Elliptic Curve Cryptography, Part 2

Cryptography, Spring 2020

L. Batina, J. Daemen

May 20, 2020

Institute for Computing and Information Sciences
Radboud University

Outline

ECC domain parameters

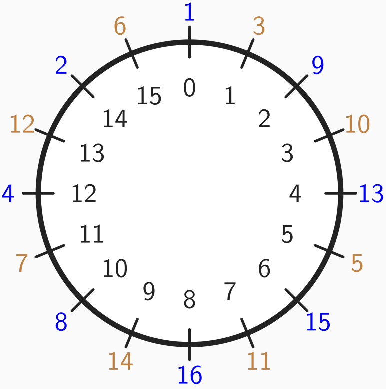
The order of the elliptic curve group

- To have n bits of security for DL it would be sufficient that:
  (1) $q = \text{ord}(G) \geq 2^{2n}$ and q prime
  (2) $\mathcal{E}$ is chosen so that it *avoids some properties*
- Due to Lagrange: $\text{ord}(G) \mid \#\mathcal{E}$
- So we need $\mathcal{E}$ with an order that is divisible by a prime $\geq 2^{2n}$
- What can we expect for $\#\mathcal{E}$?
  - For roughly half of the values $x \in \mathbb{F}_p$, the expression $x^3 + ax + b$ is a *quadratic residue*
  - If so and if y is a solution, so is $-y$
  - so $\#\mathcal{E}(\mathbb{F}_p) \approx \frac{1}{2} \cdot 2 \cdot p + 1 = p + 1$

Theorem of Hasse (Helmut Hasse, 1922)

For an elliptic curve over $\mathbb{F}_p$: $\#\mathcal{E} = p + 1 + t$ with $-2\sqrt{p} \leq t \leq 2\sqrt{p}$

Intermezzo: quadratic residues and non-residues in $(\mathbb{Z}/17\mathbb{Z})^*$



$X \in (\mathbb{Z}/17\mathbb{Z})^*$ is a quadratic residue if there exists a Y with $Y^2 = X$

If $X = 3^x$ with x even, then $Y^2 = X$ with $Y = 3^{x/2}$ and $Y = 3^{(16+x)/2}$

If $X = 3^x$ with x odd, no such Y exists

Computing the order of an elliptic curve group

Theorem of Hasse (Helmut Hasse, 1922)

For an elliptic curve over $\mathbb{F}_p$: $\#\mathcal{E} = p + 1 + t$ with $-2\sqrt{p} \leq t \leq 2\sqrt{p}$

- Curves exist with $\#\mathcal{E} = p$ or $\#\mathcal{E} = p + 1$, but on these DL is easy
- On *standard curves*, $\#\mathcal{E}$ and p are both *close* and *distant*
- Consider $p \approx 2^{256}$
  - distant (in absolute sense): $|\#\mathcal{E} - p|$ is an integer of ≈ 128 bits
  - close (relatively): 256-bit $\#\mathcal{E}$ and p differ only in last 128 bits

How to find out the order of an elliptic curve group?

Schoof's point counting algorithm

In 1985 René Schoof found an algorithm that made it feasible to determine $\#\mathcal{E}$, later improved by Noam Elkies and A. O. L. Atkin

ECC domain parameters

- We want $\mathcal{E}$ with $\#\mathcal{E} = hq$ with q a large prime and $h \leq 10$ or so
- Technique: repeat following until a suitable curve is found
  - take parameters p, a, b that would give a *good* curve
  - compute $\#\mathcal{E}$ with Schoof's algorithm
- To assure backdoor absence, choice of p, a, b should be *explainable*
- Curves are proposed by experts and standardization bodies

ECC domain parameters

- The prime p (in general, a prime power p^n including $p = 2$)
- The curve parameters a and b (may have a different shape)
- The generator G
- The order of the generator q
- The co-factor: $h = \#\mathcal{E}/q$

Standard elliptic curves

- 2000: First ECC domain parameters by company Certicom
- 2004: NIST standardized these and added some more
  - range of target security strength matching key lengths of AES
  - p are *Pseudo-Mersenne* primes for efficient modular reduction
    - $p256 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
    - $p384 = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
    - $p521 = 2^{521} - 1$ (an actual Mersenne prime!)
  - all have cofactor $h = 1$, gives certain advantages
  - all have $a = -3$, allowing optimizations
  - innuendo of NSA backdoor, up to now without proof

  look it up: https://csrc.nist.gov/publications/detail/sp/800-186/draft
- 2005: German *Brainpool* consortium proposed alternative curves
- 2010: China publishes its own curves
- 2011: la France présente sa propre courbe: *die Französische Kurve*
- etc.

