# Elliptic-Curve Cryptography (ECC)

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Talk presented in the
*Second International Conference on Mathematics and Computing* (*ICMC 2015*)
*Haldia, 5–10 January, 2015*

# Elliptic Curves and Cryptography

- Koblitz (1987) and Miller (1985) first recommended the use of elliptic-curve groups (over finite fields) in cryptosystems.
- Use of supersingular curves discarded after the proposal of the Menezes–Okamoto–Vanstone (1993) or Frey–Rück (1994) attack.
- ECDSA was proposed by Johnson and Menezes (1999) and adopted as a digital signature standard.
- Use of pairing in new protocols
  - Sakai–Ohgishi–Kasahara two-party key agreement (2000)
  - Boneh–Franklin identity-based encryption (2001)
  - Joux three-party key agreement (2004)
  - Boneh–Lynn–Shacham short signature scheme (2004)
- Numerous other applications of pairing after this.
- Supersingular curves are frequently used in these pairing-based protocols.

# Organization of the Talk

- **Part 1:** Arithmetic of Elliptic Curves (over Finite Fields)
- **Part 2:** Classical Elliptic-Curve Cryptography
- **Part 3:** Efficient Implementation
- **Part 4:** Introduction to Pairing
- **Part 5:** Pairing-Based Cryptography
- **Part 6:** Sample Application—ECDSA Batch Verification

# PART 1

## *ARITHMETIC OF ELLIPTIC CURVES*

# Elliptic Curves

Let $K$ be a field.

An **elliptic curve** $E$ over $K$ is defined by the **Weierstrass equation**:

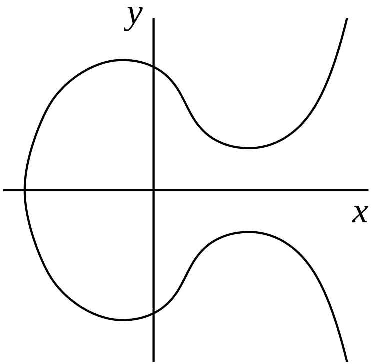
$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,\ a_i \in K.$$

The curve should be **smooth** (no singularities).

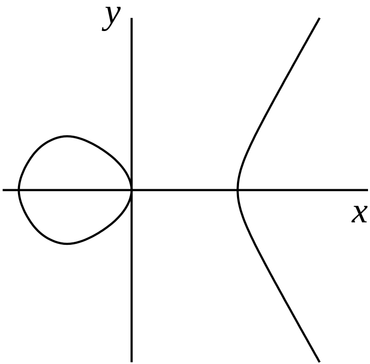
**Special forms**

- $\operatorname{char} K \neq 2, 3$: $y^2 = x^3 + ax + b,\ a, b \in K.$
- $\operatorname{char} K = 3$: $y^2 = x^3 + b_2x^2 + b_4x + b_6,\ b_i \in K.$
- $\operatorname{char} K = 2$:
  - **Non-supersingular or ordinary curve:** $y^2 + xy = x^3 + ax^2 + b,\ a, b \in K.$
  - **Supersingular curve:** $y^2 + ay = x^3 + bx + c,\ a, b, c \in K.$

# Real Elliptic Curves: Example




(a) $y^2 = x^3 - x + 1$




(b) $y^2 = x^3 - x$

# The Elliptic-Curve Group

Any $(x,y) \in K^2$ satisfying the equation of an elliptic curve $E$ is called a $K$-**rational point** on $E$.

**Point at infinity:**

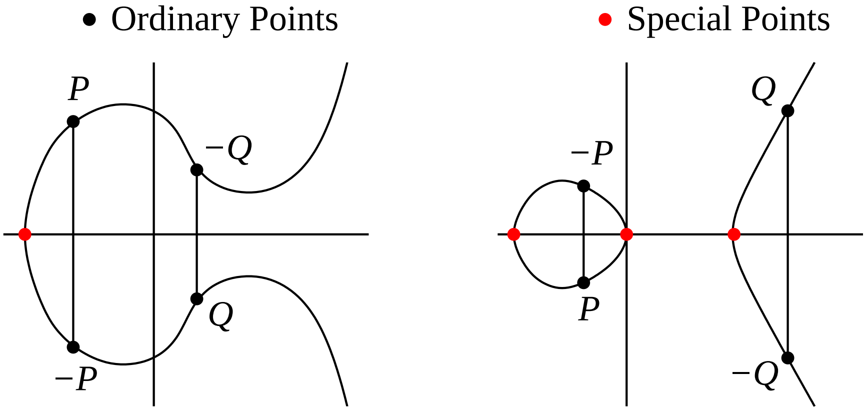
- There is a single point at infinity on $E$, denoted by $\mathcal{O}$.
- This point cannot be visualized in the two-dimensional $(x,y)$ plane.
- The point exists in the projective plane.

$E(K)$ is the set of all finite $K$-rational points on $E$ and the point at infinity.

An additive group structure can be defined on $E(K)$.

$\mathcal{O}$ acts as the identity of the group.

