Paragon Initiative Enterprises Blog

The latest information from the team that develops cryptographically secure PHP software.

Security Concerns Surrounding WebAuthn: Don't Implement ECDAA (Yet)

August 23, 2018 9:34 am by **P.I.E. Staff (/blog/author/p-i-e-staff)**

Security Engineering (/blog/category/security-engineering)

Earlier this year, the World Wide Web Consortium (W3C) and FIDO Alliance shared their latest drafts for a standard Web Authentication API called **WebAuthn**.

For context (present and historical): The current version (as of this writing) of the WebAuthn specification lives **here (https://www.w3.org/TR/2018/CR-webauthn-20180807/)**, and the most up-to-date version can be found **here (https://www.w3.org/TR/webauthn/)**.

Our security team took an interest to this proposal since WebAuthn would be used in conjunction hardware two-factor authentication devices. Hardware 2FA has proven to be far more resilient against phishing attacks than HOTP or TOTP (meanwhile, SMS-based 2FA is essentially security theater; *avoid like the plague*).

Despite the importance of WebAuthn to web security for the years to come, our analysis of the standard reveals a *lot* of concerns that almost any cryptographer should have been able to identify and remedy earlier in the design phase.

Regardless of whether this was a failure of the W3C and/or FIDO Alliance to enlist the aid of cryptography engineers, or of the cryptography community to be more proactive in preventing the deployment of error-prone cryptographic designs, there is only one path forward; and that is to fix the design of WebAuthn *before* it's set in stone.

WebAuthn Security Risk Overview

Update (2018-09-18)

Since this blog post was referenced in a **ZDNet article about WebAuthn and ECDAA (https://www.zdnet.com/article/worries-arise-about-security-of-new-webauthn-protocol)**, it's made the rounds on social media and some people have been exaggerating its contents. We suspect this was because of a lack of clarity on our part, and wish to preemptively remedy this going forward:

What follows isn't a list of critical, game over vulnerabilities in WebAuthn. Rather, this is a list of design grievances that, in similar protocols, have led to security disasters over the years.

If you were already doing so, you should still use WebAuthn.

Our purpose in publishing this document was to have a publicly accessible resource that can be easily referenced in our ongoing discussions with the FIDO Alliance and their member organizations without having to dig through email threads that might not have been consistently forwarded or carbon copied.

A lot of our criticism is easily remedied by updating the relevant documentation to clarify and/or provide a justification for specific design decisions.

For example, until August of this year, **point compression was covered by a non-expired patent (https://patents.google.com/patent/US6130946A/en)**, so the FIDO Alliance didn't consider it an option in earlier designs. However, they failed to make it explicit in their specification that point validation should be mandatory. (Instead, this is spelled out in **their compliance/policy documents (https://fidoalliance.org/certification/authenticator-certification-levels/)**. The FIDO Alliance's security secretariat agreed that this should be made clearer in the specifications themselves.) While we stand by point compression being a significantly more robust design than expecting implementors to validate points (and not telling them about this requirement until it's time to get certified), the FIDO Alliance and W3C can remedy this by updating the specification documentation to make it mandatory. No change is absolutely necessary, although it would certainly be an improvement.

The call to action was intended for implementors, not users:

1. **Help your users avoid RSA entirely (https://latacora.singles/2018/04/03/cryptographic-right-answers.html)**, if possible. EdDSA is your best bet, followed by **deterministic ECDSA (https://tools.ietf.org/html/rfc6979)**.
2. Don't implement ECDAA in its current incarnation, because it's likely to be changed in some ways, and the less we get trapped in the quagmire of backward compatibility, the better off the Internet of tomorrow will be.

WebAuthn and ECDAA are not doomed. Don't throw away your hardware tokens, revert your codebases to use SMS or TOTP, or any other such drastic measures.

> The remainder of this article continues below, unedited.

