A gentle introduction to elliptic curve cryptography

Craig Costello

Tutorial at SPACE 2016
December 15, 2016
CRRao AIMSCS, Hyderabad, India
Part 1: Diffie-Hellman key exchange

Part 2: Elliptic Curves

Part 3: Elliptic Curve Cryptography

Part 4: Next-generation ECC
Cryptography Pioneers Receive 2015 ACM A.M. Turing Award

Whitfield Diffie, former Chief Security Officer of Sun Microsystems and Martin E. Hellman, Professor Emeritus of Electrical Engineering at Stanford University, are the recipients of the 2015 ACM A.M. Turing Award, for critical contributions to modern cryptography. The ability for two parties to communicate privately over a secure channel is fundamental for billions of people around the world. On a daily basis, individuals establish secure online connections with banks, e-commerce sites, email servers and the cloud. Diffie and Hellman’s groundbreaking 1976 paper, “New Directions in Cryptography,” introduced the ideas of public-key cryptography. This groundbreaking work laid the foundation for a wide range of cryptographic protocols and technologies that are now commonplace in our daily lives.

The key concept of public-key cryptography is that it is possible for two parties to communicate securely without having to share a secret key in advance. This is achieved by using a pair of keys: a public key and a private key. The public key is used for encryption, while the private key is used for decryption. This allows for secure communication even if the public keys are intercepted, because only the holder of the corresponding private key can decrypt the message.

This innovation has had a profound impact on technology and society, enabling secure transactions, data protection, and privacy in digital communications. The winners’ contributions have been instrumental in shaping the field of cryptography and have had a lasting impact on the way we live and interact in the digital world.

While the details of their contributions are beyond the scope of this description, the essence of their work is encapsulated in the quote of the award citation: "The key concept of public-key cryptography is that it is possible for two parties to communicate securely without having to share a secret key in advance. This is achieved by using a pair of keys: a public key and a private key. The public key is used for encryption, while the private key is used for decryption. This allows for secure communication even if the public keys are intercepted, because only the holder of the corresponding private key can decrypt the message."

The impact of their work is evident in the everyday technologies we use today, from secure online transactions to encrypted communications. Diffie and Hellman’s contributions have been foundational to the development of modern cryptography and have enabled the secure exchange of information in a digital world.
Diffie-Hellman key exchange (circa 1976)

\[ q = 1606938044258990275541962092341162602522202993782792835301301 \]
\[ g = 123456789 \]

\[ g^a \mod q = 78467374529422653579754596319852702575499692980085777948593 \]

\[ 560048104293218128667441021342483133802626271394299410128798 = g^b \mod q \]

\[ g^{ab} \mod q = 437452857085801785219961443000845969831329749878767465041215 \]
Diffie-Hellman key exchange

\[ q = \frac{b^a - 1}{g - 1} \mod (p-1) \]

\[ g = 123456789 \]

\[ a = 4110646260959306603228256534418724107799729052799374597273156374202838733274422199664649346598210531138312495219675349162480658943577688915515145004091855165941297577649
\]

\[ b = 8546689595403495367524570897382960361348806197884142951156270556460232662740670140770771577243821612717424612212155678
\]

\[ g^a = 297466481822321973826628162814520555971979977625337636604981749987557545466750421858781051331382174972869890599554928429546567899476
\]

\[ g^b = 5361520786494346436747356336991697369597711878156082988753585541201560352299619671453728721182347574623380979101145205395
\]

\[ g^{ab} = 9749777061214595246975781881435745924202799120106345534546726874169963933631163136993714
\]

\[ 7161528131592202456756244795831957915666646660388394491441645570071245755172185744714
\]

\[ 9925438234061629205790751175533888161918978925955135366997012922676856551734507918082154443647380620102971803249539670984198853158111269773047896704857043710
\]
Individual secret keys secure under Discrete Log Problem (DLP): 
\[ g, g^x \rightarrow x \]

Shared secret secure under Diffie-Hellman Problem (DHP): 
\[ g, g^a, g^b \rightarrow g^{ab} \]

Fundamental operation in DH key exchange is group exponentiation: 
\[ g, x \rightarrow g^x \]
Done via “square-and-multiply”, e.g., \((x)_2 = (1,0,1,1,0,0,0,1 \ldots)\)

We are working “mod q”, but only with one operation: multiplication

Actually, fundamental operation in all public-key cryptography (key exchange, signatures, encryption, etc) is group exponentiation

Main reason for fields being so big: (sub-exponential) index calculus attacks!
DH key exchange (Koblitz-Miller style)

If all we need is a group, why not use elliptic curve groups?

