Is elliptic-curve cryptography safe?

Daniel J. Bernstein

University of Illinois at Chicago, USA

Academia Sinica, Taiwan

No.

Shor's algorithm breaks ECC.

Wait a minute

Shor's algorithm needs a big quantum computer.
Public quantum computers are still too small.

Wait a minute

Shor's algorithm needs a big quantum computer.
Public quantum computers are still too small.

Hopefully the attacker isn't very far ahead of us.

Wait a minute

Shor's algorithm needs a big quantum computer.
Public quantum computers are still too small.

Hopefully the attacker isn't very far ahead of us.

Hopefully the attacker won't be able to afford quantum attacks against many targets.

Wait a minute

Shor's algorithm needs a big quantum computer.
Public quantum computers are still too small.

Hopefully the attacker isn't very far ahead of us.

Hopefully the attacker won't be able to afford quantum attacks against many targets.

Hopefully quantum computing will simply fail.

Is ECC safe against today's non-quantum attacks?

Some bad ways to answer this question

"ECC is popular so it's safe."
"ECC is standardized so it's safe."
"There are many papers on ECC so it's safe."

Some bad ways to answer this question

"ECC is popular so it's safe."
"ECC is standardized so it's safe."
"There are many papers on ECC so it's safe."

— Example of why these don't answer the question: RC4 was standardized for use in TLS, was popular, and was the topic of many papers, but it still allowed feasible plaintext-recovery attacks.

The actual answer

There have been many real-world ECC failures.
This talk will look at some examples.

The actual answer

There have been many real-world ECC failures.
This talk will look at some examples.

Fortunately, we know how to make ECC safer.
But overconfidence continues to damage security.

The actual answer

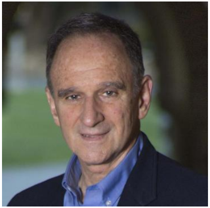
There have been many real-world ECC failures.
This talk will look at some examples.

Fortunately, we know how to make ECC safer.
But overconfidence continues to damage security.

This doesn't mean you should switch away from ECC: alternatives have had even more failures!

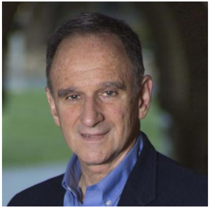
Why we're using ECC

1976 Diffie–Hellman key exchange

There's a standard prime q.
Alice has secret key a, public key $2^a \bmod q$.
Bob has secret key b, public key $2^b \bmod q$.
Alice and Bob compute shared secret $2^{ab} \bmod q$.

1976 Diffie–Hellman key exchange

There's a standard prime q.
Alice has secret key a, public key $2^a \bmod q$.
Bob has secret key b, public key $2^b \bmod q$.
Alice and Bob compute shared secret $2^{ab} \bmod q$.

Obvious attack strategy: try to compute b
given $2^b \bmod q$; i.e., compute "discrete logs".

Index calculus

1976 DH: “for certain carefully chosen values of q”, discrete-log computation “requires on the order of $q^{1/2}$ operations, using the best known algorithm”.

Index calculus

1976 DH: "for certain carefully chosen values of q", discrete-log computation "requires on the order of $q^{1/2}$ operations, using the best known algorithm".

No! 1922 Kraitchik discrete-log algorithm is much faster. (See also 1926 Kraitchik explaining how to use the same idea to factor big integers.)

Faster index calculus

1976–today: people keep finding ways to accelerate Kraitchik's "index calculus" idea, such as the "function-field sieve" and the "number-field sieve".

2020 computation: 30750-bit binary discrete log. Note: This uses a field of size $q = 2^{30750}$, not integer arithmetic mod q.

Primes q aren't as thoroughly broken, but still many speedups; e.g., 2020 computation for $q \approx 2^{795}$.

Defending against index calculus: ECC

ECC was introduced by 1986 Miller and independently 1987 Koblitz. Miller gave arguments that index calculus "is not likely to work on elliptic curves".

Koblitz gave a talk in 2000 describing some elliptic-curve features as a "golden shield protecting ECC".

Some ways that ECC has succeeded

Elliptic-curve discrete logarithms have maintained their security level much better than alternatives, so they inspire more confidence.

Some ways that ECC has succeeded

Elliptic-curve discrete logarithms have maintained their security level much better than alternatives, so they inspire more confidence.

ECC uses much smaller q than exponentiation, making it a more efficient choice.

Some ways that ECC has succeeded

Elliptic-curve discrete logarithms have maintained their security level much better than alternatives, so they inspire more confidence.

ECC uses much smaller q than exponentiation, making it a more efficient choice.

Most cryptographic applications today rely on ECC, although there are still some uses of RSA.