WebAuthn employs a standard called **COSE (RFC 8152) (https://tools.ietf.org/html/rfc8152)**, which builds on the **error-prone JOSE standards (https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid)**.

In the **COSE Algorithm Registrations (https://www.w3.org/TR/2018/CR-webauthn-20180807/#sctn-cose-alg-reg)** section of the WebAuthn specification, it notes that RSASSA-PKCS1-v1_5 is already registered by COSE and then registers two additional COSE algorithm identifiers for use in WebAuthn, based on the FIDO Alliance's **ECDAA** algorithm.

Bleichenbacher's Monster Returns

As a consequence of its COSE legacy, WebAuthn specifically requires ongoing support for RSA with PKCS1v1.5 padding. Much has been written about the past ***twenty years*** of padding oracle and signature forgery vulnerabilities inherent to PKCS1v1.5 padding.

If you're not familiar with RSA with PKCS1v1.5 padding, just know that a team of researchers won **a prestigious information security award (https://pwnies.com/winners/#crypto)** at the Black Hat conference this year for discovering that systems are still vulnerable to decades-old vulnerabilities **simply because they *still* support RSA PKCS1v1.5 padding**.

As we covered in a previous blog post, it *is* possible to **implement PKCS1v1.5 securely (https://paragonie.com/blog/2018/04/protecting-rsa-based-protocols-against-adaptive-chosen-ciphertext-attacks#rsa-anti-bb98)**, but this requires an application-layer mitigation; your library can't do it for you.

If you didn't explicitly write your RSA-based protocol to side-step these PKCS1v1.5 vulnerabilities (and, instead, you just used whatever API your version of OpenSSL and/or programming language gave you), you're probably vulnerable! (Unless you don't use this padding mode, of course.)

So while **support for RSA with PKCS1v1.5 padding is explicitly required (https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-server-v2.0-rd-20180702.html#other)** in the FIDO2 server requirements, we implore nobody to ever actually *allow* this padding mode to be used.

> **In short:** PKCS1v1.5 is bad. The exploits are almost old enough to legally drink alcohol in the United States. Don't use it!

Fortunately, WebAuthn only uses RSA for signatures, so the relevant exploits are **much easier to work around (https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/#practicesafecrypto)**. You're still much safer not using RSA at all, or if you must, only using RSASSA-PSS.

ECDAA: Exceedingly Concerning Decisions About Authentication

To fully appreciate the security concerns with **the FIDO Alliance's ECDAA specification (https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html)**, it's worth skimming over the past two decades of research into elliptic curve cryptography.

A Brief History of Real World Elliptic Curve Cryptography

In the year **2000**, Biehl, et al. publishes a paper on **Differential Fault Attacks on Elliptic Curve Cryptosystems (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.3920&rep=rep1&type=pdf)** (PDF), presented at CRYPTO 2000. This paper laid the groundwork for a class of active attacks against elliptic curve cryptosystems called **invalid curve attacks**. As recently as 2017, **invalid curve attacks have threatened ECDH-ES in the JOSE standards (https://blogs.adobe.com/security/2017/03/critical-vulnerability-uncovered-in-json-encryption.html)**.

In **2010** at the Chaos Communication Congress, fail0verflow exploits a k-value reuse to **steal Sony's ECDSA secret key (https://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html)**.

Cryptographers argued for a while about who to blame for this ECDSA failure, then Thomas Pornin published **RFC 6797: Deterministic (EC)DSA (https://tools.ietf.org/html/rfc6979)** in **2013** to prevent k-value reuse in ECDSA implementations (without breaking backwards compatibility). However, cryptographers arguing about the ECDSA failure wasn't a fruitless effort.

In **2013**, Daniel J. Bernstein and Tanja Lange published **SafeCurves (https://safecurves.cr.yp.to)**, a website that evaluated a lot of popular elliptic curve designs on very rigorous criteria: It isn't sufficient for an elliptic curve cryptography algorithm to be secure against ECDLP attacks, there are a lot more requirements for making these algorithms secure in real world ECC. SafeCurves takes these requirements into account.