Rationale: “it is extremely unlikely that an index calculus attack on the elliptic curve method will ever be able to work” [Miller, 85]
Real-world (e.g., Internet/TLS) cryptography in one slide (oversimplified)

- Public-key cryptography used to
  1. establish a shared secret key (e.g., Diffie-Hellman key exchange)
  2. authenticate one another (e.g., digital signatures)
- Symmetric key cryptography uses shared secret to encrypt/authenticate the subsequent traffic (e.g., block ciphers, AES/DES, stream ciphers, MACs)
- Hash functions used throughout (e.g., SHA’s, Keccak)
Real-world (e.g., Internet/TLS) cryptography in one slide (oversimplified)

- Public-key cryptography used to:
  - Establish a shared secret key (e.g., Diffie-Hellman key exchange)
  - Authenticate one another (e.g., digital signatures)

- Symmetric key cryptography uses shared secret to encrypt/authenticate the subsequent traffic (e.g., block ciphers, AES/DES, stream ciphers, MACs)

- Hash functions used throughout (e.g., SHA's, Keccak)
Some good references

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curves</td>
<td>Sutherland’s MIT course on elliptic curves: <a href="https://math.mit.edu/classes/18.783/2015/lectures.html">https://math.mit.edu/classes/18.783/2015/lectures.html</a></td>
</tr>
</tbody>
</table>
group $(G, +)$ can do $+$ $-$

ring $(R, +, \times)$ can do $+$ $-$ $\times$

field $(F, +, \times)$ can do $+$ $-$ $\times$ $\div$
If you’ve never seen an elliptic curve before....

Remember: an elliptic curve is a group defined over a field

elliptic curve group \((E, \oplus)\) can do \(\oplus \Theta \Theta\)
underlying field \((K, +, \times)\) can do \(+ \quad - \quad \times \quad \div\)

operations in underlying field are used and combined to compute the elliptic curve operation \(\Theta\)
Boring curves

\[ f(x, y) = 0 \quad \text{or} \quad f(X, Y, Z) = 0 \]

Degree 1 (lines)

\[ ax + by = c \quad \text{or} \quad ab \neq 0 \]

Degree 2 (conic sections)

\[ ax^2 + bxy + cy^2 + dx + ey + f = 0 \quad \text{or} \quad abc \neq 0 \]

e.g., ellipses, hyperbolas, parabolas

• “Genus” measures geometric complexity, and both are genus 0
• We know how to describe all solutions to these, e.g., over \( \mathbb{Q} \)
• Not cryptographically interesting
Elliptic curves

- Degree 3 is where all the fun begins...
  \[ ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j = 0 \]
  \[ ch(K) \neq 2,3 \]
  \[ E/K: \ y^2 = x^3 + ax + b \]
- Elliptic curves ↔ genus 1 curves
- Set of points \((x, y) \in K \times K\) satisfying above equation
- Geometrically/arithmeticly/cryptographically interesting
- Fermat’s last theorem/BSD conjecture/...
Elliptic curves, pictorially

\[ E/\mathbb{R} : y^2 = x^3 + x + 1 \]

\[ E/\mathbb{R} : y^2 = x^3 - x \]
Elliptic curves are groups

• So $E$ is a set, but to be a group we need an operation

• The operation is between points $(x_P, y_P) \oplus (x_Q, y_Q) = (x_R, y_R)$

• Remember: a group $(E, \oplus)$ defined over a field $(K, +, \times)$

• $K$ will be fields we’re used to, e.g., $\mathbb{Q}, \mathbb{C}, \mathbb{R}, \mathbb{F}_p$

• Remember: the (boring) operations $+, -, \times, \div$ in $K$ are used to compute the (exotic) operation $\oplus$ on $E$
Elliptic curve group law is easy