What has gone wrong

Index calculus for some elliptic curves

Independent papers in 1993, 1994, 1996: Curves with exactly $q - 1$ points are as weak as traditional discrete logs. Generalizations weaken more curves: e.g., curves with exactly $q + 1$ points.

("Multiplicative transfers"; "pairings".)

Even faster discrete logs for some curves

Independent papers in 1998, 1998, 1999:
Fast attack for elliptic curves with exactly q points.

("Additive transfers"; "SmartASS".)

Discrete-log speedups for more curves

1998: Very simple $\sqrt{2}$ speedup for all elliptic curves. (“Negation.”)

2000: More complicated version of the same idea; somewhat bigger speedups for some elliptic curves. (“Automorphisms.”)

Discrete-log speedups for more curves

1998: Very simple $\sqrt{2}$ speedup for all elliptic curves. ("Negation.")

2000: More complicated version of the same idea; somewhat bigger speedups for some elliptic curves. ("Automorphisms.")

1998, 2002, 2003, 2009, 2011, 2012: much bigger speedups for some elliptic curves using non-prime q. ("Weil descent"; "subfields".)

Summary of ECDLP impact

For almost all (q, E) pairs, the $\sqrt{2}$ speedup has been the only speedup in discrete-log algorithms. (Not counting small speedups in arithmetic.)

Summary of ECDLP impact

For almost all (q, E) pairs, the $\sqrt{2}$ speedup has been the only speedup in discrete-log algorithms. (Not counting small speedups in arithmetic.)

Does that sound comforting?

Summary of ECDLP impact

For almost all (q, E) pairs, the $\sqrt{2}$ speedup has been the only speedup in discrete-log algorithms. (Not counting small speedups in arithmetic.)

Does that sound comforting?

Some of the curves with bigger attack speedups are curves that had been proposed for usage! Example with $q = 2^{155}$; example with q points; example and another example with $q + 1$ points.

Turn-of-the-century standards

1999 ANSI X9.62, 2000 IEEE P1363, 2000 SEC 2, 2000 NIST FIPS 186-2, 2001 ANSI X9.63 specified how to choose curves for ECC.

Usually stayed far away from the curve structures that had led to bigger discrete-log speedups. Exception: still allowed non-prime q.

More elliptic-curve standards following similar approaches and requiring prime q: 2005 Brainpool, 2005 NSA Suite B, 2011 ANSSI FRP256V1.

Potential problem: curve manipulation

Is it possible that attackers figured out
other types of weak curves in the 1990s
and selected a secretly weak curve to standardize?

Potential problem: curve manipulation

Is it possible that attackers figured out
other types of weak curves in the 1990s
and selected a secretly weak curve to standardize?

Bad answer: "We cryptographers are so smart that we would definitely have found the attack."

Potential problem: curve manipulation

Is it possible that attackers figured out
other types of weak curves in the 1990s
and selected a secretly weak curve to standardize?

Bad answer: “We cryptographers are so smart that
we would definitely have found the attack.”

— Examples of why that’s a bad answer: OCB2
was published in 2004, standardized in 2009,
broken in 2019. XCBv2 was published in 2007,
standardized in 2010, broken in 2024. DSA
backdoor was dismissed by cryptographers in 1992
but was demonstrated in 2016 to be feasible.

Do "seeds" stop manipulation?

NIST P-256 came from Solinas at NSA.

Do “seeds” stop manipulation?

NIST P-256 came from Solinas at NSA.

The standards claimed that this curve was chosen “by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found”.

Do "seeds" stop manipulation?

NIST P-256 came from Solinas at NSA.

The standards claimed that this curve was chosen "by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found".

After Snowden revealed NSA's sabotage budget in 2013, Solinas claimed that he had "built all the seeds via hashing (SHA-1, I think) from the ASCII representation of a humorous message".

Do “seeds” stop manipulation?

NIST P-256 came from Solinas at NSA.

The standards claimed that this curve was chosen “by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found”.

After Snowden revealed NSA’s sabotage budget in 2013, Solinas claimed that he had “built all the seeds via hashing (SHA-1, I think) from the ASCII representation of a humorous message”.

Was the seed random? Or H(humorous message)?

Do “seeds” stop manipulation?

NIST P-256 came from Solinas at NSA.

The standards claimed that this curve was chosen “by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found”.

After Snowden revealed NSA’s sabotage budget in 2013, Solinas claimed that he had “built all the seeds via hashing (SHA-1, I think) from the ASCII representation of a humorous message”.

Was the seed random? Or H(humorous message)? Either way, could have searched for a weak curve.

Stopping curve manipulation

Curve25519 is $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$.

Stopping curve manipulation

Curve25519 is $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$.

Why $2^{255} - 19$: fastest prime around 256 bits.