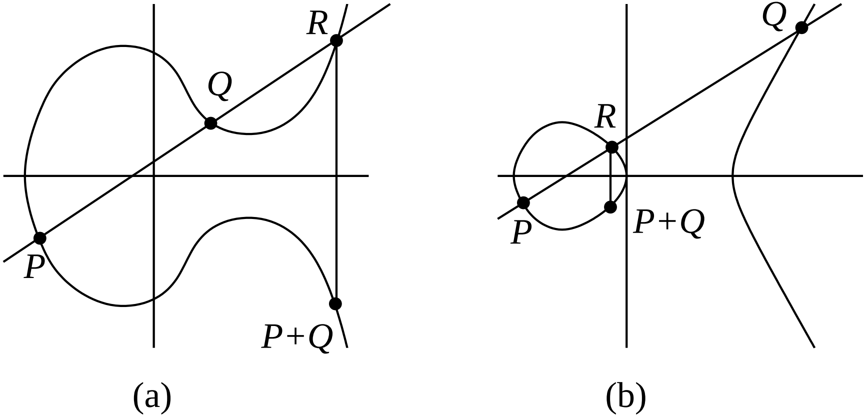
# The Opposite of a Point




(a) (b)

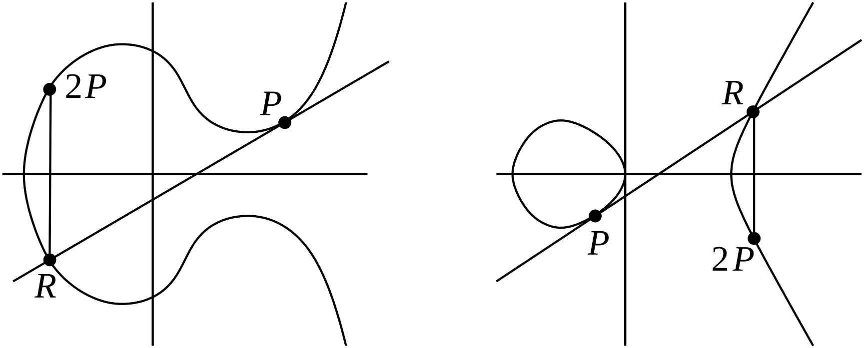
# Addition of Two Points

**Chord and tangent rule**



# Doubling of a Point

**Chord and tangent rule**




(a) (b)

# Addition and Doubling Formulas

Let $P = (h_1, k_1)$ and $Q = (h_2, k_2)$ be finite points.
Assume that $P + Q \neq \mathscr{O}$ and $2P \neq \mathscr{O}$.
Let $P + Q = (h_3, k_3)$ (Note that $P + Q = 2P$ if $P = Q$).

$E : y^2 = x^3 + ax + b$

$$\begin{aligned}
-P &= (h_1, -k_1) \\
h_3 &= \lambda^2 - h_1 - h_2 \\
k_3 &= \lambda(h_1 - h_3) - k_1, \text{ where} \\
\lambda &= \begin{cases} \dfrac{k_2 - k_1}{h_2 - h_1}, & \text{if } P \neq Q, \\[2ex] \dfrac{3h_1^2 + a}{2k_1}, & \text{if } P = Q. \end{cases}
\end{aligned}$$

# Addition and Doubling in Non-Supersingular or Ordinary Curves

$E : y^2 + xy = x^3 + ax^2 + b$ (with char $K = 2$).

$$
\begin{aligned}
-P &= (h_1, k_1 + h_1), \\
h_3 &= \begin{cases} \left(\dfrac{k_1 + k_2}{h_1 + h_2}\right)^2 + \dfrac{k_1 + k_2}{h_1 + h_2} + h_1 + h_2 + a, & \text{if } P \neq Q, \\[2ex] h_1^2 + \dfrac{b}{h_1^2}, & \text{if } P = Q, \end{cases} \\
k_3 &= \begin{cases} \left(\dfrac{k_1 + k_2}{h_1 + h_2}\right)(h_1 + h_3) + h_3 + k_1, & \text{if } P \neq Q, \\[2ex] h_1^2 + \left(h_1 + \dfrac{k_1}{h_1} + 1\right) h_3, & \text{if } P = Q. \end{cases}
\end{aligned}
$$

# Addition and Doubling in Supersingular Curves

$E: y^2 + ay = x^3 + bx + c$ (with char $K = 2$).

$$
\begin{aligned}
-P &= (h_1, k_1 + a), \\
h_3 &= \begin{cases} \left(\dfrac{k_1+k_2}{h_1+h_2}\right)^2 + h_1 + h_2, & \text{if } P \neq Q, \\[2ex] \dfrac{h_1^4 + b^2}{a^2}, & \text{if } P = Q, \end{cases} \\
k_3 &= \begin{cases} \left(\dfrac{k_1+k_2}{h_1+h_2}\right)(h_1 + h_3) + k_1 + a, & \text{if } P \neq Q, \\[2ex] \left(\dfrac{h_1^2 + b}{a}\right)(h_1 + h_3) + k_1 + a, & \text{if } P = Q. \end{cases}
\end{aligned}
$$

# Size of the Elliptic-Curve Group

Let $E$ be an elliptic curve defined over $\mathbb{F}_q = \mathbb{F}_{p^n}$.

