. Prehashing undermines the collision-resilience property brought about by prefixing the message with part of the siganture prior to hashing. On the other hand, the context string improves robustness against cross-protocol attacks. As specified, FIPS186-5 makes us choose between context strings and collision resilience. No rationale is given for why we have to make this uncomfortable and unnecessary tradeoff.

2.5 §A.3.3: Per-Message Secret Number Generation for Deterministic ECDSA

The algorithm given doesn't match RFC6979.

Notably, RFC6979 (`bits2octets`, §2.3.4) takes the the incoming hash H, truncates it to match the bitlength of the group order q (part of `bits2int`, §2.3.2), converts the truncated hash to an integer (`bits2int`, §2.3.2), reduces this integer modulo q (part of `int2octets`, §2.3.3), and finally converts the reduced integer back to an octet string. The truncation and reduction are omitted from the FIPS186-5 draft: none of the conversion subroutines in §B.2 accept a q as input or perform reduction; and §A.3.3 (step 1.2) invokes conversion from bitstring to octet string without mentioning truncation or reduction either.

2.6 §B.1: Computation of the Inverse Value

This section explains how to calculate a modular inverse using an extended version of Euclid's algorithm. Inversion is part of DSA and ECDSA signature computation, applied to the per-message secret. Alas, Euclid's algorithm is very difficult to implement in a timing-invariant manner in software: it takes a variable number of iterations, and requires a number of Euclidean division steps – which is itself rather tricky to implement efficiently in constant time.

DSA and ECDSA only require inversion modulo a prime q. It would be useful to point out that this can be performed using Fermat inversion: $k^{-1} \equiv k^{q-2} \pmod{q}$. Fermat inversion requires constant-time modular exponentiation, but this is relatively straightforward, and similar to the scalar multiplication which is already a necessary part of ECDSA.

Alternatively, it may be valuable to cite Bernstein and Yang, 'Fast constant-time GCD computation and modular inversion', https://eprint.iacr.org/2019/266.

(I know that Fermat or Bernstein–Yang inversion are acceptable according to the 'or an algorithm which produces an equivalent result' clause.)

2.7 §B.3: Probabilistic Primality Tests

> A probabilistic primality test may be required during the generation and validation of prime numbers. An approved robust probabilistic primality test shall be selected and used.

Despite this introductory paragraph, the rest of this section appears to be concerned only with components of RSA keys. Indeed, the use of the Miller–Rabin probabilistic test with a small number of iterations to validate the primality of adversarially provided integers is unsound, so these techniques are not suitable for validating domain parameters: see Albrecht, Massimo, Paterson, and Somorovsky, 'Prime and Prejudice: Primality Testing Under Adversarial Conditions': `https://eprint.iacr.org/2018/749`. On the other hand, it seems that using multiple MR tests followed by a Lucas test is overkill: it's an open problem to construct a Baillie–PSW (MR test with base 2 and Lucas test) pseudoprime, and it may be that none exist with fewer than 10000 digits.

3 Editorial problems

3.1 §3.2: (General) Digital Signature Generation

This section describes a general signing procedure which involves the following steps:

1. hash the message;
2. generate a per-message secret;
3. compute the signature;
4. optionally, verify the signature.

This doesn't work for non-prehashed variants of EdDSA, which want to hash the message themselves, twice, with different prefixes.

3.2 §3.3: (General) Digital Signature Verification and Validation

Similar to §3.2, this describes a verification procedure which involves hashing the message before looking at the signature.

3.3 §5.4: (RSA) PKCS #1

Footnote 1 refers back to an item '(f)'. I think it should be '(h)'.

3.4 §A.1.1: Criteria for IFC Key Pairs

A lot of work in this section is done by two words 'provable' and 'probable'. Alas, they are almost identical, differing in only one letter in the middle of the word. This makes the text unnecessarily hard to read.

3.5 §A.1.2.2: Construction of the Provable Primes p and q

Steps 6.1 and 7.1 refer to an Appendix C.10 which does not exist. I think this should refer to B.10 instead.

3.6 §A.4.2: Conversion of a Bit String to an Integer mod n via the Discard Method

Missing space in 'n =N'.

3.7 §B.10: Construct a Provable Prime . . .

Step 6 refers to Appendix C.6, which does not exist. Step 5.1 correctly refers to Appendix B.6.

3.8 §E: Revisions

Typo 'Aonstructing . . .'; also '. . . are allowed' should be '. . . is allowed': the subject is 'constructing'.