ECC 2.0 curves

Efficient curves based on new insights and advanced math, best known:

- 2005: Curve25519 by Dan Bernstein, 126-bit security
- 2015: Curve448-Goldilocks by Mike Hamburg, 224-bit security
- 2015: FourQ by Craig Costello/Patrick Longa, 123-bit security

Their introduction was followed by a fierce battle for adoption

- Technical merit plays a role for adoption but other aspects too
- Lobbying in standardization groups and development community
  - ISO, NIST, BSI, ...
  - Internet standard governing TLS 1.3, SSH, ...: CFRG
  - OpenSSL, OpenVPN, etc.
  - Signal, WhatsApp, ...

Scalar multiplication

Efficient scalar multiplication

- Scalar multiplication is the ECC counterpart of exponentiation
- Computing $[a]G$ in naive way takes $a-1$ point additions
- Infeasible if a and the coordinates of G are hundreds of bits long
- ECC counterpart of square-and-multiply is double-and-add
- Example: $[43]G$ with $G=(5,1)\in\mathcal{E}(\mathbb{F}_{23}): y^2=x^3-x-4$

10	$[2]G$	$=$	$G+G$	$[2]G=$	$(2,18)$	$=$	$(5,1)+(5,1)$
100	$[4]G$	$=$	$[2]G+[2]G$	$[4]G=$	$(14,9)$	$=$	$(2,18)+(2,18)$
1000	$[8]G$	$=$	$[4]G+[4]G$	$[8]G=$	$(21,17)$	$=$	$(14,9)+(14,9)$
10000	$[16]G$	$=$	$[8]G+[8]G$	$[16]G=$	$(5,22)$	$=$	$(21,17)+(21,17)$
100000	$[32]G$	$=$	$[16]G+[16]G$	$[32]G=$	$(2,5)$	$=$	$(5,22)+(5,22)$

working it out:

11	$[3]G$	$=$	$[2]G+G$	$[3]G=$	$(20,15)$	$=$	$(5,1)+(2,18)$
1011	$[11]G$	$=$	$[8]G+[3]G$	$[11]G=$	$(9,7)$	$=$	$(21,17)+(20,15)$
101011	$[43]G$	$=$	$[32]G+[11]G$	$[43]G=$	$(21,6)$	$=$	$(5,22)+(9,7)$

- Only 5 doublings and 3 additions instead of 42
- Side note: this example can be done in 3 doubling and 1 addition (find out why!)

Pseudocode for double-and-add, left-to-right variant

Input: point $G \in \mathcal{E}$, scalar $a \in \mathbb{Z}/q\mathbb{Z}$
Output: $A \in \mathcal{E}$ with $A = [a]G$

Let $a = a_0 + 2a_1 + 2^2a_2 + 2^3a_3 + \ldots + 2^{n-1}a_{n-1}$ and $\forall i : a_i \in \mathbb{Z}/2\mathbb{Z}$
$T \leftarrow G$
for $i \leftarrow n-2$ down to 0 **do**
  $T \leftarrow [2]T$
  if $a_i = 1$ **then** $T \leftarrow T + G$
end for
return $A \leftarrow T$

- there are many other algorithms for scalar multiplication
- for better efficiency, protection against side channel or fault injection attacks, . . .
- these are out of scope of this course, except *NAF*

Non-adjacent form (NAF)

- In $(\mathbb{Z}/p\mathbb{Z})^*$, $A/B = A \times B^{-1}$ is more expensive than $A \times B$
- In $\mathcal{E}$, $A - B = A + (-B)$ has same cost as $A + B$
- Take $[15]G$
  - classically $[15]G = G + [2]G + [4]G + [8]G$: 3 double, 3 add
  - alternative $[15]G = [16]G - G$: 4 double, 1 add
- Represent scalar with *signed bits*, e.g., $15 = (1, 0, 0, 0, -1)$
- Signed-digit representation $a = \sum_{i=0}^{l-1} a_i 2^i$ with $a_i \in \{-1, 0, 1\}$
- An integer has many signed-digit representations
- Non-adjacent form (NAF): adjacent digits are not both non-zero
  - NAF representation is unique
  - and has minimal density of all signed digit representations
  - average # of nonzero digits is 1/3
  - note: length can increase by 1