**Fun fact:** homomorphism between Jacobian of elliptic curve and elliptic curve itself.

**Upshot:** you don’t have to know any algebraic geometry (e.g., what a Jacobian is) to understand/do elliptic curve cryptography.
The elliptic curve group law $\oplus$

$$(x_p, y_p) \oplus (x_q, y_q) = (x_r, y_r)$$

**Question:** Given two points lying on a cubic curve, how can we use their coordinates to give a third point lying on the curve?
The elliptic curve group law $\oplus$

We need $(x_P, y_P) \oplus (x_Q, y_Q) = (x_R, y_R)$

**Question:** Given two points lying on a cubic curve, how can we use their coordinates to give a third point lying on the curve?

**Answer:** A line that intersects a cubic twice must intersect it again, so we draw a line through the points $(x_P, y_P)$ and $(x_Q, y_Q)$
The elliptic curve group law $\oplus$

$R = P \oplus Q$

$R = P \oplus P$
The elliptic curve group law $\oplus$

\[ y = \lambda x + \nu \quad \cap \quad y^2 = x^3 + ax + b \]
\[ x^3 - (\lambda x + \nu)^2 + ax + b = 0 \]
\[ x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + (b - \nu^2) = (x - x_P)(x - x_Q)(x - x_R) \]

\[ x_R = \lambda^2 - x_1 - x_2 \]
\[ y_R = -(\lambda x_R + \nu) \]

\[ \lambda = \frac{y_2 - y_1}{x_2 - x_1} \]
\[ \lambda = \frac{dy}{dx} = \frac{3x^2 + a}{2y} \]
A toy example

\[ E/\mathbb{R} : y^2 = x^3 - 2x \]

What about \[ E/\mathbb{Q} : y^2 = x^3 - 2 \]?
The (abelian) group axioms

- **Closure**: the third point of intersection must be in the field

- **Identity**: $E_{a,b}(K) = \{(x, y) : y^2 = x^3 + ax + b\} \cup \{\infty\}$

- **Inverse**: $\Theta (x, y) = (x, -y)$

- **Associative**: proof by picture

- **Commutative**: line through $P$ and $Q$ same as line through $Q$ and $P$
A toy example, cont.

\[ E/\mathbb{F}_{11}: y^2 = x^3 - 2x \]

\[(7,5) \oplus (8,10) = (10,1)\]
Scalar multiplications via double-and-add

How to (naively) compute \( k, Q \mapsto [k]Q \) ?

\[ P \leftarrow Q \]

\[ k = (k_n, k_{n-1}, ..., k_0)_2 \]

for \( i \) from \( n - 1 \) downto 0 do

\[ P \leftarrow [2]P \quad \text{DBL} \]

if \( k_i = 1 \) then

\[ P \leftarrow P \oplus Q \quad \text{ADD} \]

end if

end for

return \( P \ (= [k]Q) \)
Scalar multiplications via double-and-add

How to compute $k, Q \mapsto [k]Q$ on $y^2 = x^3 + ax + b$?

$k = (k_n, k_{n-1}, \ldots, k_0)$

$(x_P, y_P) \leftarrow Q$

for $i$ from $n - 1$ downto 0 do

    $\lambda \leftarrow (3x_P^2 + a)/(2y_P)$; 
    $\nu \leftarrow y_P - \lambda x_P$;

    $x_P \leftarrow \lambda^2 - 2x_P$; 
    $y_P \leftarrow -(\lambda x_P + \nu)$;

    if $k_i = 1$ then

        $\lambda \leftarrow (y_P - y_Q)/(x_P - x_Q)$; 
        $\nu \leftarrow y_P - \lambda x_P$;

        $x_P \leftarrow \lambda^2 - x_P - x_Q$; 
        $y_P \leftarrow -(\lambda x_P + \nu)$

    end for

return $(x_P, y_P) = [k](x_Q, y_Q)$
Projective space

- Recall we defined the group of $K$-rational points as
  \[ E_{a,b}(K) = \{(x, y) : y^2 = x^3 + ax + b\} \cup \{\infty\} \]