Received: December 30, 2019
Status: Posted
Posted: January 29, 2020
Tracking No. 1k3-9e5r-hvdh
Comments Due: January 29, 2020
Submission Type: API

Yousar, Annie

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-2019-0004-0005
Comment on FR Doc # 2019-23742

Submitter Information

Name: Annie Yousar
a.yousar@informatik.hu-berlin.de

General Comment

See attached file(s)

Attachments

Change proposal for Appendix A.4

Annie Yousar (a.yousar@informatik.hu-berlin.de)

This proposal concerns the generation of integers in a given interval. A third method is added which is as appropriate as the modular reduction. Note that in some sense it is even better than the modular reduction. I.e., if all $2^l = N$ integers are distributed in n boxes, then the small remainder boxes are filled via modular reduction with one more integer (exactly N div n +1), whereas all the others are filled with N div n integers only. If one reduces the N integers via the described here multiplication method, and put them in the corresponding boxes, then the boxes with one integer more are "uniformly" distributed over the n remainder boxes.

The following text has revision marks (few editorial hints added).

A.4 Random Values mod *n*

Key pair generation and per-message secret number generation for ECC require random values generated in the interval $[1,n-1]$. This appendix provides three methods which all use the output of an **approved** (i.e., cryptographically strong) RBG and convert this to an integer in this interval; the respective methods differ in how they reduce bias.

Gelöscht: two
Gelöscht: both

The method of Appendix A.4.1 reduces the output of this RBG modulo $n–1$ while ensuring that any bias introduced during this conversion process is negligible in practice, whereas the method of Appendix A.4.2 uses an equivalent procedure via multiplication. The method of Appendix A.4.3 simply checks whether the random output is in the requested interval.

Gelöscht: *N*

A.4.1 Conversion of a Bit String to an Integer mod *n* via Modular Reduction

This method uses the output of an approved RBG in the interval $[0, N-1]$ and converts this to an integer in the interval $[1, n-1]$ by simply reducing this output modulo n –1 while ensuring that any biases introduced during this conversion process are negligible in practice. If n –1 does not divide N, this invariably introduces some bias, no matter the quality of the input distribution, which is easy to determine from N and n. In particular, if the bit-length of N is sufficiently larger than that of n, the bias introduced by the modular reduction operation is negligible in practice. The same is true if N is close to a multiple of n (e.g., if n is close to a power of two, and the input distribution is generated by a strong RBG with a fixed bit-length output). The method ensures that outputs are in the interval $[1, n-1]$.

Gelöscht: 0

Inputs:

1. Positive integer n, greater than 1
2. Threshold value $0 \le \varepsilon \le 2^{-64}$ (the upper bound on bias)
3. Bit-string X, the string to be converted

Output: Integer x in the interval $[1, n-1]$, or INVALID.

Process:

1. Let l be the bit length of X. Set $N = 2^l$, $r = N \bmod (n-1)$, and $\rho = r/(n-1)$. If $N < n$ then return INVALID;
2. If $2\rho(1-\rho)(n-1) > \varepsilon \cdot N$ or if $\varepsilon > 2^{-64}$, return INVALID;
3. Convert the bit-string X to the integer x using the procedure of Appendix B.2.1;
4. Set $x = x \bmod (n-1)$;
5. Set $x = x + 1$;
6. Output x.

Formatiert: Schriftart: Kursiv
Gelöscht: ϵ

A.4.2 Conversion of a Bit String to an Integer mod *n* via Multiplication

This method uses the output of an approved RBG in the interval $[0, N-1]$ and converts this to an integer in the interval $[0, n-1]$ by simply multiplying this output with n and reducing the result by truncation to the left-most bits. This procedure ensures as well that any biases introduced during this conversion process are negligible in practice. If n does not divide N, this invariably introduces some bias, no matter the quality of the input distribution, which is easy to determine from N and n. In particular, if the bit-length of N is sufficiently larger than that of n, the bias introduced by the multiplication and truncation is in practice as negligible as the method in the Appendix A.4.1. The method ensures that outputs are in the interval $[1, n-1]$.

Inputs:

1. Positive integer n, greater than 1
2. Threshold value $0 \leq \varepsilon \leq 2^{-64}$ (the upper bound on bias)
3. Bit-string X, the string to be converted

Output: Integer x in the interval $[0, n-1]$, or INVALID.

Process:

1. Let l be the bit length of X. Set $N = 2^l$, $r = N \bmod n$, and $\rho = r/n$. If $N < n$ then return INVALID;
2. If $2\rho(1-\rho)n > \varepsilon \cdot N$ or if $\varepsilon > 2^{-64}$, return INVALID;
3. Convert the bit-string X to the integer x using the procedure of Appendix B.2.1;
4. Set $x = x \cdot n$;
5. Set $x = x$ div N;
6. If $x = 0$, output INVALID;
7. Output x.

