Failures in NIST’s ECC standards

Daniel J. Bernstein, Tanja Lange

2015.12.15
Review of the (prime-field) NIST curves I

- Presented by NIST in 1999
- Curve names: P-192, P-224, P-256, P-384, P-521
  - Curve is defined over $\mathbb{F}_p$ where $p$ has 192 bits, 224 bits, etc.
- Primes are pseudo-Mersenne primes:
  - e.g. P-224 prime is $2^{224} - 2^{96} + 1$
  - e.g. P-256 prime is $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
  - Why? Efficiency
    - NSA’s Jerry Solinas chose these curves and wrote papers about the speed of these primes
Review of the (prime-field) NIST curves I

- Presented by NIST in 1999
- Curve names: P-192, P-224, P-256, P-384, P-521
  - Curve is defined over $\mathbb{F}_p$ where $p$ has 192 bits, 224 bits, etc.
- Primes are pseudo-Mersenne primes:
  - e.g. P-224 prime is $2^{224} - 2^{96} + 1$
  - e.g. P-256 prime is $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Why? Efficiency
  - NSA’s Jerry Solinas chose these curves and wrote papers about the speed of these primes
  - Possible additional motivation: avoiding the Crandall patents (which expired in 2011)
Review of the (prime-field) NIST curves II

- Curve shape specifically $y^2 = x^3 - 3x + b$
  - About 50% of all curves
  - Absolutely nothing worrisome from an ECDLP perspective
  - “For reasons of efficiency”
    - cites IEEE P1363 standard
      - P1363 cites 1987 paper by Chudnovsky brothers
      - P1363 claims that its choices “provide the fastest arithmetic on elliptic curves”

- Cofactor choice:
  - NIST takes cofactor “as small as possible” for “efficiency reasons”
  - All cofactors for NIST curves are 1, 2, or 4
  - All cofactors for prime-field NIST curves are 1
Why did NIST choose these curves?

Most people we have asked: “security”

Actual NIST design document: “efficiency”

There are some minimal security requirements

Enough to make ECDLP hard

Not enough to make ECC secure

Amusing side notes regarding efficiency:

Addition formulas presented in standard are suboptimal, even for exactly these curves

NIST’s prime choices are suboptimal: $2^{255} - 19$ etc. are simpler and faster

Cofactor 4 is much more efficient than cofactor 1
Why did NIST choose these curves?

- Most people we have asked: “security”
- Actual NIST design document: “efficiency”
- There are some minimal security requirements
  - Enough to make ECDLP hard
  - Not enough to make ECC secure
- Amusing side notes regarding efficiency:
  - addition formulas presented in standard are suboptimal, even for exactly these curves
  - NIST’s prime choices are suboptimal: $2^{255} - 19$ etc. are simpler and faster
  - cofactor 4 is much more efficient than cofactor 1
What goes wrong with computing $kQ$?

- Simplest scalar-multiplication inner loop: $P \leftarrow P + P$; $P \leftarrow P + Q$ if current bit of $k$ is set
- Huge timing channel, but that’s not the only problem
- Simplest way to implement “+”: use the addition formulas
  \[
  \lambda = \frac{y_P - y_Q}{x_P - x_Q};
  x_3 = \lambda^2 - x_P - x_Q; y_3 = \lambda(x_P - x_3) - y_P
  \]
What goes wrong with computing $kQ$?

- Simplest scalar-multiplication inner loop: $P \leftarrow P + P$; $P \leftarrow P + Q$ if current bit of $k$ is set
- Huge timing channel, but that’s not the only problem
- Simplest way to implement “+”: use the addition formulas
  \[ \lambda = \frac{y_P - y_Q}{x_P - x_Q}; x_3 = \lambda^2 - x_P - x_Q; y_3 = \lambda(x_P - x_3) - y_P \]
  - But this doesn’t work for doublings; all tests fail
  - So implementor checks book, implements $\text{dbl}(P)$
- New inner loop: $P \leftarrow \text{dbl}(P)$; $P \leftarrow P + Q$ if current exponent bit is set
- This passes all tests but still has failure cases
  - e.g., what if $P = Q$? what if $P = -Q$?
- Maybe implementor instead has “+” check for $P = Q$
  - less likely: this is slower and more complicated code
  - doesn’t catch all the failure cases
- Attacker triggers the failure cases
  - Fancy example: Izu–Takagi “exceptional procedure attack”
Alternative: Montgomery curves $y^2 = x^3 + ax^2 + x$