Stopping curve manipulation

Curve25519 is $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$.

Why $2^{255} - 19$: fastest prime around 256 bits.

Why $y^2 = x^3 + Ax^2 + x$: fastest curve shape for ECDH. Uses formulas from 1987 Montgomery.

Stopping curve manipulation

Curve25519 is $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$.

Why $2^{255} - 19$: fastest prime around 256 bits.

Why $y^2 = x^3 + Ax^2 + x$: fastest curve shape for ECDH. Uses formulas from 1987 Montgomery.

Why 486662: first A satisfying security criteria.

Stopping curve manipulation

Curve25519 is $y^2 = x^3 + 486662x^2 + x$ mod $2^{255} - 19$.

Why $2^{255} - 19$: fastest prime around 256 bits.

Why $y^2 = x^3 + Ax^2 + x$: fastest curve shape for ECDH. Uses formulas from 1987 Montgomery.

Why 486662: first A satisfying security criteria.

Only a few choices here that I was controlling: choosing 256; choosing details of criteria.

Definite problem: invalid-curve attacks

Attack strategy pointed out in 2000. Vulnerability announcements: 2015 (Java TLS, Bouncy Castle), 2015 (OpenSSL), 2019 (Bluetooth), 2019 (OpenPGP.js), 2019 (WPA3 Dragonfly), 2022 (openCryptoki), 2023 (free5GC udm).

Definite problem: invalid-curve attacks

Attack strategy pointed out in 2000. Vulnerability announcements: 2015 (Java TLS, Bouncy Castle), 2015 (OpenSSL), 2019 (Bluetooth), 2019 (OpenPGP.js), 2019 (WPA3 Dragonfly), 2022 (openCryptoki), 2023 (free5GC udm).

Attacker sends points (x, y) that aren't on the curve; observes Bob's reaction to those points; computes Bob's secret key from those reactions.

Definite problem: invalid-curve attacks

Attack strategy pointed out in 2000. Vulnerability announcements: 2015 (Java TLS, Bouncy Castle), 2015 (OpenSSL), 2019 (Bluetooth), 2019 (OpenPGP.js), 2019 (WPA3 Dragonfly), 2022 (openCryptoki), 2023 (free5GC udm).

Attacker sends points (x, y) that aren't on the curve; observes Bob's reaction to those points; computes Bob's secret key from those reactions.

Note: ECC software without point-on-curve checks will pass typical tests of software correctness.

Stopping invalid-curve attacks

X25519 (Curve25519 ECDH) sends x, not (x, y).
Each possible x is on Curve25519 or its "twist".
Furthermore, Curve25519 is "twist-secure".
This eliminates the leak of information to attackers, even for software without point-on-curve checks.

Stopping invalid-curve attacks

X25519 (Curve25519 ECDH) sends x, not (x, y).
Each possible x is on Curve25519 or its "twist".
Furthermore, Curve25519 is "twist-secure".
This eliminates the leak of information to attackers,
even for software without point-on-curve checks.

"Let's use just x for NIST P-256 too!
Its twist-security level turns out to be high enough!"

Stopping invalid-curve attacks

X25519 (Curve25519 ECDH) sends x, not (x, y).
Each possible x is on Curve25519 or its “twist”.
Furthermore, Curve25519 is “twist-secure”.
This eliminates the leak of information to attackers, even for software without point-on-curve checks.

“Let’s use just x for NIST P-256 too!
Its twist-security level turns out to be high enough!”

— Unfortunately NIST P-256 doesn’t have shape $y^2 = x^3 + Ax^2 + x$. It has shape $y^2 = x^3 - 3x + B$.
This turns out to complicate x-coordinate software.
Implementors will tend to take the simplest option.

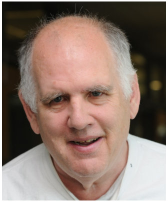
Another definite problem: timing attacks

Vulnerability announcements: 2009 (OpenSSL), 2011 (OpenSSL), 2015 (OpenSSL), 2019 (FIPS-certified CC-certified Athena IDProtect smart card, 7 other certified devices, 4 out of 13 software libraries), 2019 (two certified TPMs), 2020 (mbedTLS), 2020 (OpenSSL), 2023 (Firefox).

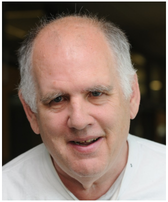
Standard NIST P-256 computations involve many special cases, taking time that depends on secrets.

Stopping timing attacks

Montgomery's formulas for x-coordinate computations on $y^2 = x^3 + Ax^2 + x$ turn out to work for all inputs. No special cases!



Stopping timing attacks



Montgomery's formulas for x-coordinate computations on $y^2 = x^3 + Ax^2 + x$ turn out to work for all inputs. No special cases!