A.4.3 Conversion of a Bit String to an Integer mod *n* via the Discard Method

This method for converting a probability distribution on $[0, N-1]$ into a probability distribution on the interval $[0, n-1]$ accepts an output in the interval $[0, N-1]$ only if this is also in the interval $[0, n-1]$ and returns INVALID otherwise. If n is an integer in the interval $[N/2, N]$, this results in a distribution that is always close to that of the original distribution. In contrast to the method in Appendix A.4.1, this method is nondeterministic unless $n = N$. Note that if n is close to a power of two, and the input distribution is generated by a strong RBG with a fixed bit-length output, the probability of returning INVALID is low.

Inputs:

1. Bit-string X, of length l,
2. Positive integer n, where $2 \leq n < 2^l$.

Output: Integer x in the interval $[1, n-1]$, or INVALID.

Process:

1. Set $N = 2^l$, where l is the bit-length of X;
2. If $n \leq 1$, or $n \geq N$, then output INVALID;
3. Convert the bit-string X to the integer x using the procedure of Appendix B.2.1;
4. If x is not an integer in the interval $[0, n-2]$, output INVALID;
5. Set $n = n+1$;
6. Output x.

Formatiert: Schriftart: Kursiv

Formatiert: Schriftart: Kursiv

Formatiert: Schriftart: Kursiv

Zaverucha, Greg

From: Greg Zaverucha <gregz@microsoft.com>
Sent: Friday, January 24, 2020 5:18 PM
To: fips186-comments <fips186-comments@nist.gov>
Subject: Comment on Draft FIPS 186-5

I’ve attached my comments.

Greg

Comments on FIPS 186-5
Greg Zaverucha
January 24, 2020

Here are two comments on the draft of FIPS 186-5 [1]. Note that changing FIPS 186-5 based on the first comment would not break interoperability with existing implementations of EdDSA and Deterministic ECDSA. The change discussed in the second comment could cause keys created in one implementation to not work properly in another, but signatures would verify across implementations.

1. When deriving the per-message secret number using a deterministic construction that depends on (part of) the private key and message to be signed, the standard should allow implementations to add additional randomness to the derivation process. The deterministic construction addresses security risks due to randomness failures (e.g., biased output from an RBG), but makes certain fault and side-channel attacks much easier to mount. Adding a random nonce to derivation prevents certain side-channel attacks, while retaining security against randomness failures. In [2] it is shown for Schnorr-type signatures (including EdDSA) many fault attacks are prevented when a different nonce is used for each signature, and recommends randomization for this reason. The IETF draft [3] also recommends this randomization, and both [2] and [3] cite multiple existing uses of it.

Proposed change:

A.3.3 Per-Message Secret Number Generation for Deterministic ECDSA

- Add an optional additional_input field.

7.6 EdDSA Signature Generation

- step 2.1, 2.2: add optional additional_input field to derivation of r

In both places recommend that the additional_input be a 256-bit random string (output by approved an RBG).

2. The EdDSA scheme as specified has the property that revealing a hash of the private key (denoted d on page 29) breaks the scheme, since d is a seed value used to re-derive the effective private key.

Of course, it's well understood by cryptographers that revealing a hash of a private key voids any security guarantees provided by a primitive, and the document rightly forbids this in Section "5.2 RSA Key Pair Management".

However, it appears to be secure to reveal a hash of the private key of signature algorithms currently in use (ECDSA, RSA-PKCS#1-v1.5 and RSA-PSS) since they do not use a hash of the private key during signature generation. So in this sense EdDSA (as specified) is not a drop-in replacement; an existing system that uses a hash of the private key (e.g., as a key identifier or for integrity) may be secure with the ECDSA and RSA signature algorithms, but would become insecure with the addition of EdDSA.

I don't have strong evidence or examples that suggest the risk of exposing $H(d)$ is high. But since it would make the standard more resistant to misuse without a significant performance impact, it seems prudent to address it. Two options come to mind. First, one could choose $(s, \mathit{hdigest2})$ as independent random values. This is simple but increases the size of the private key. Another option is to add a constant to the input of H, effectively computing H', to reduce the chance that an existing system computes $H'(d)$ in another context.

References

[1] FIPS PUB 186-5 (Draft). Digital Signature Standard (DSS). FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5-draft.pdf`

[2] Diego F. Aranha and Claudio Orlandi and Akira Takahashi and Greg Zaverucha. Security of Hedged Fiat-Shamir Signatures under Fault Attacks. IACR ePrint report 2019/956. August 2019. `https://eprint.iacr.org/2019/956`. To appear at EUROCRYPT 2020.

[3] John Preuss Mattsson, Erik Thormarker and Sini Ruohomaa. Deterministic ECDSA and EdDSA Signatures with Noise. December 2019. `https://tools.ietf.org/html/draft-mattsson-cfrg-det-sigs-with-noise-00`