Applied Cryptography
Spring Semester 2023
Lectures 29 and 30

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of this lecture

- Key Sizes
- Elliptic Curves
- Cryptography from Elliptic Curves
- ECIES and ECDSA
- ECC adoption

Key Sizes

Key sizes

- Cryptographic algorithms provide different strength levels depending on their underlying design and key size.
- Although it is not generally an easy task, it is possible to identify comparable strength levels between algorithms.
- The algorithms are considered to be comparable if the amount of work needed to break the algorithms or determine the keys, is approximately the same using a given resource.
- https://www.keylength.com/ gives a summary of different methodologies and results.

Key sizes

Security Level	RSA modulus size	Elliptic Curve group size	Discrete Logarithm field, subgroup size
80	1024	160	1024, 160
112	2048	224	2048, 224
128	3072	256	3072, 256
192	7680	384	7680, 384
256	15360	512	15360, 512

Motivation for Elliptic Curve Cryptography

- It is possible to define variants of the DL-based algorithms over *any* cyclic group.
- Relevant hard problems are then DLP, CDH, DDH etc, in the chosen group.
- In circa 1985, Koblitz and Miller independently proposed using cyclic groups coming from a class of mathematical objects called *elliptic curves.*
- Core idea: no sub-exponential algorithms are known for those groups in general.
- Only generic $O(n^{1/2})$ algorithms for DLP apply for elliptic curves in general (but there are some weak special cases to avoid).
- This allows smaller bit-sizes to be used, which leads to significant performance benefits.
- ECC only started to become widely used in mid 2010s – 25 years from invention to mass deployment.

Elliptic Curves

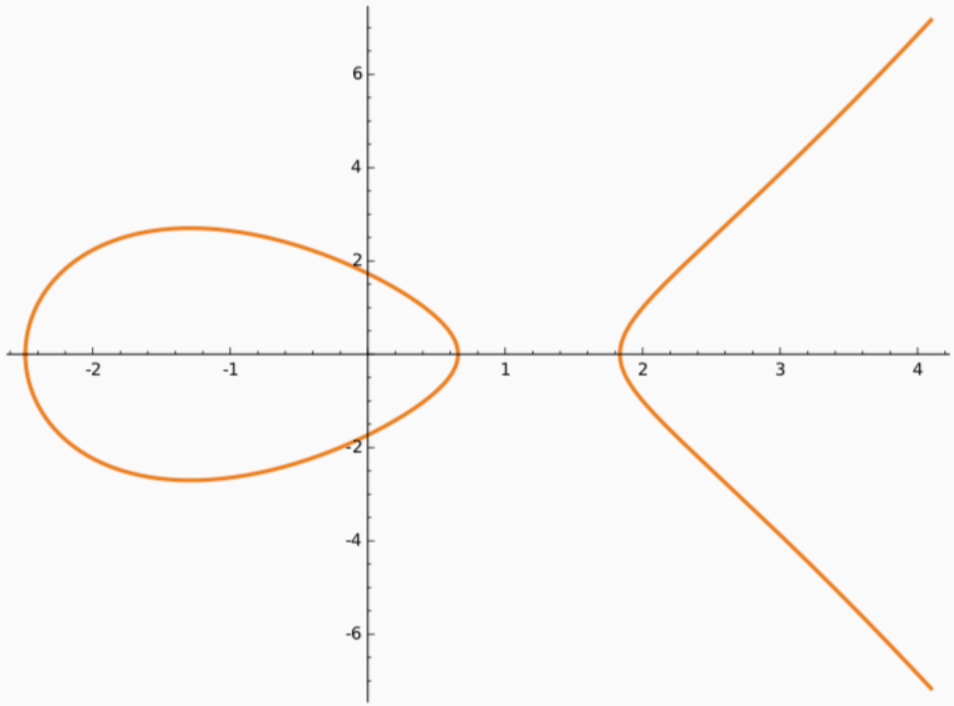
Elliptic Curves

- An elliptic curve over a field F is a set of pairs $(x,y) \in F \times F$, where F is some field.
- A common form for the equation of an elliptic curve is