Pseudocode for NAF double-and-add, left-to-right variant

Input: point $G \in \mathcal{E}$, scalar $a \in \mathbb{Z}/q\mathbb{Z}$
Output: $A \in \mathcal{E}$ with $A = [a]G$

Build the NAF representation of a: $(a_0, a_1, a_2, \ldots, a_{n-1})$
$T \leftarrow G$
for $i \leftarrow n - 2$ down to 0 **do**
  $T \leftarrow [2]T$
  if $a_i = 1$ **then** $T \leftarrow T + G$
  if $a_i = -1$ **then** $T \leftarrow T - G$
end for
return $A \leftarrow T$

This requires n doublings and on average $n/3$ additions/subtractions.

Computing the NAF representation

A simple recipe (we denote -1 by $\bar{1}$)

(1) start from the binary representation

(2) while there are non-zero adjacent digits, repeat following

- replace sequences $0(1^n)$ with $n > 1$ by $10^{n-1}\bar{1}$
- replace sequences $1\bar{1}$ by 01
- replace sequences $\bar{1}1$ by $0\bar{1}$

Example: 5631620749

$101001111101010111011011010001101$
$10100{\color{red}11111}01010{\color{red}111}0{\color{red}11}0{\color{red}11}01000{\color{red}11}01$
$101010000\bar{1}0101100\bar{1}10\bar{1}10\bar{1}010010\bar{1}01$
$101010000\bar{1}010{\color{red}11}00{\color{red}\bar{1}1}0{\color{red}\bar{1}1}0\bar{1}010010\bar{1}01$
$101010000\bar{1}0110\bar{1}000\bar{1}00\bar{1}0\bar{1}010010\bar{1}01$
$101010000\bar{1}0{\color{red}11}0\bar{1}000\bar{1}00\bar{1}0\bar{1}010010\bar{1}01$
$101010000\bar{1}10\bar{1}0\bar{1}000\bar{1}00\bar{1}0\bar{1}010010\bar{1}01$
$101010000{\color{red}\bar{1}1}0\bar{1}0\bar{1}000\bar{1}00\bar{1}0\bar{1}010010\bar{1}01$
$1010100000\bar{1}0\bar{1}0\bar{1}000\bar{1}00\bar{1}0\bar{1}010010\bar{1}01$

This works, but it can actually be done in a single pass from right-to-left

Projective coordinates

Projective space

- $\mathcal{O}$: an element of $\mathcal{E}$ but **not** a solution of the Weierstrass equation
- ...that defines a subset of the affine plane: $\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p\}$

More *natural*: picture the elliptic curve in the *projective plane*: the affine plane extended with the "points at infinity"

The projective plane $\mathbb{P}^2$ over a field K

Set of equiv. classes of triplets (X, Y, Z) (all in K) excluding $(0, 0, 0)$
The equivalence relation is defined as
$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \Leftrightarrow \exists \lambda \in K : (X_1, Y_1, Z_1) = (\lambda X_2, \lambda Y_2, \lambda Z_2)$

- The points $(X, Y, 0)$ are the "points at infinity" or "line at infinity"
- The other points are called "finite points"
- We can now represent $\mathcal{O}$ in coordinates: $(0, 1, 0)$!

The elliptic curve equation in homogeneous coordinates

Each affine point (x, y) has projective counterpart (X, Y, Z) with

$$x = X \times Z^{-1} \text{ and } y = Y \times Z^{-1} \text{ with } Z \neq 0$$

The infinite projective points have no counterpart

Substitution in the Weierstrass equation $y^2 = x^3 + ax + b$ yields

$$Y^2 Z = X^3 + aXZ^2 + bZ^3$$

In this equation all terms have the same degree (3) $\Rightarrow$ *homogeneous*

Therefore these (X, Y, Z) are called homogeneous coordinates

Clearly $\mathcal{O} = (0, 1, 0)$ satisfies this equation and $-(X, Y, Z) = (X, -Y, Z)$