Warning: Still have to be careful to carry out arithmetic mod q in constant time. The Python code on the next slide isn't safe: Python has variable-time arithmetic.

Complete X25519 software

```
def bit(x,i): return 1&(x>>i)
def cswap(x,X,b): return x+b*(X-x),X-b*(X-x)
def montgomery_ladder(P,n):
  q,A,x,z,X,Z = 2**255-19,486662,1,0,P,1
  for i in reversed(range(255)):
    ni = bit(n,i)
    x,X = cswap(x,X,ni)
    z,Z = cswap(z,Z,ni)
    X,Z = 4*(x*X-z*Z)**2,4*P*(x*Z-z*X)**2
    x,z = (x**2-z**2)**2,4*x*z*(x**2+A*x*z+z**2)
    X,Z = X%q,Z%q
    x,z = x%q,z%q
    x,X = cswap(x,X,ni)
    z,Z = cswap(z,Z,ni)
  return (x*pow(z,q-2,q))%q
```

Stopping timing attacks for signatures

Signatures (unlike DH) are easier using (x, y).
("Multi-scalar" mult, not "single-scalar" mult.)

Stopping timing attacks for signatures

Signatures (unlike DH) are easier using (x, y).
("Multi-scalar" mult, not "single-scalar" mult.)

2007 Edwards: new (x, y) addition formulas. 2007 Bernstein–Lange: for Curve25519, the Edwards formulas are faster than traditional (x, y) formulas, and have no special cases!

Stopping timing attacks for signatures

Signatures (unlike DH) are easier using (x, y).
("Multi-scalar" mult, not "single-scalar" mult.)

2007 Edwards: new (x, y) addition formulas. 2007 Bernstein–Lange: for Curve25519, the Edwards formulas are faster than traditional (x, y) formulas, and have no special cases!

2011 Bernstein–Duif–Lange–Schwabe–Yang: EdDSA/Ed25519 signatures using Edwards formulas. (Also avoiding some other ECDSA flaws.)

Lessons

The pattern of standards failing

Implementors typically try to write simple code, then modify it until it passes some tests. Sometimes also modify it for speed.

But ECC software has tensions between simplicity, speed, and security.

The standards assume perfect implementations, rather than being designed to minimize tensions.

SafeCurves

2013 Bernstein–Lange: “The core problem is that **if you implement the standard curves, chances are you’re doing it wrong** . . . These problems are exploitable by real attackers, taking advantage of the gaps between ECDLP and real-world ECC . . . Secure implementations of the standard curves are theoretically possible but very hard.”

SafeCurves, continued

“Most of these attacks would have been ruled out by better choices of curves that allow *simple* implementations to be *secure* implementations. This is the primary motivation for SafeCurves. **The SafeCurves criteria are designed to ensure ECC security, not just ECDLP security.**”

SafeCurves, continued

"Most of these attacks would have been ruled out by better choices of curves that allow *simple* implementations to be *secure* implementations. This is the primary motivation for SafeCurves. **The SafeCurves criteria are designed to ensure ECC security, not just ECDLP security.**"

Criteria are summarized on next two slides.
See SafeCurves page for full details;
see also the new SafeCurves paper.

SafeCurves includes the standard criteria

Basics:

- “field”: q is prime.
- “equation”: curve is elliptic.
- “base”: standard base point has prime order ℓ.

Standard ECDLP criteria:

- “rho”: ℓ is big enough.
- “transfer”: curve does not have q points; curve’s “embedding degree” is big.
- “disc”: curve “discriminant” is big.

SafeCurves includes extra criteria

Protecting against curve manipulation:

- "rigid": limited choices in curve generation.

Keeping implementations safe:

- "ladder": simplicity and speed don't push implementors of this curve away from constant-time x-coordinate ECDH.
- "twist": curve twist-security is high enough.
- "complete": simplicity and speed don't push implementors of this curve into computations that need special cases.
- "ind": a way to view strings as curve points.

Success metric 1: security

2013–today: The failure patterns predicted by SafeCurves show up repeatedly in vulnerability announcements for ECC signatures and ECDH.

Example: 9 of the software libraries targeted in the 2019 "Minerva" attack paper supported both ECDSA and EdDSA; Minerva broke 3 of the ECDSA implementations, 0 of the EdDSA implementations.

Success metric 2: safe deployment

Nicolai Brown maintains massive lists of applications of X25519 and Ed25519.

Some applications already had P-256 but added Curve25519 as an extra option. Example: TLS, where X25519 turned into the "most commonly used key exchange algorithm" despite not being required by the TLS standard.

Closing words

2019 NIST, regarding the NIST curves: “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.”

— What if they *aren't* implemented correctly? Welcome to the real world!