$$E = \{ (x,y) \in F \times F \mid y^2 = x^3 + ax + b \} \cup \{ O \}$$

  where $4a^3 + 27b^2 \neq 0$ over F.
- This is called *short Weierstrass form using affine coordinates*; other common forms used in cryptography include: Montgomery form, Edwards form.
- Here a and b are coefficients from F that can vary to give us different curves.
- Here "point at infinity" O is a special curve point that does not have a representation as a pair of field elements.
- In applications, we usually work with one fixed, standardised curve whose properties have been carefully evaluated.

Elliptic Curve over the Rationals with a = −5, b = 3




```
sage: E = EllipticCurve([-5, 3])
```

$y^2 = x^3 + 2x + 4$ modulo 5

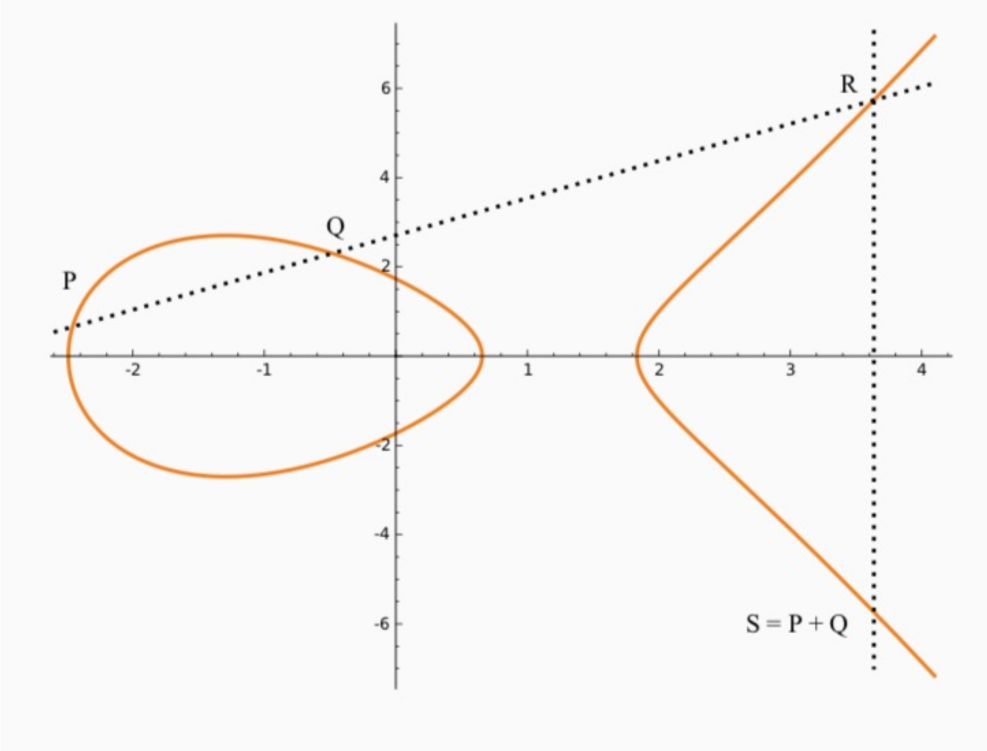
x	0	1	2	3	4
x^3	0	1	3	2	4
$2x$	0	2	4	1	3
4	4	4	4	4	4
y^2	4	2	1	2	1
y	2,3		1,4		1,4

- Here, we see fairly typical behaviour of elliptic curve over a finite field (using p=5).
- $x^3 + 2x + 4$ takes on 3 distinct values; of these 2 values have square roots mod 5, leading to points (0,2), (0,3), (2,1), (2,4), (4,1), (4,4).
- Including O, we get a total of 7 points on our curve E.