- The *natural habitat* for elliptic curve groups is in $\mathbb{P}^2(K)$, not $\mathbb{A}^2(K)$

- For (easiest) example, rather than $(x, y) \in \mathbb{A}^2$, take $(X:Y:Z) \in \mathbb{P}^2$ modulo the equivalence $(X:Y:Z) \sim (\lambda X : \lambda Y : \lambda Z)$ for $\lambda \in K^*$

- Replace $x$ with $X/Z$ and $y$ with $Y/Z$, so $E_{a,b}(K)$ is the set of solutions $(X:Y:Z) \in \mathbb{P}^2(K)$ to
  \[ E : \quad Y^2Z = X^3 + aXZ^2 + bZ^3 \]

- So the affine points $(x, y)$ from before become $(x : y : 1) \sim (\lambda x : \lambda y : \lambda)$ and the point at infinity is the unique point with $Z = 0$, i.e., $(0 : 1 : 0) \sim (0 : \lambda : 0)$
Projective space, cont.

- One practical benefit of working over $\mathbb{P}^2$ is that the explicit formulas for computing $\oplus$ become much faster, by avoiding field inversions.

- Thus, the fundamental ECC operation $k, P \mapsto [k]P$ becomes much faster...

\[
(x', y') = [2](x, y)
\]

\[
\begin{align*}
\lambda &\leftarrow (3x^2 + a)/(2y); \\
x' &\leftarrow \lambda^2 - 2x; \\
y' &\leftarrow -(\lambda(x' - x) + y); \\
1S + 2M + 1I
\end{align*}
\]

\[
(X' : Y' : Z') = [2](X : Y : Z)
\]

\[
\begin{align*}
X' &= 2XY((3X^2 + aZ^2)^2 - 8Y^2XZ) \\
Y' &= (3X^2 + aZ^2)(12Y^2XZ - (3X^2 + aZ^2)^2) - 8Y^4Z^2 \\
Z' &= 8Y^3Z^3 \\
5M + 6S
\end{align*}
\]
Projective scalar multiplications

How to compute $k, Q \mapsto [k]Q$ on $y^2 = x^3 + ax + b$?

$k = (k_n, k_{n-1}, ..., k_0)$

$\begin{align*}
(X_P : Y_P : Z_P) &\leftarrow Q \\
\text{for } i \text{ from } n - 1 \text{ downto } 0 \text{ do} & \\
(X_P : Y_P : Z_P) &\leftarrow [2](X_P : Y_P : Z_P) & 5M + 6S \\
\text{if } k_i = 1 \text{ then} & \\
(X_P : Y_P : Z_P) &\leftarrow (X_P : Y_P : Z_P) \oplus (X_Q : Y_Q : Z_Q) & 9M + 2S \\
\text{end for} & \\
\text{return} & (x_P, y_P) \leftarrow (X_P/Z_P, Y_P/Z_P) & 1I + 2M
\end{align*}$
Part 1: Diffie-Hellman key exchange

Part 2: Elliptic Curves

Part 3: Elliptic Curve Cryptography

Part 4: Next-generation ECC
Diffie-Hellman key exchange (circa 2016)

\[ g = 123456789 \]

\[ g^a = 234567890 \pmod{q} \]

\[ g^b = 987654321 \pmod{q} \]

\[ g^{ab} = 345678901 \pmod{q} \]
NIST Curve P-256

RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE

July 1999

This collection of elliptic curves is recommended for Federal government use and contains choices of private key lengths and underlying fields.

§1. Parameter Choices

§2. Curves over Prime Fields

For each prime $p$, a pseudo-random curve

$$E : \quad y^2 \equiv x^3 - 3x + b \pmod{p}$$
ECDH key exchange (1999 – nowish)

\[ p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \]
\[ p = 11579208921035624876677723109536672553920946131882517347620712973 \]

\[ E/\mathbb{F}_p: y^2 = x^3 - 3x + b \]
\[ \#E = 11579208921035624876677723109536672553920946131882517347620712973 \]

\[ P = (4843956129390645175905258525797914202762949526041747995844080717082404635286, \]
\[ 36134259567497959858512791958788195661106672985015071877198253568414405109) \]