- **Hasse's Theorem:**
  $|E(\mathbb{F}_q)| = q + 1 - t$, where $-2\sqrt{q} \leqslant t \leqslant 2\sqrt{q}$.
- $t$ is called the **trace of Frobenius** at $q$.
- If $t = 1$, then $E$ is called **anomalous**.
- If $p | t$, then $E$ is called **supersingular**.
- If $p \not| t$, then $E$ is called **non-supersingular** or **ordinary**.
- Let $\alpha, \beta \in \mathbb{C}$ satisfy $1 - tx + qx^2 = (1 - \alpha x)(1 - \beta x)$. Then,
  $|E(\mathbb{F}_{q^m})| = q^m + 1 - (\alpha^m + \beta^m)$.

**Note:** $E(\mathbb{F}_q)$ is not necessarily cyclic.

# Example of Elliptic-Curve Arithmetic

$E : y^2 = x^3 - 5x + 1$ defined over $\mathbb{F}_{17}$.
Take the finite points $P = (3, 8)$ and $Q = (10, 13)$ on $E$.

- **Opposite:** $-P = (3, 9)$, and $-Q = (10, 4)$.
- **Point addition**
  - The line $L$ joining $P$ and $Q$ has slope $\lambda \equiv \frac{13-8}{10-3} \equiv 8 \pmod{17}$.
  - $L$ has equation $L : y = 8x + c$. Since $L$ passes through $P$, we have $c = 1$.
  - Substitute this in the equation for $E$ to get $(8x+1)^2 \equiv x^3 - 5x + 1 \pmod{17}$, that is, $x^3 + 4x^2 + 13x \equiv 0 \pmod{17}$, that is, $x(x-3)(x-10) \equiv 0 \pmod{17}$.
  - The third point of intersection is $(0, 1)$, so $P + Q = -(0, 1) = (0, 16)$.
- **Point doubling**
  - The tangent $T$ to $E$ at $P$ has slope $\frac{3 \times 3^2 - 5}{2 \times 8} \equiv 12 \pmod{17}$.
  - The equation for $T$ is $y = 12x + 6$.
  - Substitute $T$ in $E$ to get $x^3 + 9x^2 + 4x + 16 \equiv 0 \pmod{17}$, that is, $(x-3)^2(x-2) \equiv 0 \pmod{17}$.
  - The third point of intersection is $(2, 13)$, so $2P = -(2, 13) = (2, 4)$.

# PART 2

## *CLASSICAL ELLIPTIC-CURVE CRYPTOGRAPHY*

# The Classical Intractable Problems

Let $G$ be a finite cyclic additive group with a generator $P$. Let $r = |G|$.

---

- **Discrete Logarithm Problem (DLP):** Given $Q \in G$, find $x$ such that $Q = xP$.
- **Diffie–Hellman Problem (DHP):** Given $aP, bP \in G$ (but not $a$ and $b$), compute $abP$.
- **Decisional Diffie–Hellman Problem (DDHP):** Given $aP, bP, zP \in G$ (but not $a$, $b$ and $z$), decide whether $zP = abP$, that is, whether $z \equiv ab \pmod{r}$.

---

- For elliptic-curve groups of suitable sizes, these problems are assumed to be intractable.
- We use the terms ECDLP and ECDHP to highlight the case of elliptic-curve groups.
- Elliptic-curve groups are not necessarily cyclic, so we usually work in sufficiently large cyclic subgroups with known generators.

# How Easy Is It to Solve ECDLP/ECDHP?

- ECDLP and ECDHP are believed to be equivalent.
- The DLP for finite fields can be solved by subexponential algorithms (like NFS and FFS).
- For general elliptic curves, subexponential algorithms are neither known nor likely to exist.
- Only the square-root methods work (Baby-Step-Giant-Step, Pollard rho and lambda, Pohlig–Hellman). For a group of size $n$, these methods run in $\tilde{O}(\sqrt{n})$ time.
- The ECDLP on a curve over $\mathbb{F}_q$ can be mapped to the finite-field DLP over $\mathbb{F}_{q^k}$ (MOV or FR reduction).
- In general, $k \approx n$. For supersingular curves, $k \in \{1,2,3,4,6\}$.
- For anomalous curves, a linear-time algorithm is known for the ECDLP.
- Supersingular and anomalous curves are not used in classical ECC.

# ElGamal Encryption

Let $G$ be an additive cyclic group of size $r$ and with a generator $P$.

- **Permanent key pair** (of Bob)
  - Private key: A random integer $d \in \{2, 3, \ldots, r-1\}$.
  - Private key: The group element $Y = dP$.
- **Encryption**
  - Alice wants to encrypt the message $M \in G$.
  - Alice generates a random session private key $d' \in \{2, 3, \ldots, r-1\}$.
  - Alice computes $S = d'P$ and $T = M + d'Y$ (where $Y$ is Bob's public key).
  - Alice sends $(S, T)$ to Bob.
- **Decryption**
  - Bob recovers $M = T - dS$ using his private key $d$.
  - Correctness: $dS = d'Y = dd'P$.
- **Security**
  - An eavesdropper knows $dP$ and $d'P$.
  - Computing the mask $dd'P$ is equivalent to solving an instance of the DHP in $G$.