In **2016**, the **Crypto Forum Research Group (https://irtf.org/cfrg)** approves the publication of **RFC 7748 (https://tools.ietf.org/html/rfc7748)** and **RFC 8032 (https://tools.ietf.org/html/rfc8032)**.

These RFCs were the result of *years* of bikeshedding over the subtleties of elliptic curve designs and parameter choices: Weierstrass vs Montgomery vs Edwards curves, cofactors, twists, the tradeoffs of 1 mod 4 versus 3 mod 4, etc.

CFRG discussions are very intense, deeply technical, and at many points heated.

At some point the mounting tension in the CFRG was briefly broken by someone vigorously demanding feedback for **their homemade cipher "Crystalline" (https://www.ietf.org/mail-archive/web/cfrg/current/msg06791.html)**, to which renowned security expert and nocoiner Tony Arcieri eventually **obliged (https://www.ietf.org/mail-archive/web/cfrg/current/msg06805.html)**.

Takeaway

1. Invalid curve attacks leak your secret key
2. Nonce reuse in ECDSA leaks your secret key
3. Elliptic Curve Cryptography parameter choice is a very complicated issue best left to experts (who will still take years to arrive at a satisfactory answer)
4. Don't roll your own crypto

ECDAA Considered Harmful

Out of all the hard-won ECC security lessons one could glean from the past two decades of real world security failures, ECDAA seems to have learned precisely **none** of them.

ECDAA Specifies Uncompressed Points

There are two schools of thought for preventing Invalid Curve Attacks:

1. Tell implementors to verify that any points they receive are on the curve, and hope they remember to do so consistently.
2. Use point compression and make it *not their problem*.

Point compression means sending only the X or Y coordinate, and the sign (positive or negative) of the other coordinate.

Uncompressed	Compressed
x= ..... y=	x= ..... y= pos/neg
Binary: 0x[04][x][y]	Binary: 0x[02|03][x]
(x,y) not guaranteed to be on the curve	If x is valid, y is on the curve
Dangerous	Safe

Point compression is widely regarded among cryptographers and cryptography engineers as the preferred mitigation strategy for preventing invalid curve attacks.

ECDAA explicitly specifies using uncompressed points (**ECDAA specification, section 3.1.2 (https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html#encoding-ecpoint-values-as-byte-strings-ecpointtob)** and **WebAuthn, section 8.6 (https://www.w3.org/TR/2018/CR-webauthn-20180807/#fido-u2f-attestation)**), which allows an attacker to choose an (x,y) pair that isn't on the curve.

Point compression, by contrast, is not allowed per the current ECDAA specification; nor is it mentioned anywhere.

ECDAA Specifies Non-Deterministic Signatures

Section 3.5 of the ECDAA specification (https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html#ecdaa-sign) contains several invocations of randomness (i.e. $RAND(p)$).

As we saw in the **history of real world ECC security** section of this post, randomly generated k-values have historically led to security disaster (i.e. revealing the secret keys for the ECDSA signatures that helped secure the PlayStation 3). The correct way to use ECDSA is with deterministic signatures.

Further, it is worth noting that the ECDAA specification's definition of $RAND(x)$ **does not include any cryptographic security requirements (https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html#notation)**. It is incredibly likely that developers would implement this using a Linear Congruent Generator or Mersenne Twister, rather than **the kernel's CSPRNG (https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-languages)**.

Therefore, it is very likely that continuing to rely on randomness in ECDAA signing will lead to more ECC security disasters.