\[ aP = (8411620826131589167593067868200525612344221886333785331584793435449501658416, \]
\[ 102885655542185598026739250172885300109680266058548048621945393128043427650740) \]

\[ [a]P = (10122888292005762667970413154540793024589549154209098999577542687271695288383, \]
\[ 778874181903040229941165950345562577670807185615679689372138134363978498341594) \]

\[ [b]P = (10122888292005762667970413154540793024589549154209098999577542687271695288383, \]
\[ 778874181903040229941165950345562577670807185615679689372138134363978498341594) \]

\[ [a]P + [b]P = (10122888292005762667970413154540793024589549154209098999577542687271695288383, \]
\[ 778874181903040229941165950345562577670807185615679689372138134363978498341594) \]
ECDH key exchange (1999 – nowish)

\[ p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \]
\[ p = 115792089210356248766977446749470573530086143415290314195533631308867097853951 \]

\[ E/\mathbb{F}_p : y^2 = x^3 - 3x + b \]
\[ \#E = 115792089210356248766974467494705735329996955224135760342422259061068512044369 \]

\[ P = (48439561293906451759052585252797914202762949526041747995844080717082404635286, \]
\[ 36134250956749795798585127919587881956611106672985015071877198253568414405109) \]

\[ a = 89130644591246033577639 \]
\[ 77064146285502314502849 \]
\[ 28352556031837219223173 \]
\[ 24614395 \]

\[ a \cdot P = (8411620826131589167593067868200525612344221886333785331584793435449501658416, \]
\[ 10288565542185598026739250172885300109680266058548048621945393128043427650740) \]

\[ b = 1095557463932786418806 \]
\[ 93831619070803277191091 \]
\[ 90584053916797810821934 \]
\[ 05190826 \]

\[ b \cdot P = (10122888292005762667970413154540793024589549154209098999577542687271695288383, \]
\[ 77887418190304022994116595034556257760807185615679689372138134363978498341594) \]

\[ ab \cdot P = (10122888292005762667970413154540793024589549154209098999577542687271695288383, \]
\[ 77887418190304022994116595034556257760807185615679689372138134363978498341594) \]

**Question 1:** how to compute \( \#E \)?

**Question 2:** why \( p \approx \#E \approx 2^{256} \)?
#E and Schoof's algorithm

- Given $E/F_p : y^2 = x^3 + ax + b$ (i.e., given $p, a, b$), how do we compute $#E(F_p)$?

- Hasse principle: $#E(F_p) = p + 1 - t$, where $-2\sqrt{p} \leq t \leq 2\sqrt{p}$, so $#E$ is (relatively) close to $p$, but exponentially many possible $t$.

- Schoof’s algorithm (unlocks ECC): compute $t \mod \ell_i$ for many small primes $\ell_i$ until $\prod_i \ell_i > 4\sqrt{p}$, so $t$ uniquely determined in Hasse interval.
Handwaving Schoof’s algorithm

- The key to Schoof’s algorithm lies in computing \( t \mod \ell \)
- For all \((x, y) \in E(\mathbb{F}_p)\), the trace \( t \) satisfies
  \[
  \left( x^{p^2}, y^{p^2} \right) - \left[ t \right] (x^p, y^p) + \left[ p \right] (x, y) = \infty
  \]
- The \( \ell \)-division polynomial (more later), \( \Phi_\ell \in \mathbb{F}_p[a, b, x, y] \) vanishes precisely at the points that vanish under multiplication by \( \ell \)
- Schoof: work indeterminately in \( \mathbb{F}_p[x, y]/\langle \Phi_\ell, E \rangle \), and replace \( p \) with \( p \mod \ell \) to recover \( t \mod \ell \)
Finding secure curves for ECC

• Given $p, a, b$, Schoof computes $#E_{a,b}(\mathbb{F}_p)$ in $O(\log(p)^8)$ steps

• General philosophy: find a prime of the appropriate bitlength, and iterate through $a$ and $b$ until $#E_{a,b}(\mathbb{F}_p)$ is (almost) prime. E.g.,

  - **NIST**: fixed $p$ special, $a = -3$, iterated $b$ as hash output until $#E$ prime.
  - **Brainpool**: $p, a, b$ all output of iterated hash functions, until $#E$ prime.