# Elliptic Curve Digital Signature Algorithm (ECDSA)

Let $G$ be an additive cyclic group of size $r$ and with a generator $P$.

- **Key pair:** Private key $d \in \{2, 3, \ldots, r-1\}$, and public key $Y = dP$.
- **Signature generation**
  - Bob maps the message $M$ to a representative $m \in \{0, 1, 2, \ldots, r-1\}$.
  - Bob generates a random session key $d' \in \{2, 3, \ldots, r-1\}$.
  - Bob computes $S = d'P$, $s \equiv x(S) \pmod r$ and $t \equiv (m + ds)d'^{-1} \pmod r$.
  - Bob's signature on $M$ is the pair $(s, t)$.
- **Signature verification**
  - Compute $w \equiv t^{-1} \pmod r$, $u \equiv mw \pmod r$, and $v \equiv sw \pmod r$.
  - Compute $V = uP + vY \in G$ (here, $Y$ is Bob's public key).
  - Accept the signature if and only if $x(V) \equiv s \pmod r$.
- **Correctness**
  - $d' \equiv (m + ds)t^{-1} \equiv (mw + dsw) \equiv u_1 + u_2 d \pmod r$.
  - $S = d'P = uP + vdP = uP + vY$.

# PART 3

*EFFICIENT IMPLEMENTATION*

# What to Implement?

- A good finite-field library is the basic necessity. We assume that such a library is available.
- Elliptic-curve point addition and doubling are governed by fixed formulas.
- The most time-consuming operation in classical ECC is **elliptic-curve scalar multiplication**: Given an integer $n$ and an elliptic-curve point $P$, compute $nP$.
- It is easy to find the opposite of a point, so we assume $n > 0$.
- Scalar multiplication is the inverse of ECDLP (given $P$ and $nP$, compute $n$).
- Scalar multiplication behaves like a one-way function.
- A lot of optimization techniques apply to scalar-multiplication implementations.
- Here, we deal with software implementations only.

# Left-to-Right Scalar Multiplication

We are given a point $P$ on an elliptic curve $E$ defined over some $\mathbb{F}_q$.
We assume that the arithmetic functions of $\mathbb{F}_q$ are already available.
Let $r$ be the order of $P$.
Our task is to compute $nP$ for some integer $n \in \{1, 2, \ldots, r-1\}$.

---

- Let $n = (1n_{s-1}n_{s-2}\ldots n_1n_0)_2$ be the binary representation of $n$.
- Initialize $S = P$.
- For $i = s-1, s-2, \ldots, 1, 0$, repeat:
  - Set $S = 2S$. /* Doubling */
  - If $(n_i = 1)$, then set $S = S + P$. /* Conditional adding */
- Return $S$.

---

$s$ doubling operations.
At most $s$ addition operations. $s/2$ additions on an average.
$s \approx \log_2 n$.

# Left-to-Right Scalar Multiplication: Example

Consider the curve $E : y^2 = x^3 + 4x + 3$ modulo $p = 607$.
Take $P = (234, 121)$, and $n = 410 = (110011010)_2$.

- [Init] $S = P = (234, 121)$.
- $[i = 7]$ Dbl: $S := 2S = (65, 216)$, Add: $S := S + P = (2, 176)$.
- $[i = 6]$ Dbl: $S := 2S = (223, 283)$, Add: skipped.
- $[i = 5]$ Dbl: $S := 2S = (485, 464)$, Add: skipped.
- $[i = 4]$ Dbl: $S := 2S = (484, 76)$, Add: $S := S + P = (573, 25)$.
- $[i = 3]$ Dbl: $S := 2S = (31, 196)$, Add: $S := S + P = (403, 378)$.
- $[i = 2]$ Dbl: $S := 2S = (461, 250)$, Add: skipped.
- $[i = 1]$ Dbl: $S := 2S = (389, 228)$, Add: $S := S + P = (170, 25)$.
- $[i = 0]$ Dbl: $S := 2S = (541, 197)$, Add: skipped.

Therefore, $nP = (541, 197)$. Requires $8D + 4A$.

# Windowed Scalar Multiplication

Choose a small window size $w$.

---

- Precompute $aP$ for $a = 0, 1, 2, \ldots, 2^w - 1$.
- Let $n = (N_t N_{t-1} N_{t-2} \ldots N_1 N_0)_{2^w}$ be the $2^w$-ary representation of $n$.
- Initialize $S = N_t P$ (use the precomputed table).
- For $i = t-1, t-2, \ldots, 1, 0$, repeat:
  - For $j = 0, 1, 2, \ldots, w-1$, set $S = 2S$.
  - Set $S = S + N_i P$ (use the precomputed table).
- Return $S$.