Addition of Points on an Elliptic Curve

- Any pair of points on an elliptic curve can be **added** to obtain a third point.
- The point at infinity O acts as an (additive) identity for this addition operation.
  - $P + O = O + P = P$ for all elliptic curve points P.
- Each point P has an (additive) inverse denoted $-P$.
  - If $P = (x, y)$ then $-P = (x, -y)$.
  - $P + (-P) = O$.
  - Point at infinity O is its own inverse: $O + O = O$.

Addition of Points on an Elliptic Curve



- There is a geometric interpretation of the addition process: to find *P* + *Q*, draw a straight line through *P* and *Q*, find the point of intersection with the curve, and project through the x-axis.

Addition of Points on an Elliptic Curve

- We provide explicit formulae for point addition (P+Q) and point doubling (P+P).
- These are based on the geometric interpretation from the previous slide.
- To **add** two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ with $x_1 \neq x_2$:
  1. $\lambda = (y_2 - y_1)/(x_2 - x_1)$
  2. $x_3 = \lambda^2 - x_1 - x_2$
  3. $y_3 = \lambda(x_1 - x_3) - y_1$
  4. return (x_3, y_3)
- To **double** a point $P = (x, y)$, i.e. to compute $P + P$:
  1. $\lambda = (3x^2 + a)/(2y)$
  2. $x' = \lambda^2 - 2x$
  3. $y' = \lambda(x - x') - y$
  4. return (x', y')

Elliptic Curves as Groups

- This addition law turns the set of points on an elliptic curve over a field into a **group**.
- (This demands a proof, particularly for the associative law, namely $(P+Q)+R = P+(Q+R)$ for any three points P, Q, R, but we do not provide one here.)
- The group operation is written as "+", and we speak of **adding** two points (our usual notation up to this point has been multiplicative).
- The identity in the group is the special point O (the point at infinity).
- The group **order** is the number of points on the curve.
- By carefully choosing E, we can ensure that the group has prime or nearly prime order.
- We work in a prime-order subgroup because this maximises security against generic algorithms.

Example: Elliptic Curves as Groups

- Recall the curve E with equation $y^2 = x^3 + 2x + 4$ over $F = F_5$.
- This curve has points O, (0,2), (0,3), (2,1), (2,4), (4,1), (4,4).
- So the group order is 7, a prime.
- Take $P = (0,2)$.
- Then P, $P+P$, $P+P+P$,… gives all 7 group elements.
- So P is a **generator** of the group of points on E.
- Compare with $\{1, g, g^2, \ldots g^{q-1}\}$ in the usual discrete logarithm setting (where we have a subgroup of order q mod p).

Scalar Multiplication

- We write $[k]P$ for the operation of adding P to itself k times.
- This is called *scalar multiplication by k*.
- This is the analogue of exponentiation in the usual discrete log setting:

$$[k]P \text{ on } E \quad \leftrightarrow \quad g^x \bmod p$$

- In our example, P, $[2]P$, $[3]P$,… gives us the full set of points on the curve.
- NB1: $[7]P = O$ in our example.
- NB2: If $P = (x,y)$, then $[k]P \neq (kx, ky)$ in general!

Scalar Multiplication

- To compute some scalar multiple of a point P we use an analogue of square-and-multiply from the multiplicative setting called *double-and-add.*
- **Example**: Suppose we want to compute $[9]P$.
- In binary 9 = 1001, so we compute $[9]P$ by the following chain:

1:	$O \longrightarrow [2]0+P = P$	(double and add)
0:	$P \longrightarrow [2]P$	(double)
0:	$[2]P \longrightarrow [4]P$	(double)
1:	$[4]P \longrightarrow ([4]P+[4]P) + P = [9]P$	(double and add)

- In general, if the scalar k has t bits, then scalar multiplication of P by k can be accomplished in at most t doublings and t additions of points.
- There is a vast literature involved in making $[k]P$ go fast and be resistant to side-channel attacks.

Cryptography from Elliptic Curves

The Elliptic Curve Discrete Logarithm Problem

Recall the (classical) discrete logarithm problem:

The Discrete Logarithm Problem in G_q:

Let (p, q, g) be group parameters (so q divides p-1; g has order q mod p).