- Use Montgomery ladder for scalar multiplication
  - per bit 1 doubling + 1 differential addition
  - differential addition: compute $P + Q$ given $P, Q, P - Q$
  - automatic uniform pattern independent of $n$; good against timing and simple side-channel attacks
- Represent a point as its $x$-coordinate
  - very fast doubling, very fast differential addition
  - faster scalar multiplication than $y^2 = x^3 - 3x + b$
  - for Montgomery curves that have unique point of order 2:
    - infinity and 0 behave the same way
    - the formulas always work (2006 Bernstein)
Any reasons not to choose Montgomery curves?

- Is security the same?
  - Cannot be very different
    - Every curve is a Montgomery curve over a small extension field
  - Almost half of all curves are Montgomery curves over the same field
    - Any serious attack on Montgomery curves would be huge ECC news
  - Cofactor for Montgomery curves is a multiple of 4
    - Requires slightly larger primes
- Limitation: only for single-scalar multiplication
  - Signature verification needs double-scalar multiplication
  - But no problem for DH, El Gamal, etc.
Does this work for the NIST curves?

- Not easily; NIST cofactor 1 is incompatible with Montgomery
- Can still try to imitate part of the Montgomery approach
- Double and always add
  - Slow, more complicated than standard approach
  - More smart-card trouble: extra vulnerability to fault attacks
- Can stop timing attacks but does nothing to fix failure cases
- Ladder
  - Representing point as $(x, y)$: very slow
  - Just $x$: not as slow (Brier–Joye, Hutter–Joye–Sierra) but still complicated
  - Maybe fixes failure cases; analysis has never been done
Problems with NIST curves as actually implemented

- What if input point $P$ is not on $E$ but on a different curve?
- Simplest implementation doesn’t check. What happens?
- Typical ECDH answer: successfully obtain $nP$ on that other curve; use $nP$ as shared secret to encrypt data
- Attacker chooses $P$ so that, e.g., $1009P = 0$; checks encryption, quickly figures out $n \mod 1009$
- Attacker figures out $n$ by CRT
Problems with NIST curves as actually implemented

- What if input point $P$ is not on $E$ but on a different curve?
- Simplest implementation doesn’t check. What happens?
- Typical ECDH answer: successfully obtain $nP$ on that other curve; use $nP$ as shared secret to encrypt data
- Attacker chooses $P$ so that, e.g., $1009P = 0$; checks encryption, quickly figures out $n \mod 1009$
- Attacker figures out $n$ by CRT
- Recent paper at ESORICS (Jager, Schwenk, Somorovsky): ECC implementations of Oracle and Bouncy Castle do not check for point on curve. Practical attack on ECC in TLS. http://www.nds.rub.de/research/publications/ESORICS15/
Countermeasures

- Countermeasure: send \((x, \text{bit}(y))\), recover \(y\) or fail.
- Simpler: send and use only \(x\) in Montgomery ladder.
  - Only two possible curves: \(E\) and its “nontrivial quadratic twist”
  - 2001 Bernstein: stop attack by choosing twist to be secure
  - Twist security might happen by accident, but random curves are usually less secure
  - NIST P-256 has a somewhat weaker twist (security \(2^{120.3}\))
  - NIST P-224 has a much weaker twist (security \(2^{58.4}\))
  - BrainpoolP256t1 has a much, much weaker twist (security \(2^{44.5}\))
Suggestions so far

- Choose Montgomery curves (with unique point of order 2)
- Represent points as $x$-coordinates
- In particular choose twist-secure curves
- Simple implementation is fine
- Main limitation: how to handle signatures?
Alternative: Edwards curves $x^2 + y^2 = 1 + dx^2y^2$