---

$s$ doubling operations.
About $s/w$ additions at the cost of $2^w$ additions during precomputation.
Practical choice of window size: $w = 4$.

# Windowed Scalar Multiplication: Example

Consider the curve $E : y^2 = x^3 + 4x + 3$ modulo $p = 607$.
Take $P = (234, 121)$, $w = 3$, and $n = 410 = (110\ \ 011\ \ 010)_2 = (632)_8$.

- [Precomputation] $2P = (65, 216)$, $3P = (2, 176)$, $4P = (368, 523)$, $5P = (14, 539)$, $6P = (223, 283)$, and $7P = (96, 385)$.
- [Init] $S := 6P = (223, 283)$.
- [$i = 1$] Dbl: $S := 2S = (485, 464)$
  Dbl: $S := 2S = (484, 76)$
  Dbl: $S := 2S = (431, 45)$
  Add: $S := S + 3P = (403, 378)$
- [$i = 0$] Dbl: $S := 2S = (461, 250)$
  Dbl: $S := 2S = (389, 228)$
  Dbl: $S := 2S = (402, 361)$
  Add: $S := S + 2P = (541, 197)$

Requires $6D + 2A$ in the loop. Precomputation requires $1D + 5A$.
For large exponents, the precomputation overhead is insignificant.

# Windowed Method with Reduced Precomputation

- We represent $n = (N_t N_{t-1} N_{t-2} \ldots N_1 N_0)_{2^w}$ for a $w$-bit window.
- Precompute only the odd multiples $P, 3P, 5P, \ldots, (2^w - 1)P$.
- Express each $N_i = 2^{r_i} v_i$ with $v_i$ odd.
- Earlier, we had $w$ doubling operations followed by one addition.
- Now, we have:
  - $w - r_i$ doubling operations ($S := 2S$)
  - One addition ($S = S + v_i P$)
  - $r_i$ doubling operations ($S := 2S$)

The counts of doubling and addition operations do not change in the loop.
Precomputation effort is almost halved.

## Windowed Method: Example

Consider the curve $E : y^2 = x^3 + 4x + 3$ modulo $p = 607$.
Take $P = (234, 121)$, $w = 3$, and $n = 410 = (110\ \ 011\ \ 010)_2 = (632)_8$.

- [Precomputation] $2P = (65, 216)$, $3P = (2, 176)$, $5P = (14, 539)$, and $7P = (96, 385)$.
- [Init] $S = \mathcal{O}$.
- [$i = 2$] Dbl: $S := 2S = \mathcal{O}$
  Dbl: $S := 2S = \mathcal{O}$
  Add: $S := S + 3P = (2, 176)$
  Dbl: $S := 2S = (223, 283)$
- [$i = 1$] Dbl: $S := 2S = (485, 464)$
  Dbl: $S := 2S = (484, 76)$
  Dbl: $S := 2S = (431, 45)$
  Add: $S := S + 3P = (403, 378)$
- [$i = 0$] Dbl: $S := 2S = (461, 250)$
  Dbl: $S := 2S = (389, 228)$
  Add: $S := S + P = (170, 25)$
  Dbl: $S := 2S = (541, 197)$

# Sliding (Non-Adjacent) Window Method

- Precompute only the odd multiples of $P$.
- Skip 0's after a window (do doubling operations only).
- The next window starts at the first 1 located after the last window.
- The next window is handled as in the windowed method with reduced precomputation.

---

- Example: Take $n = 410 = (110011010)_2$.
- The windows are: $\underline{110}$ 0 $\underline{110}$ $\underline{10}$.
- Now, the sequence of operations is:
  - Init $S$ to $\mathcal{O}$.
  - First window: Dbl, Dbl, Add ($3P$), Dbl.
  - Skip: Dbl.
  - Second window: Dbl, Dbl, Add ($3P$), Dbl.
  - Third window: Dbl, Add ($P$), Dbl.

# Signed Binary Representation

- Allow negative digits.
- Represent $n$ as $(n_t n_{t-1} n_{t-2} \ldots n_1 n_0)_2 = \sum_{i=0}^{t} n_i 2^i$ with each $n_i \in \{-1, 0, 1\}$.
- If no two consecutive digits are non-zero, this representation is called a **non-adjacent form (NAF)**.
- It is easy to precompute $-P$.
- Replace runs of consecutive 1's.
- $\ldots 0111110 \ldots$ can be replaced by $\ldots 100001̄0 \ldots$, where $\bar{1} = -1$.
- Signed-binary representation of $n$ is not unique. For example, $23 = 16 + 4 + 2 + 1 = (10111)_2 = 16 + 8 - 1 = (1100\bar{1})_2 = 32 - 8 - 1 = (10\bar{1}00\bar{1})_2$.
- The NAF representation is unique and has the least possible number of signed digits.