ECDAA in WebAuthn is Specified over Barreto-Naehrig (BN) Curves

WebAuthn **specifies (https://www.w3.org/TR/2018/CR-webauthn-20180807/#sctn-cose-alg-reg)** two ECDSA algorithms: ED256 (**TPM_ECC_BN_P256 (https://trustedcomputinggroup.org/wp-content/uploads/TCGAlgorithmRegistry_Rev01.15.pdf)** with SHA256) and ED512 (ECC_BN_ISOP512 with SHA512). These are both Barreto-Naehrig curves used for pairing-based cryptography, and suffer from a **pretty serious security reduction (https://moderncrypto.org/mail-archive/curves/2016/000740.html)**.

Generally, if you have an elliptic curve with a prime of magnitude 2^n, you have roughly $\frac{n}{2}$ bits of security against the Elliptic Curve Discrete Logarithm Problem (ECDLP). Therefore:

- 128-bit curves should offer 64 bits of ECDLP security.
- 256-bit curves should offer 128 bits of ECDLP security.
- 512-bit curves should offer 256 bits of ECDLP security.

However, due to advancements in cryptanalysis, 256-bit BN curves **"no longer offer 128 bits of security, but perhaps closer to 96 or so" (https://ellipticnews.wordpress.com/2016/09/02/crypto-and-ches-2016-santa-barbara-ca-usa/)**. This is a speedup factor of 32 bits (roughly 4 billion).

The ill consequences of this curve choice are exacerbated by several **ECC (https://safecurves.cr.yp.to/transfer.html) security (https://safecurves.cr.yp.to/disc.html) deficits (https://safecurves.cr.yp.to/ladder.html) inherent (https://safecurves.cr.yp.to/twist.html)** to **BN (https://safecurves.cr.yp.to/complete.html) curves (https://safecurves.cr.yp.to/ind.html)**.

		Parameters:			ECDLP security:				ECC security:			
Curve	Safe?	field	equation	base	rho	transfer	disc	rigid	ladder	twist	complete	ind
Anomalous	False	True✓	True✓	True✓	True✓	False	False	True✓	False	False	False	False
M-221	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
E-222	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
NIST P-224	False	True✓	True✓	True✓	True✓	True✓	True✓	False	False	False	False	False
Curve1174	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
Curve25519	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
BN(2,254)	False	True✓	True✓	True✓	True✓	False	False	True✓	False	False	False	False
brainpoolP256t1	False	True✓	True✓	True✓	True✓	True✓	True✓	True✓	False	False	False	False
ANSSI FRP256v1	False	True✓	True✓	True✓	True✓	True✓	True✓	False	False	False	False	False
NIST P-256	False	True✓	True✓	True✓	True✓	True✓	True✓	False	False	True✓	False	False
secp256k1	False	True✓	True✓	True✓	True✓	True✓	False	True✓	False	True✓	False	False
E-382	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
M-383	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
Curve383187	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
brainpoolP384t1	False	True✓	True✓	True✓	True✓	True✓	True✓	True✓	False	True✓	False	False
NIST P-384	False	True✓	True✓	True✓	True✓	True✓	True✓	False	False	True✓	False	False
Curve41417	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
Ed448-Goldilocks	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
M-511	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓
E-521	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓	True✓

The FIDO Alliance Rolled Their Own Crypto

To round off the list of takeaways that ECDAA failed to take into consideration, the FIDO Alliance designed their own cryptography standard for anonymous attestations (using pairing-based cryptography over elliptic curves).

That puts us at 0 for 4 on **learning from history so we're not doomed to repeat it.**

In an interesting twist of fate, according to someone familiar with ECDAA and the FIDO Alliance, **they hadn't yet implemented ECDAA themselves (https://twitter.com/herrjemand/status/1031511164671483906)** (**mirrored (https://archive.fo/6CflR)**).

While this is *really weird* to hear (why would anyone attempt to standardize a cryptography protocol they hadn't implemented, let alone tested, yet?), it does present to us an opportunity to fix the standard before it's burdened by backward compatibility requirements and we end up with an echo of **POODLE (https://www.openssl.org/~bodo/ssl-poodle.pdf)**.

Recommendations

> Developers: Please, do NOT implement ECDAA in your WebAuthn libraries.