Set $y = g^x \bmod p$, where x is a uniformly random value in {0,1,…,q-1}.

Given (p, q, g) and y, find x.

The Elliptic Curve Discrete Logarithm Problem (ECDLP):

Let E be an elliptic curve over the field F of prime order p.

Let P be a point of prime order q on E.

Set $Q = [x]P$ where x is a uniformly random value in {0,1,…,q-1}.

Given E and points P, Q, find x.

The Elliptic Curve Discrete Logarithm Problem

- The essence of Elliptic Curve Cryptography is that, except for some special cases, the best algorithms for solving ECDLP run in time $O(q^{1/2})$ where q is the prime order of the generator P.
- These are in fact *generic* algorithms that work in any cyclic group.
  - Baby-steps-Giant-Steps, Pollard lambda algorithm, Pollard rho algorithm, Method of Wild and Tame Kangaroos,…
  - These all require running time (and, in some cases, space) that are **exponential** in $\log_2 q$, the bit-size of q.
- This enables us to choose much smaller parameters than are needed in "mod p" discrete-log-based cryptography.
- This results in more compact keys, ciphertexts, etc, and faster cryptographic operations.

Cryptography from ECDLP

- Most schemes for the DLP setting can be translated easily into the ECDLP setting.
- ECIES and ECDSA are translations of DHIES and DSA.
- Example: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral).
  - Alice and Bob agree on a curve E and a base-point P of prime order q.
  - Alice chooses x uniformly at random from $\{0,1,..,q-1\}$, and sends Bob $[x]$P.
  - Bob chooses y uniformly at random from $\{0,1,..,q-1\}$, and sends Alice $[y]$P.
  - Both sides can now compute $[xy]P$: Alice computes $[x]([y]P)$ and Bob computes $[y]([x]P)$.
  - Security?

ECC Setup

To set up a system for using elliptic curve cryptography:

- We need to decide on a field F (usually a prime field for some prime p).
- We need to decide on a curve E over that field.
- We need to find a base point P on the curve of known and large prime order q.
- We need to support the new arithmetic of scalar multiplication on our curve, in a fast and secure manner.

Given the additional complexity of the new operations, there is lots of scope for errors and new attack vectors!

- Example: basic doubling and adding operations use different formulae, leading to timing side channels.
- Example: computing $[k]P$ may be faster if MSBs of k are zero, again resulting in timing side channels (and possible leak of ECDSA private key).

Curve Selection

- For the field F of prime order p, a curve E over F has n points where:

$$p + 1 - 2\sqrt{p} \leq n \leq p + 1 + 2\sqrt{p}$$

- This is known as the **Hasse-Weil bound.**
- For large p, it means that the bit-size of n is the same as that of p.
- Prime order curves (where $n=q$ is prime) are popular and enjoy some implementation advantages.
- Otherwise, we typically ensure $n = h \cdot q$ where h (called the co-factor) is small and q is prime.
- The Schoof-Elkies-Adkin (SEA) algorithm can be used to compute the number of points on an elliptic curve in a fairly efficient manner.
- Easier and better to rely on curves that are *standardised* by trusted sources.

An example standardised curve: NIST P-256

- $p = 2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1$.
- $a = -3$
- b := 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b.
- $h = 1$; q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
- A base point is also specified.
- NIST P-256 is a curve of prime order q; special *sparse form* of p can make mod p arithmetic faster.
- Very widely supported in crypto libraries.
- p and q have 256 bits, so complexity of solving ECDLP is about 2^{128}.

An example standardised curve: Curve25519

- Introduced by Bernstein in 2005/2006.
- $p = 2^{255} - 19$, allowing very fast modular reduction mod p.
- Curve equation: $y^2 = x^3+486662x^2+x$.
- Not in reduced Weierstrass form, but instead Montgomery form, allowing ECDH operations to be done using only x coordinates in a side-channel resistant manner.
- Group order: $8(2^{252}+ 27742317777372353535851937790883648493)$.
- Co-factor of 8 has caused problems in various implementations/applications.
- "Minimal" curve satisfying various security/performance criteria.
- Offers a bit less than 128-bit security, improved speed compared to, e.g. NIST P-256.
- Adopted for use in TLS 1.3 (along with NIST P-256, NIST P-384, NIST P-521 and Curve448-Goldilocks).
- See https://cr.yp.to/ecdh/curve25519-20060209.pdf and RFC 7748 for further details.