• Once (almost) prime order curve chosen, double-check other (exponentially unlikely) properties, e.g., low MOV degree, $#E \neq p$, etc.

• What do we mean by *appropriate bitlength*?
ECDLP security and Pollard’s rho algorithm

- The best known ECDLP algorithm on (well-chosen) elliptic curves remains generic, i.e., elliptic curves are as strong as is possible!

- ECDLP: given $P, Q \in E(\mathbb{F}_p)$ of prime order $N$, find $k$ such that $Q = [k]P$

- Pollard’78: compute pseudo-random $R_i = [a_i]P + [b_i]Q$ until we find a collision $R_i = R_j$ with $b_i \neq b_j$, then $k = (a_j - a_i)/(b_i - b_j)$

- Birthday paradox says we can expect collision after computing $\sqrt{\frac{\pi N}{2}}$ group elements $R_i$, i.e., after $\approx \sqrt{N}$ group operations.
Summary so far

• Elliptic curves are the only useful groups we know that are as secure as a black-box group. Upshot: use them for public-key cryptography!

• Old school method to setup ECC (e.g., ECDH):
  * choose a prime $p$ twice the length of your target security
  * find $a$ and $b$ such that $\#E_{a,b}(\mathbb{F}_p)$ is prime (and check stuff)
  * publish $E_{a,b}/\mathbb{F}_p$ and a prime order generator $P$

• Old school method to compute $k, P \mapsto [k]P$, etc.
  * work in projective space, e.g., $(x, y) = \left(\frac{X}{Z}, \frac{Y}{Z}\right)$ or $(x, y) = \left(\frac{X}{Z^2}, \frac{Y}{Z^3}\right)$
  * compute $[k]P$ via a sequence of doublings and additions
Questions so far?
Part 1: Diffie-Hellman key exchange

Part 2: Elliptic Curves

Part 3: Elliptic Curve Cryptography

Part 4: Next-generation ECC
What’s wrong with old school ECC?

• **Side-channel attacks**: starting with Kocher’99, side-channel attacks and their countermeasures have become extremely sophisticated (cf. Lejla’s tutorials from yesterday and a bunch of talks here!)

• **Decades of new research**: we now know much better/faster/simpler/safer ways to do ECC

• **Suspicion surrounding previous standards**: Snowden leaks, dual EC-DRBG backdoor, etc., lead to conjectured weaknesses in the NIST curves
NSA Curve P-256

“I no longer trust the constants. I believe the NSA has manipulated them”

Bruce Schneier (2013)

“So, sigh, why didn’t they do it that way? Do they want to be distrusted?”

Mike Scott (1999)
Next generation elliptic curves

- 2014: CFRG receives formal request from TLS working group for recommendations for new elliptic curves
- 2015: NIST holds workshop on ECC standards
- 2015: CFRG announces two chosen curves, both specified in Montgomery (1987) form

$$E/\mathbb{F}_p : y^2 = x^3 + Ax^2 + x$$

- Bernstein’s Curve25519 [2006]: $p = 2^{255} - 19$ and $A = 486662$
- Hamburg’s Goldilocks [2015]: $p = 2^{448} - 2^{224} - 1$ and $A = 156326$
- Both primes offer fast software implementations!
- Their group orders are divisible by 8 and 4, but this form offers several advantages.
Montgomery’s fast differential arithmetic

\[ E/\mathbb{F}_p : y^2 = x^3 + Ax^2 + x \]

- drop the \( y \)-coordinate, and work with \( x \)-only.
- projectively, work with \( (X : Z) \in \mathbb{P}^1 \) instead of \( (X : Y : Z) \in \mathbb{P}^2 \)
- But (pseudo-)addition of \( x(P) \) and \( x(Q) \) requires \( x(Q \ominus P) \)

Extremely fast pseudo-doubling: \( x\text{DBL} \)

\[
\begin{align*}
X_{[2]P} &= (X_P + Z_P)^2(X_P - Z_P)^2 \\
Z_{[2]P} &= 4X_PZ_P((X_P - Z_P)^2 + (A + 2)X_PZ_P)
\end{align*}
\]