# Computation of NAF

- Let $n = (n_s n_{s-1} n_{s-2} \ldots n_1 n_0)_2$.
- We add $n$ with $2n$. The sum may have a bit-size two more than that of $n$.
- $$\begin{array}{r|ccccccccc} & & & & & & & & & \\ n & 0 & 0 & n_s & n_{s-1} & \ldots & n_2 & n_1 & n_0 \\ 2n & 0 & n_s & n_{s-1} & n_{s-2} & \ldots & n_1 & n_0 & 0 \\ \hline 3n & d_{s+1} & d_s & d_{s-1} & d_{s-2} & \ldots & d_1 & d_0 & n_0 \\ \hline \text{Output carry} & c_{s+2} & c_{s+1} & c_s & c_{s-1} & \ldots & c_2 & c_1 & c_0 \end{array}$$
- We have $c_{i+1} = \lfloor (n_i + n_{i+1} + c_i)/2 \rfloor$, and $d_i = n_i + n_{i+1} + c_i - 2c_{i+1}$.
- Now, we subtract $n$ from $3n$ and discard the rightmost zero bit. We do not do any borrow adjustment here, that is, $0 - 1$ is retained as $\bar{1} = -1$.
- $$\begin{array}{ccccccccc} 3n & d_{s+1} & d_s & d_{s-1} & d_{s-2} & \ldots & d_1 & d_0 & n_0 \\ n & 0 & 0 & n_s & n_{s-1} & \ldots & n_2 & n_1 & n_0 \\ \hline 2n & m_{s+1} & m_s & m_{s-1} & m_{s-2} & \ldots & m_1 & m_0 & 0 \end{array}$$
- Therefore, $m_i = d_i - n_{i+1} = n_i + c_i - 2c_{i+1}$.
- $d_i$ need not be computed. $c_{i+1}$ and $m_i$ can be computed from $n_i, n_{i+1}, c_i$ alone. Table lookup can be used (only eight cases).

# Computation of NAF: The Algorithm

- Let $n = (n_s n_{s-1} n_{s-2} \ldots n_1 n_0)_2$. We take $n_{s+1} = n_{s+2} = 0$.
- To compute the NAF $(m_{s+1} m_s m_{s-1} \ldots m_1 m_0)$ of $n$.

---

- Initialize $c = 0$.
- For $i = 0, 1, 2, \ldots, s+1$, repeat: /* You may use table lookup */
  - Set $c_{next} = \lfloor (n_i + n_{i+1} + c)/2 \rfloor$.
  - Set $m_i = n_i + c - 2c_{next}$.
  - Set $c = c_{next}$.
- Return $(m_{s+1} \ldots m_1 m_0)$.

---

- The digits are generated in the right-to-left order.
- The expansion must be *stored* for use in left-to-right scalar-multiplication algorithms.
- Algorithms for left-to-right generation of *optimal* signed binary representation are also known.

# Computation of NAF: Examples

Take $n = 23 = (10111)_2$.

- Computation of $n + 2n$:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $n = 23$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| $2n = 46$ | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| $3n = 69$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| Output carry | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

- Computation of $3n - n$:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $3n = 69$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| $n = 23$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| $2n = 46$ | 1 | 0 | $\bar{1}$ | 0 | 0 | $\bar{1}$ | 0 |

- Therefore, $n = 23 = (10\bar{1}00\bar{1})_2 = 2^5 - 2^3 - 2^0$.
- The NAF for 410 is $10\bar{1}010\bar{1}010$.
- For a 3-bit sliding window, we need to precompute $\pm P, \pm 3P, \pm 5P, \pm 7P$.
- Now, the odd-valued windows are $\underline{10\bar{1}}$ 0 $\underline{10\bar{1}}$ 0 $\underline{1}$ 0
- The NAF property guarantees that at least one zero exists between two consecutive windows.

# Width-$w$ Non-Adjacent Form ($w$NAF or NAF$_w$)

- Take an integer width $w \geqslant 2$.
- Represent $n$ in the base 2.
- The signed digits are zero or odd integers with absolute values $< 2^{w-1}$.
- Among any $w$ consecutive digits, at most one is non-zero.
- The $w$NAF representation is unique and optimal.
- The average density of non-zero digits in the $w$NAF representation is $1/(w+1)$.
- The basic NAF corresponds to $w = 2$.

---

- Some other variants based on addition chains
  - Signed fractional window method
  - Mixed radix
  - $\tau$-NAF (applicable to Koblitz curves)

# Computation of the $w$NAF

- Set $i = 0$.
- While $(n > 0)$, repeat:
  - If $n$ is even, set $m_i = 0$,
  - else set $r = n \operatorname{rem} 2^w$, if $r > 2^{w-1}$, set $r = r - 2^w$, set $m_i = r$ and $n = n - r$.
  - Set $n = n/2$ and increment $i$.
- Return $(m_{i-1}m_{i-2}\ldots m_2m_1m_0)$.

---

- This expansion is from right to left.
- If $n$ is even, then we get the next digit as 0.
- If $n$ is odd, we compute the next (odd) remainder $r$ of $n$ modulo $2^w$. It is ensured that $r$ lies in the range $[-(2^{w-1}-1), +(2^{w-1}-1)]$.
- When this $r$ is subtracted from $n$, it is guaranteed that the next $w-1$ digits are all 0.