Computing with these, we can avoid multiplicative inverses! Intuition:

$$(X/R, 0, Z) = (X, 0, Z \times R)$$

Jacobian projective coordinates

There are other projective representations and most often used are *Jacobian* coordinates

Each affine point (x, y) has Jacobian counterpart (X, Y, Z) with

$$x = X \times Z^{-2} \text{ and } y = Y \times Z^{-3} \text{ with } Z \neq 0$$

The equivalence relation is defined as
$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \Leftrightarrow \exists \lambda \in K : (X_1, Y_1, Z_1) = (\lambda^2 X_2, \lambda^3 Y_2, \lambda Z_2)$

The Weierstrass equation $y^2 = x^3 + ax + b$ now becomes:

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

Filling in $Z = 0$ now yields $X^3 = Y^2$ and hence $\mathcal{O} = (1, 1, 0)$

As for homogeneous coordinates, we have $-(X, Y, Z) = (X, -Y, Z)$

Point addition and doubling in Jacobian projective coordinates

Consider three points P, Q, R and their Jacobian coordinates:

$$(X_p, Y_p, Z_p),\ (X_q, Y_q, Z_q) \text{ and } (X_r, Y_r, Z_r)$$

[exact formulas are for info only]

Computing $R = P + Q$

$$\begin{aligned}
r &= X_p Z_q^2, \\
s &= X_q Z_p^2, \\
t &= Y_p Z_q^3, \\
u &= Y_q Z_p^3, \\
v &= s - r, \\
w &= u - t, \\
X_r &= -v^3 - 2rv^2 + w^2, \\
Y_r &= -tv^3 + (rv^2 - X_r)w, \\
Z_r &= vZ_pZ_q.
\end{aligned}$$

Computing $R = 2P$

$$\begin{aligned}
v &= 4X_p Y_p^2 \\
w &= 3X_p^2 + aZ_p^4 \\
X_r &= -2v + w^2 \\
Y_r &= -8Y_p^4 + (v - X_r)w \\
Z_r &= 2Y_pZ_p
\end{aligned}$$

More operations than in affine form, but **no more inversions**

On the choice of representation

There is a wide variety of representations, **make sure to check**
https://hyperelliptic.org/EFD/

- hardness of ECDLP is independent of the representation
- Affine is most compact and hence used in communication
- Projective avoids inversions and hence used in computation
- Converting projective to affine requires inverting Z
- Best choice of type of projective coordinates depends on
  - protocol: key agreement, signature, encryption, ...
  - platform: CPU instruction set, co-processor presence, ASIC, ...
  - domain parameters: pseudo-Mersenne or not, value of a, ...
  - need for protection against side channel attacks, ...
  - this is a subject of cryptographic engineering

Elliptic Curve Cryptosystems

Elliptic Curve Cryptosystems

Key pair generation in ECC

$a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$
$A \leftarrow [a]G$

ECC variants of classical discrete log schemes:

- ECDH: shared secret is x-coordinate of point on curve
- EC ElGamal encryption: plaintext and ciphertext are points on curve
- EC Schnorr authentication
- EC Schnorr signature variants:
  - ECDSA
  - EdDSA

All we said about classical discrete-log schemes applies to EC variants too

But there are some specifics . . .

Elliptic Curve Diffie-Hellman (ECDH) key exchange

	Alice		Bob
have in advance:	$\mathcal{E}, G, (q), a, A$		$\mathcal{E}, G, (q), b, B$
		$\xrightarrow{\text{Alice}, A}$	
		$\xleftarrow{\text{Bob}, B}$	
	$P \leftarrow [a]B$		$P \leftarrow [b]A$

Alice and Bob arrive at the same shared secret point P

$$P = [a]B = [a][b]G = [ab]G = [b][a]G = [b]A$$

- As shared secret one takes the x-coordinate of the shared point P
- Does this reduce the security?
  - given $P \in \mathcal{E}$, x_p almost fully determines P
  - y_p has 2 possible values, so carries one more bit of information
- Alice and Bob derive key(s) from secret: $K \leftarrow \mathrm{H}(\text{“KDF”}; x_p)$

EC ElGamal encryption

Alice		Bob
$\mathcal{E}, G, (q), B$		$\mathcal{E}, G, (q), b, B(= [b]G)$
$a \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$		
$A \leftarrow [a]G$		
$C \leftarrow M + [a]B$	$\xrightarrow{\text{Alice},(C,A)}$	$M \leftarrow C - [b]A$