Extremely fast pseudo-addition: \( x\text{ADD} \)

\[
\begin{align*}
X_{P+Q} &= Z_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)\right]^2 \\
Z_{P+Q} &= X_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)\right]^2
\end{align*}
\]

\[ 2M + 2S \quad \text{and} \quad 4M + 2S \]
Differential additions and the Montgomery ladder

- Given only the \( x \)-coordinates of two points, the \( x \)-coordinate of their sum can be two possibilities.
- Inputting the \( x \)-coordinate of the difference resolves ambiguity.
- The (ingenious!) Montgomery ladder fixes all differences as the input point: in \( k, x(P) \mapsto x([k]P) \), every \( \text{xADD} \) is of the form \( \text{xADD}(x([n+1]P), x([n]P), x(P)) \).
- We carry two multiples of \( P \) “up the ladder”: \( x(Q) \) and \( x(Q \oplus P) \).
- At \( i^{\text{th}} \) step: compute \( x([2]Q \oplus P) = \text{xADD}(x(Q \oplus P), x(Q), x(P)) \).
- At \( i^{\text{th}} \) step: pseudo-double (\( \text{xDBL} \)) one of them depending on \( k_i \).
Fast, compact, simple, safer Diffie-Hellman

- Write $k = \sum_{i=0}^{\ell-1} k_i 2^i$ with $k_{\ell-1} = 1$ and $P = (x_P, y_P)$ in $E[n]$ (e.g., on Curve25519 or Goldilocks)

\[
\begin{align*}
(x_0, x_1) &\leftarrow (\text{xDBL}(x_P), x_P) \\
\text{for } i = \ell - 2 \text{ downto } 0 &\text{ do} \\
(x_0, x_1) &\leftarrow \text{cSWAP}((k_{i+1} \otimes k_i), (x_0, x_1)) \\
(x_0, x_1) &\leftarrow (\text{xDBL}(x_0), \text{xADD}(x_0, x_1, x_P)) \\
\text{end for} \\
(x_0, x_1) &\leftarrow \text{cSWAP}(k_0, (x_0, x_1)) \\
\text{return } x_0 &= x_{[k]P}
\end{align*}
\]

- $x$-only Diffie-Hellman (Miller’85): $x([ab]P) = x([a][b]P) = x([b][a]P)$

see https://tools.ietf.org/html/rfc7748 (Elliptic curves for security)
Curve25519 and Goldilocks in the real world

- Both curves integrated into TLS ciphersuites
- In 2014, OpenSSH defaults to Curve25519
ECC is the best of both worlds

attacker’s toolbox vs. our toolbox
Elliptic curves: the best of both worlds

attacker: generic vs. us: not generic

\[
\begin{align*}
\mathbb{P}^1 \times \mathbb{P}^1 \quad \mathcal{L} \in \mathbb{R}^n \\
p := 2^{127} - 1 \\
\text{Hom}(\mathcal{E}, \hat{\mathcal{E}}) \\
\phi \\
\pi_p : \mathcal{E}_W^\sigma \to \mathcal{E}_W \\
Q(\sqrt{D}) \\
(m, 0, 0, 0) + \mathcal{L}
\end{align*}
\]
One curve to rule them all...

\[ p := 2^{127} - 1 \]

\[ \mathcal{E}/\mathbb{F}_{p^2} : -x^2 + y^2 = 1 + dx^2y^2 \]

\[ d := 125317048443780598345676279555970305165 \cdot i + 4205857648805777768770. \]

- Group order is \( 2^3 \cdot 7^2 \cdot N \), where \( N \) is a 246-bit prime!

- Fastest formulas \([HCWD08]\) “complete” \((x_1, y_1) + (x_2, y_2) = \left( \frac{x_1y_1 + x_2y_2}{y_1y_2 - x_1x_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2} \right)\)

- Degree-2 \( \mathbb{Q} \)-curve, meaning degree \( 2p \) endomorphism \( \psi \)

- CM by ring of integers in \( \mathbb{Q}(\sqrt{-40}) \), meaning degree 5 endomorphism \( \phi \)
What’s an endomorphism?