Base Point Selection

- Suppose E defined over F has n points where n has a large prime divisor q.
- Choose a non-O point P so that P has order q, i.e. check that $[q]P = O$.
- If $n = q$, then every point P on the curve will have this property; otherwise take a random point P' and compute $[h]P'$ and check $[h]P' \neq O$.
- How to find a random point on the curve?
  - Pick a random x, compute $x^3 + ax + b$, and try to solve for y in:

    $$y^2 = x^3 + ax + b \bmod p.$$

  - Requires an algorithm for taking square roots mod p – use Tonelli-Shanks.
  - This will succeed roughly half the time (half of the non-zero elements mod p are squares).
- Standardised curves normally come with specified base points.

Point Compression

- The point P can be represented by a pair (x, y) in $F \times F$.
- Then 2 field elements are needed to represent a point, requiring $2\log_2 p$ bits.
- This can be reduced to $1+\log_2 p$ bits using **point compression**.
  - Use $\log_2 p$ bits to define the x-coordinate, and 1-bit to represent the "sign" of y.
  - Can always extract two candidates (x, y) and $(x, p-y)$ for the point given x, by solving $y^2 = x^3 + ax + b \bmod p$.
  - Use the "sign" bit to decide between the two.

Key Pair Generation

- Suppose E defined over F has n points where n has a large prime divisor q; let P be a point of order q.
- To generate key pair for ECC:
  - Choose a random scalar k in {0,1,....,q-1}.
  - Set $Q = [k]P$.
  - The private key is k; the public key is Q.
- The problem of extracting the private key from the public key is the ECDLP.
- We've already seen how to use this set up to do an elliptic-curve analogue of ephemeral Diffie-Hellman (ECDHE).

ECIES and ECDSA

ECIES

- ECIES is a direct translation of DHIES to the ECC setting.
- Recall that DHIES involves a public key $X = g^x \bmod p$ and private key x.
- ECIES uses public key $X = [x]P$ and private key x where P is a base point of order q on some curve E.

ECIES

Global parameters:

E, p, q, P a point of order q on E.

KeyGen:

- Pick x uniformly at random from {0,1,…,q-1}.
- Return $pk = X = [x]P$, $sk = x$

Enc(pk=X, m):

1. Select r uniformly at random from {0,1,…,q-1}.
2. Set $Y = [r]P$, $Z = [r]X$, $K = H(Z,X,Y)$.
3. Split K into K_e and K_m.
4. Compute $C' =$ EtM(M) using keys K_e and K_m for encryption and MAC, respectively.
5. Output the ciphertext $C = (Y, C')$.

ECIES

<u>Dec(sk=x,(Y,C')):</u>

1. Check that Y is on E and has order q, return "fail" if not.
2. Compute $Z = [x]Y$.
3. Set $K = H(Z,X,Y)$.
4. Split K into K_e and K_m.
5. Decrypt C' using keys K_e and K_m for encryption and MAC, respectively.
6. Output "fail" if step 5 fails, otherwise output the message returned in step 5.

- So ECIES replaces the "DH" in DHIES with an "ECDH".
- Ciphertext overhead is one elliptic curve point plus MAC tag, roughly 256+128 bits at the 128-bit security level.
- Encryption dominated by cost of 2 scalar multiplications ($Y = [r]P$, $Z = [r]X$); decryption dominated by cost of 1 scalar multiplication ($Z = [x]Y$).