- Cryptogram consists of two points on the curve: 4 affine coordinates
- Reduce data overhead by using *compressed representation*:
  - x-coordinate and parity of y: $y \bmod 2$
  - requires reconstruction of y-coordinate by receiver
- Reconstruction: compute $x^3 + ax + b$ and take its square root
- Square root is non-trivial but feasible: [for info only]
  - if $p \bmod 4 = 3$, $\sqrt{x} = \pm x^{(p+1)/4}$
  - for $p \bmod 4 = 1$ it is more complicated

EC Schnorr authentication protocol

Alice		Bob
$\mathcal{E}, G, q, A, a$		$\mathcal{E}, G, q$ (Alice: A)
$v \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$		
$V \leftarrow [v]G$	$\xrightarrow{\text{Alice}, V}$	$c \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$
	$\xleftarrow{c}$	
$r \leftarrow v - ca$	$\xrightarrow{r}$	$V \stackrel{?}{=} [r]G + [c]A$

- Just a different cyclic group
- Commitment V is now much shorter
- ...and can be shortened more with compressed point representation

EC Digital Signature Algorithm (ECDSA)

- NIST standard FIPS 186 defined DSA
  - This standard is updated regularly
  - FIPS 186-2 (2000) refers to ECDSA in an ANSI standard
  - FIPS 186-3 (2009) specifies ECDSA
  - Currently: draft FIPS 186-5 under revision
- ECDSA is probably the most implemented DL signature algorithm

This thing looks like this [for information only]:

Alice		Bob
$\mathcal{E}, G, q, A, a$		$\mathcal{E}, G, q$ (Alice: A)
$v \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, $V \leftarrow [v]G$		
$c \leftarrow x_v \bmod q$		
$r \leftarrow v^{-1}(\mathrm{H}(m) + ca)$	$\xrightarrow{m,(r,c)}$	$w \leftarrow r^{-1}$
		$P \leftarrow [\mathrm{H}(m)w]G + [cw]A$
		$c \stackrel{?}{=} x_p \bmod q$

EdDSA: the return of Schnorr!

Dan Bernstein proposes EdDSA as ECDSA alternative in 2007

- Ed stands for Edwards curve but maybe also *deterministic*
- It derives ephemeral key v from message
  - for this the private key is extended with a secret k
  - this avoids weaknesses due to bad randomness
  - ... but introduces other potential vulnerabilities
- Ed25519: EdDSA using SHA-512 and Curve25519
- Ed448: EdDSA using SHAKE256 and Curve448 (much nicer!)

Specifications are a messy affair, but in our formalism it looks like this:

Alice		Bob
$\mathcal{E}, G, q, A, a, k$		$\mathcal{E}, G, q$ (Alice: A)
$v \leftarrow \mathrm{H}(k; m)$, $V \leftarrow [v]G$		
$c \leftarrow \mathrm{H}(\mathcal{E}; G; A; V; m)$		
$r \leftarrow v + ca$	$\xrightarrow{m,(r,V)}$	$c \leftarrow \mathrm{H}(\mathcal{E}; G; A; V; m)$
		$[r]G \stackrel{?}{=} V + [c]A$

Deployment of elliptic curve cryptography

ECC is probably the most widespread public key crypto, e.g.,

- handshake in TLS 1.3 (HTTPS)
- Secure Shell (SSH)
- key agreement in Signal, Whatsapp
- Software update signatures (Sony, . . .)
- Signatures in Bitcoin and other cryptocurrencies

For more examples, see

In search of CurveSwap: Measuring elliptic curve implementations in the wild, L. Valenta et al.
`https://eprint.iacr.org/2018/298.pdf`.

Conclusions

Conclusions

- ECC is probably the most widespread public key crypto
- Elliptic curves provide great groups for discrete log based crypto
  - key exchange, encryption, authentication and signatures
  - short public keys, signatures and shared secrets
  - there is a wide variety of curves and representations
- ECC is efficient
  - projective coordinates for efficient point addition and doubling
  - double-and-add and NAF for efficient scalar multiplication
  - point compression for very short public keys and signatures
- Elliptic curves support *pairings* that allow exotic functionality (out of scope of this introductory course)