At least, not yet.

It might sound like a tremendous amount of work to shore up the security of ECDAA but two of the three proposed fixes are very simple.

Fix #1: Require (Or, At Least Allow) Point Compression

This recommendation should be obvious at this point. **Read about point compression above**, if you haven't already done so.

With point compression, you'll be eliminating an entire class of active attacks and preventing a long tail of implementation error.

Fix #2: Use Deterministic Nonces

There are two (good, but different) approaches to ensuring deterministic signatures:

1. Implement an HMAC-based approach, similar to **RFC 6797 for DSA and ECDSA (https://tools.ietf.org/html/rfc6979)**
2. Use a hash of the message (with optional domain separation) instead of a random integer.

For example, there are two points in **EcdaaSign (https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html#ecdaa-sign)** that random values are being generated.

These could easily be replaced by $H(c1||msg)$ and $H(c2||msg)$, where $c1$ is the byte `0xF1` repeated a number of times equal to the block size of the hash function (32 for SHA256, 64 for SHA512), and $c2$ is the byte `0xD0` repeated the same number of times.

(The constants `0xF1` and `0xD0` were chosen as a visual nod to the FIDO Alliance when represented in hexadecimal.)

The motivation to use a domain-separated hash function instead of HKDF or HMAC is that ECDAA is going to be implemented in hardware, and since a hash function is already being used, the overhead for this change is minimal.

Fix #3: Reconsider BN Curves

In light of the BN curve security reduction mentioned previously, an alternative curve for pairing-based cryptography should be considered.

When we shared our initial criticism of ECDAA with the FIDO Alliance, we CC'd Tony Arcieri (mentioned in the history section above) and he suggested looking at **BLS12-381 (https://eprint.iacr.org/2017/1050)** ("JubJub") as an alternative to BN_P256.

Fix #4: Hire Cryptographers to Review Your Designs and Implementations

Cryptography code is hard. **Mistakes happen (http://www.cryptofails.com/post/70059600123/saltstack-rsa-e-d-1)**. Misunderstandings are everywhere.

While not everyone who works with cryptography is on board, there is a large community of computer security experts and cryptographers that prioritize solutions over blame, and that wants to see projects like W3C's WebAuthn succeed.

If anyone wants to design a novel cryptography standard, reach out to cryptography experts. The CFRG is probably not the worst place to find one.

If yours is a commercial product, contract or hire at least one cryptographer to review your design and suggest changes, then implement them.

That being said, it **cannot** be the case that an alliance consisting of large companies and hardware security token vendors that advocates for improved web security fails to learn from the past two decades of real world cryptographic security research. Nobody wins in this scenario, except the attackers.

Permalink (/b/ya9unbDYhvmp2EUy)

Discuss (https://news.ycombinator.com/item?id=17852347)

View source (Markdown) (/blog/2018/08/security-concerns-surrounding-webauthn-don-t-implement-ecdaa-yet/source)

Authentication (/blog/tag/authentication) **Authorization (/blog/tag/authorization)**
Cryptography (/blog/tag/cryptography) **Login (/blog/tag/login)**
Public Key Cryptography (/blog/tag/public-key-cryptography) **Signatures (/blog/tag/signatures)**
Vulnerability (/blog/tag/vulnerability)

About the Author

P.I.E. Staff (/blog/author/p-i-e-staff)

Paragon Initiative Enterprises

Paragon Initiative Enterprises is a Florida-based company that provides software consulting, application development, code auditing, and security engineering services. We specialize in PHP Security and applied cryptography.

About

Paragon Initiative Enterprises
offers **technology consulting (/service/technology-consulting)** and **web development (/service/web-development)** services to businesses with attention to **security above and beyond compliance**

(/service/appsec).