# Computation of the *w*NAF: Example

Let us compute the width-4 NAF of $n = 1234567$.

| $i$ | $n$ | $m_i$ | $n - m_i$ | $(n - m_i)/2$ |
|---|---|---|---|---|
| 0 | 1234567 | 7 | 1234560 | 617280 |
| 1 | 617280 | 0 | | 308640 |
| 2 | 308640 | 0 | | 154320 |
| 3 | 154320 | 0 | | 77160 |
| 4 | 77160 | 0 | | 38580 |
| 5 | 38580 | 0 | | 19290 |
| 6 | 19290 | 0 | | 9645 |
| 7 | 9645 | −3 | 9648 | 4824 |
| 8 | 4824 | 0 | | 2412 |
| 9 | 2412 | 0 | | 1206 |
| 10 | 1206 | 0 | | 603 |
| 11 | 603 | −5 | 608 | 304 |
| 12 | 304 | 0 | | 152 |
| 13 | 152 | 0 | | 76 |
| 14 | 76 | 0 | | 38 |
| 15 | 38 | 0 | | 19 |
| 16 | 19 | 3 | 16 | 8 |
| 17 | 8 | 0 | | 4 |
| 18 | 4 | 0 | | 2 |
| 19 | 2 | 0 | | 1 |
| 20 | 1 | 1 | 0 | 0 |

$$\begin{aligned} 1234567 &= (10003000\bar{5}000\bar{3}0000007) \\ &= 2^{20} + 3 \times 2^{16} + (-5) \times 2^{11} + \\ &\quad (-3) \times 2^7 + 7. \end{aligned}$$

# Multiple Scalar Multiplication

Let $P, Q$ be elliptic-curve points, and $m, n$ positive integers of the same bit-size. We can compute $mP + nQ$ in a single loop.

---

- Precompute the point $P + Q$.
- Let $m = (m_s m_{s-1} m_{s-2} \ldots m_1 m_0)_2$ be the binary representation of $m$.
- Let $n = (n_s n_{s-1} n_{s-2} \ldots n_1 n_0)_2$ be the binary representation of $n$.
- Initialize $S = \mathscr{O}$.
- For $i = s, s-1, s-2, \ldots, 1, 0$, repeat:
  - Set $S = 2S$.
  - If $(m_i, n_i) = (1, 0)$, set $S = S + P$,
    else if $(m_i, n_i) = (0, 1)$, set $S = S + Q$,
    else if $(m_i, n_i) = (1, 1)$, set $S = S + (P + Q)$ (use precomputed value).
- Return $S$.

---

# Multiple Scalar Multiplication (Contd)

**Comparison with two scalar multiplications**

- The number of doubling operations is halved.
- On an average, the number of addition reduces from $s$ to $\frac{3}{4}s$.

**Windowed adaptation**

- Precompute $aP + bQ$ for all $a, b \in \{0, 1, 2, \ldots, 2^w - 1\}$.
- $w = 2$ is a practical choice.
- $w \geqslant 3$ calls for too much precomputation.

**Generalization to the sum of three (or more) scalar products**

- To compute $lP + mQ + nR$.
- Precompute $P + Q$, $P + R$, $Q + R$, and $P + Q + R$.
- Depending upon the bits $l_i, m_i, n_i$, add $P, Q, R$ or one of the precomputed points to $S$.

# Fixed-Base Scalar Multiplication

- We want to compute $nP$ for some $n \in \{0, 1, 2, \ldots, r-1\}$.
- Let the bit size of $r$ be $s$.
- Precompute and store $P, 2P, 4P, 8P, \ldots, 2^{s-1}P$.
- Express $n = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_k}$.
- Add the precomputed points $2^{i_j}P$.
- No doubling required.
- Huge permanent storage overhead.
- Efficient only when $P$ does not change frequently.

# Fixed-Base Multiple Scalar Multiplication

- To compute $mP + nQ$ with $s$-bit scalars $m$ and $n$.
- $P$ and $Q$ are assumed to be fixed.
- Precompute and store the points $2^i P$, $2^i Q$ and $2^i (P+Q)$ for all $i = 0, 1, 2, \ldots, s-1$.
- Let the $i$-th bits of $m$ and $n$ be $m_i$ and $n_i$.
  - If $(m_i, n_i) = (0, 0)$, do nothing.
  - If $(m_i, n_i) = (1, 0)$, add $2^i P$.
  - If $(m_i, n_i) = (0, 1)$, add $2^i Q$.
  - If $(m_i, n_i) = (0, 1)$, add $2^i (P+Q)$.
- No doubling needed.
- Huge permanent storage.
- If $P$ is fixed, but $Q$ changes frequently, the amortized cost of the precomputations of $2^i Q$ and $2^i (P+Q)$ may be high.