- An endomorphism is a homomorphism from the curve to itself
  \[ \phi : E \rightarrow E \]

- For our (crypto) purposes, an efficiently computable endomorphism is like a cheap teleport/shortcut to a fixed scalar multiple
  \[ \phi(P) = \lambda[P] \]

- Easy example on the Bitcoin curve
  \[ E/\mathbb{F}_p : y^2 = x^3 + 7 \]
  with \( p \equiv 1 \text{ mod } 3 \), since there exists \( \xi \in \mathbb{F}_p \) where \( \xi^3 = 1 \) and \( \xi \neq 1 \)

- For any \( P = (x, y) \in E \), \( \phi(P) = (\xi x, y) = [\lambda]P \), where

\[ \lambda = 37718080363155996902926221483475020450927657555482586988616620542887997980018 \]
How to use endomorphisms

- Recall our task: given integer $k$ and point $P$, compute $[k]P$
- For any $P$, we can now quickly get the three points $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$, where
  $\phi(P) = [\lambda_{\phi}]P,$
  $\psi(P) = [\lambda_{\psi}]P,$ and
  $\psi(\phi(P)) = [\lambda_{\phi} \lambda_{\psi}]P$

\[
k \mapsto (a_1, a_2, a_3, a_4)
\]

$$[k]P = [a_1]P + [a_2]\phi(P) + [a_3]\psi(P) + [a_4]\psi(\phi(P))$$

$$k \equiv a_1 + a_2 \lambda_{\phi} + a_3 \lambda_{\psi} + a_4 \lambda_{\phi} \lambda_{\psi} \mod N$$
The multiscalar multiplication

• Computed $\phi(P)$, $\psi(P)$, $\psi(\phi(P))$, and $k \mapsto (a_1, a_2, a_3, a_4)$, now what?

$$k = 64840569332679984426672436340494668739430332089137885001096300239355695153788$$

$$\begin{align*}
a_1 &= 14445124749170047041 \\
a_2 &= 11638376461179115075 \\
a_3 &= 5032911711680286358 \\
a_4 &= 881092582828842431
\end{align*}$$

• Instead of multiplying by a 246-bit scalar, do a 4-way multi-scalar exponentiation by 64-bit scalars

• 64-doublings, 64-additions, uniform dbl-and-always-add algorithm
**FourQ** versus Curve25519 and Curve p-256

<table>
<thead>
<tr>
<th>Platform</th>
<th>FourQ</th>
<th>Curve25519</th>
<th>NIST p-256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-Longa’15</td>
<td>Bernstein’06</td>
<td>NIST’99 [GK15]</td>
</tr>
<tr>
<td>Atom Pineview</td>
<td>442</td>
<td>1,109</td>
<td>-</td>
</tr>
<tr>
<td>Intel Sandy</td>
<td>72</td>
<td>157</td>
<td>400</td>
</tr>
<tr>
<td>Intel Haswell</td>
<td>56</td>
<td>162</td>
<td>312</td>
</tr>
<tr>
<td>AMD Kaveri</td>
<td>122</td>
<td>301</td>
<td>-</td>
</tr>
</tbody>
</table>

*Speed (in thousands of cycles) of \( k, P \mapsto [k]P \) on some 64-bit platforms.*

<table>
<thead>
<tr>
<th>Platform</th>
<th>FourQ</th>
<th>Curve25519</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-Longa’15 [Lon16]</td>
<td>Bernstein’06 [BS12,eBACS]</td>
</tr>
<tr>
<td>Cortex-A7</td>
<td>378</td>
<td>926</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>242</td>
<td>497</td>
</tr>
<tr>
<td>Cortex-A9</td>
<td>257</td>
<td>568</td>
</tr>
<tr>
<td>Cortex-A15</td>
<td>133</td>
<td>315</td>
</tr>
</tbody>
</table>

*Speed (in thousands of cycles) of \( k, P \mapsto [k]P \) on some 32-bit platforms.*
• Internet draft Curve4Q (by Barnes, Ladd, Longa)

• Fast SchorrQ signatures (based on EdDSA signature scheme)

• Library protected against simple timing attacks, cache attacks, exception attacks, invalid curve and small subgroup attacks

• Version 3.0 coming soon…
Questions?