✔ Our Professional Experience (/experience)

Archives

Blog Categories

- Business (/blog/category/business)
- Paragon Initiative (/blog/category/paragon-initiative)
  - Community (/blog/category/community)
    - Open Source (/blog/category/open-source)
    - Pharaoh (/blog/category/pharaoh)
    - Security Advice (/blog/category/security-advice)
  - Our Products (/blog/category/our-products)
    - Airship (/blog/category/airship)
    - ASGard (/blog/category/asgard)
    - Ward (/blog/category/ward)
  - Slice of PIE (/blog/category/slice-pie)
- Security News (/blog/category/security-news)
- Technology (/blog/category/technology)
  - Cryptology (/blog/category/cryptology)
  - Databases (/blog/category/databases)
  - Hardware (/blog/category/hardware)
  - Programming (/blog/category/programming)
  - Quality Assurance (/blog/category/quality-assurance)
  - Security Engineering (/blog/category/security-engineering)
  - System Administration (/blog/category/system-administration)
- Uncategorized (/blog/category/)

Tags

Access Controls (/blog/tag/access-controls) Application Security (/blog/tag/application-security) Authentication (/blog/tag/authentication) Authorization (/blog/tag/authorization) Automatic Updates (/blog/tag/automatic-updates) Business (/blog/tag/business) Central Florida (/blog/tag/central-florida) Cryptography (/blog/tag/cryptography) CSPRNG (/blog/tag/csprng) Data Science (/blog/tag/data-science) Encryption (/blog/tag/encryption) HTTPS (/blog/tag/https) Integrity (/blog/tag/integrity) Libsodium (/blog/tag/libsodium) Login (/blog/tag/login) Math (/blog/tag/math) .NET (/blog/tag/net) Networking (/blog/tag/networking) Node.js (/blog/tag/node-js) Open Source (/blog/tag/open-source) Orlando (/blog/tag/orlando) OWASP (/blog/tag/owasp) OWASP Top Ten (/blog/tag/owasp-top-ten) PHP (/blog/tag/php) PostgreSQL (/blog/tag/postgresql) Public Key Cryptography (/blog/tag/public-key-cryptography) Python (/blog/tag/python) Ruby (/blog/tag/ruby) Secret Key Cryptography (/blog/tag/secret-key-cryptography) Security (/blog/tag/security) Signatures (/blog/tag/signatures) SQL (/blog/tag/sql) SQL Injection (/blog/tag/sql-injection) Statistics (/blog/tag/statistics) Vulnerability (/blog/tag/vulnerability) Web Development (/blog/tag/web-development) XSS (/blog/tag/xss)

Mailing Lists

Paragon Initiative Quarterly (http://eepurl.com/bqnTnf)
Paragon Initiative Vanguard (http://eepurl.com/bqnRzf)

Elsewhere

- ParagonIE on Github (https://github.com/paragonie)
- @ParagonIE (https://twitter.com/ParagonIE)

☐ Paragonie on Facebook (https://facebook.com/ParagonIE)

Need Technology Consultants?

Will tomorrow bring **costly and embarrassing data breaches (https://paragonie.com/white-paper/2015-why-invest-application-security)**? Or will it bring growth, success, and peace of mind?

Our team of technology consultants have extensive knowledge and experience with application security and web/application development.

We specialize in **cryptography (https://paragonie.com/blog/tag/cryptography)** and **secure PHP development (https://paragonie.com/blog/tag/php)**.

Let's Work Together Towards Success (/contact)

Our Security Newsletters

Want the latest from Paragon Initiative Enterprises delivered straight to your inbox? We have two newsletters to choose from.

The first mails quarterly and often showcases our behind-the-scenes projects.

The other is unscheduled and gives you a direct feed into the findings of our open source security research initiatives.

Quarterly Newsletter (http://eepurl.com/bqnTnf)

Security Announcements (http://eepurl.com/bqnRzf)

(https://paragonie.com/contact) (https://facebook.com/paragonie)
(https://github.com/paragonie) (https://twitter.com/ParagonIE)