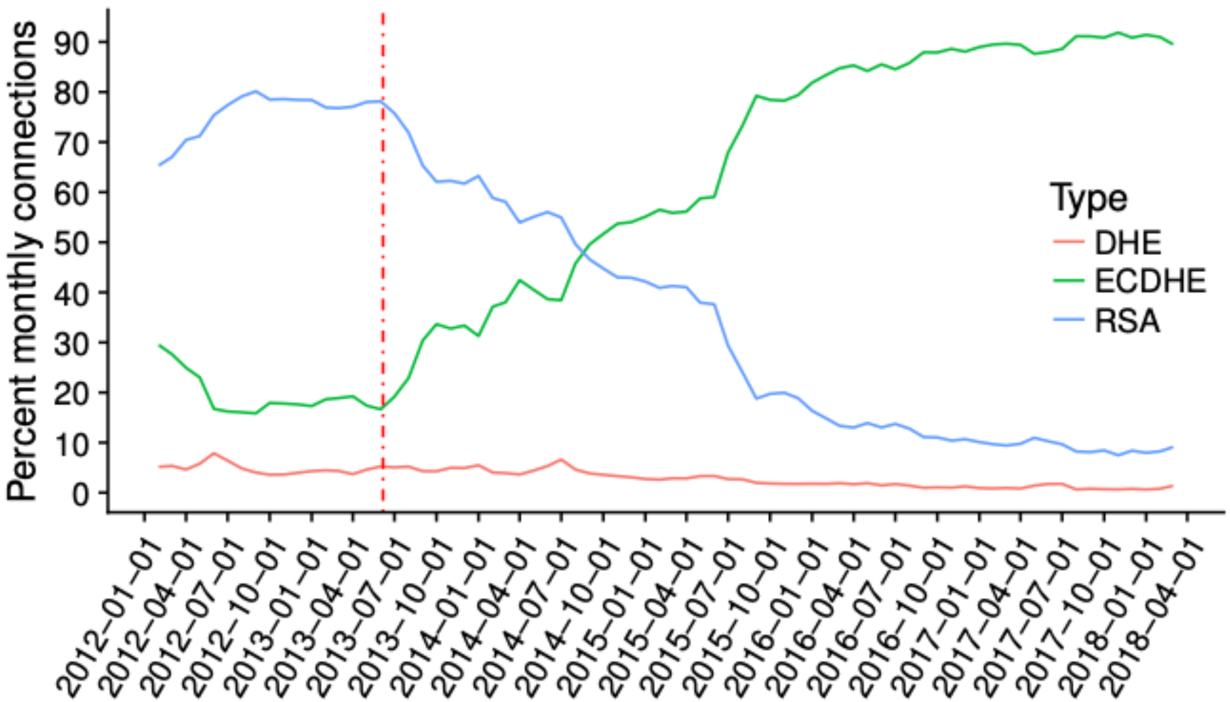
ECDSA

- ECDSA is a translation of DSA to the elliptic curve setting.
- Specified in: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf and ANSI X9.62
- Signatures are pairs (r,s) where r and s are integers mod q, the order of base point P; hence 512 bits at the 128-bit security level.
- See Boneh-Shoup, Section 19.3 for details.
- ECDSA has same reliance as DSA on per-signature nonce, with fatal loss of security if same nonce is used twice.
  - Also vulnerable to attacks based on partial knowledge of nonces.
- ECDSA has an unfortunate malleability property: if (r,s) is a valid signature for message m and verification key vk, then so is $(r,-s)$.
  - Hence not SUF-CMA secure; plausibly UF-CMA secure in generic group model (see: D. R. Brown. Generic groups, collision resistance, and ECDSA. Designs, Codes and Cryptography, 35(1):119–152, 2005).

The slow take-up of ECC

- ECC was "invented" in mid 1980s by Koblitz and Miller but ECC only became widespread in mid 2010s.
- Reasons for slow adoption include:
  - Mathematical and implementation complexity compared to RSA.
  - Uncertainty over security (RSA vs Certicom in early 1990s).
  - Lack of mature standards.
  - Unclear patent situation (Certicom, NSA Suite B).
  - Hard to displace existing widely-deployed technology (RSA-based crypto).
- Drivers for adoption:
  - Better performance than RSA encryption for key establishment in TLS and other protocols.
  - Using ECDHE in TLS enables forward security combatting passive, massive surveillance.
  - Patent situation clarified with expiry of key patents and licensing deal with US gov.
- Mass-scale adoption also in Bitcoin, Ethereum and other crypto currencies.