- Focus on complete Edwards curves: non-square $d$
  - about 25% of all elliptic curves
  - includes Curve25519; does not include the NIST curves
- Simplest addition law is complete
  - $x_3 = (x_1y_2 + x_2y_1)/(1 + dx_1x_2y_1y_2)$
  - $y_3 = (y_1y_2 - x_1x_2)/(1 - dx_1x_2y_1y_2)$
  - no exceptions: works for doubling, $P + (-P)$, etc.
  - easy to implement; It Just Works™
  - can implement separate doubling but don’t have to
  - also very fast (see http://hyperelliptic.org/EFD)
- Guarantees Montgomery compatibility
  - easy secure single-scalar multiplication
- Also good for other ECC protocols
  - simplest signature-verification implementation is fine
Problems with protocols

- Notation: public key $A$; signature $(R, S)$; message $M$ to verify; standard base point $B$ and curve and hash function $H$

- NIST’s ECDSA: verify $H(M)B + x(R)A = SR$

- Equivalent view: $B + H'(R, M)A = S'R$ with $H'(R, M) = x(R)/H(M)$
Problems with protocols

- Notation: public key \( A \); signature \((R, S)\); message \( M \) to verify; standard base point \( B \) and curve and hash function \( H \)
- NIST’s ECDSA: verify \( H(M)B + x(R)A = SR \)
- Equivalent view: \( B + H'(R, M)A = S'R \) with \( H'(R, M) = x(R)/H(M) \)
- Our EdDSA (Schnorr-based): verify \( SB = R + H(R, A, M)A \)
  - ECDSA needs divisions for signer etc.; EdDSA puts \( S \) in front of \( B \) rather than \( R \)
  - ECDSA isn’t resilient against collisions; EdDSA replaces weird \( H' \) with normal hash \( H \)
  - ECDSA has concerns regarding multi-key attacks; EdDSA includes \( A \) as an extra hash input
- ECDSA \( R \) gen: hard to audit, hard to test, Sony PS3 disaster; EdDSA generates \( R \) by deterministically hashing (secret, \( M \))
Summary

- ECDLP security does not guarantee ECC security
- Choose protocols carefully (ECDSA is horrible)
- Add extra requirements on curve choices
  - Recognize the importance of friendliness to implementors
  - NIST curves cause real trouble
- Require Montgomery compatibility (NIST curves flunk)
- Require Edwards compatibility (NIST curves flunk)
- Require completeness (NIST curves flunk)
- Require twist security (NIST curves are weak)
- Easy to generate curves meeting all these requirements: Curve25519, Curve41417, E-521, etc.
Will there ever be progress in the NIST ECC standards?

- We already presented this perspective in May 2013: http://cr.yp.to/talks.html#2013.05.31
- Many successful ECC timing attacks since then: e.g., https://eprint.iacr.org/2015/1141

- 2015.06: NIST ran a "Workshop on ECC Standards".
- 2015.10: NIST reopened its ECC standards for comments.
- We sent comments. Paper coming soon: "Failures in NIST's ECC standards."
- But is NIST trying to fix actual problems with ECC? Or is it focusing entirely on the possibility of back doors?
Will there ever be progress in the NIST ECC standards?

- We already presented this perspective in May 2013: http://cr.yp.to/talks.html#2013.05.31
- Many successful ECC timing attacks since then: e.g., https://eprint.iacr.org/2015/1141
- 2015.06: NIST ran a “Workshop on ECC Standards”.
- 2015.10: NIST reopened its ECC standards for comments.
- We sent comments.
  Paper coming soon: “Failures in NIST’s ECC standards.”
Will there ever be progress in the NIST ECC standards?

- We already presented this perspective in May 2013: http://cr.yp.to/talks.html#2013.05.31
- Many successful ECC timing attacks since then: e.g., https://eprint.iacr.org/2015/1141
- 2015.06: NIST ran a “Workshop on ECC Standards”.
- 2015.10: NIST reopened its ECC standards for comments.
- We sent comments. Paper coming soon: “Failures in NIST’s ECC standards.”
- But is NIST trying to fix actual problems with ECC? Or is it focusing entirely on the possibility of back doors?