The slow take-up of ECDHE in TLS



From: Kotzias et al.: Coming of Age: A Longitudinal Study of TLS Deployment, IMC 2018.
https://www.icir.org/johanna/papers/imc18tlsdeployment.pdf

Homework

- Read Chapter 15.1-15.3 of Boneh Shoup.
- Prep for lab this week
- Coming up: lectures on Key Management, Authentication and Key Exchange, TLS, Signal, Telegram, Threema.

ECDSA – Details

ECDSA – The Gory Details

Parameters: (E, p, n, q, h, P, H) defining a curve E over field F_p with $n = q \cdot h$ points, subgroup of prime order q and generator P of order q; H is a hash function, e.g. SHA-256 (here we assume output of H is at least bit-size of q).

<u>KeyGen</u>:

Set $Q = [x]P$ where x is uniformly random from $\{1,\ldots,q\text{-}1\}$.

Output verification key: $\boldsymbol{Q}$; signing key: $\boldsymbol{x}$.

<u>Sign</u>: Inputs (x, m) // x is private key; m is the message to be signed

h = bits2int(H(m)) mod q. // take len(q) MSBs of H(m), cast to BigInt, reduce mod q.

Do:

1. Select k uniformly at random from $\{1,\ldots,q\text{-}1\}$.
2. Compute r = x-coord($[k]P$) mod q. // $[k]P$ is a point on E; its x-coord is in F_p; we consider that as an integer and reduce mod q.
3. Compute $s = k^{-1}(h + xr) \bmod q$.

Until $r \neq 0$ and $s \neq 0$. // works first try w.h.p.

Output (r,s).

ECDSA – The Gory Details

Verify: Inputs (Q, m, (r,s))   // Q is verification key; m is message; (r, s) is claimed signature.

1. check that $1 \le r \le q\text{-}1$ and $1 \le s \le q\text{-}1$.
2. compute $w = s^{-1} \bmod q$.
3. compute h = bits2int(H(m)) mod q.
4. compute $u_1 = w \cdot h \bmod q$ and $u_2 = w \cdot r \bmod q$.
5. compute $Z = [u_1]P + [u_2]Q$.
6. If (x-coord(Z) mod q == r) then output 1 else output 0.

Correctness:

Suppose (r,s) is a signature for message m under key Q. Then:

$$Z = [u_1]P + [u_2]Q = [s^{-1}h]P + [s^{-1}r]Q = [s^{-1}(h + xr)]P = [k]P.$$

Here we used $s = k^{-1}(h + xr) \bmod q$ from the signing algorithm to obtain $s^{-1}(h + xr) = k \bmod$ q.

Recalling that r = x-coord($[k]P$) mod q completes the argument.

ECDSA Security and Implementation Pitfalls

- Implementation requires:
  - Various fiddly conversions of bit-strings to integers, etc: bits2int() and conversion of mod p integers to mod q integers.
  - Uniform sampling of integers k in the range $\{1,\ldots, q\text{-}1\}$ – use rejection sampling (sample from $[0,2^t]$ for t = bitsize(q), until result is in $\{1,\ldots, q\text{-}1\}$).
  - Computation of multiplicative inverses mod q: k^{-1}, s^{-1}.
  - Scalar multiplications: $Q = [x]P$; $[k]P$; $[u_1]P + [u_2]Q$.
  - Sanity checks on r, s.
- There are lots of ways to get some or all of this wrong!
  - Sampling k wrongly, e.g. choose k from $[0,2^t]$ where t is bit-size of q, and reduce mod q.
  - Repeating k, or k being predictable due to bad RNG.
  - Leaking some or all of k through a side-channel attack, e.g. running time of $[k]P$ or computation of k^{-1} mod q.