# Affine Curves

- $K$ is a field.
- $\overline{K}$ is the algebraic closure of $K$.
- It is often necessary to assume that $K$ is algebraically closed.
- **Affine plane:** $K^2 = \{(h,k) \mid h,k \in K\}$.
- For $(h,k) \in K^2$, the field elements $h,k$ are called **affine coordinates**.
- **Affine curve:** Defined by a polynomial equation:

$$C : f(X,Y) = 0.$$

- It is customary to consider only irreducible polynomials $f(X,Y)$. If $f(X,Y)$ admits non-trivial factors, the curve $C$ is the set-theoretic union of two (or more) curves of smaller degrees.
- **Rational points on $C$:** All points $(h,k) \in K^2$ such that $f(h,k) = 0$.
- Rational points on $C$ are called **finite points**.

# Affine Curves: Examples

- **Straight lines:** $aX + bY + c = 0$.
- **Circles:** $(X-a)^2 + (Y-b)^2 - r^2 = 0$.
- **Conic sections:** $aX^2 + bXY + cY^2 + dX + eY + f = 0$.
- **Elliptic curves:** Defined by the *Weierstrass equation*:
  $Y^2 + (a_1X + a_3)Y = X^3 + a_2X^2 + a_4X + a_6$.
  If $\operatorname{char} K \neq 2, 3$, this can be simplified as $Y^2 = X^3 + aX + b$.
- **Hyperelliptic curves of genus $g$:** $Y^2 + u(X)Y = v(X)$ with $\deg u \leqslant g$, $\deg v = 2g+1$, and $v$ monic.
  If $\operatorname{char} K \neq 2$, this can be simplified as $Y^2 = w(X)$ with $\deg w = 2g+1$ and $w$ monic.
- Parabolas are hyperelliptic curves of genus 0.
- Elliptic curves are hyperelliptic curves of genus 1.

# Projective Plane

- Define a relation $\sim$ on $K^3 \setminus \{(0,0,0)\}$ as $(h,k,l) \sim (h',k',l')$ if $h' = \lambda h$, $k' = \lambda k$ and $l' = \lambda l$ for some non-zero $\lambda \in K$.
- $\sim$ is an equivalence relation on $K^3 \setminus \{(0,0,0)\}$.
- The equivalence class of $(h,k,l)$ is denoted by $[h,k,l]$.
- $[h,k,l]$ can be identified with the line in $K^3$ passing through the origin and the point $(h,k,l)$.
- The set of all these equivalence classes is the **projective plane** over $K$.
- The projective plane is denoted as $\mathbb{P}^2(K)$.
- $h,k,l$ in $[h,k,l]$ are called **projective coordinates**.
- Projective coordinates are unique up to multiplication by non-zero elements of $K$.
- The three projective coordinates cannot be simultaneously 0.

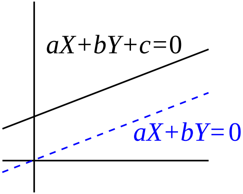
# Relation Between the Affine and the Projective Planes

- $\mathbb{P}^2(K)$ is the affine plane $K^2$ plus the points at infinity.
- Take $P = [h, k, l] \in \mathbb{P}^2(K)$.
- **Case 1:** $l \neq 0$.
  - $P = [h/l, k/l, 1]$ is identified with the point $(h/l, k/l) \in K^2$.
  - The line in $K^3$ corresponding to $P$ meets $Z = 1$ at $(h/l, k/l, 1)$.
  - $P$ is called a **finite point**.
- **Case 2:** $l = 0$.
  - The line in $K^3$ corresponding to $P$ does not meet $Z = 1$.
  - $P$ does not correspond to a point in $K^2$.
  - $P$ is a **point at infinity**.
  - For every slope of lines in the $X, Y$-plane, there exists exactly one point at infinity.
- A line passes through all the points at infinity. It is the **line at infinity**.
- Two distinct lines (parallel or not) in $\mathbb{P}^2(K)$ always meet at a unique point.
- Through any two distinct points in $\mathbb{P}^2(K)$ passes a unique line.

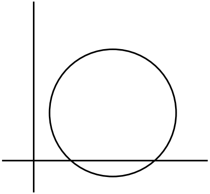
# Passage from Affine to Projective Curves

- A (multivariate) polynomial is called **homogeneous** if every non-zero term in the polynomial has the same degree.
- Example: $X^3 + 2XYZ - 3Z^3$ is homogeneous of degree 3. $X^3 + 2XY - 3Z$ is not homogeneous. The zero polynomial is homogeneous of any degree.
- Let $C : f(X,Y) = 0$ be an affine curve of degree $d$.
- $f^{(h)}(X,Y,Z) = Z^d f(X/Z, Y/Z)$ is the **homogenization** of $f$.
- $C^{(h)} : f^{(h)}(X,Y,Z) = 0$ is the **projective curve** corresponding to $C$.
- For any non-zero $\lambda \in K$, we have $f^{(h)}(\lambda h, \lambda k, \lambda l) = \lambda^d f^{(h)}(h,k,l)$. So $f^{(h)}(\lambda h, \lambda k, \lambda l) = 0$ if and only if $f^{(h)}(h,k,l) = 0$.
- The rational points of $C^{(h)}$ are all $[h,k,l]$ with $f^{(h)}(h,k,l) = 0$.
- **Finite points on $C^{(h)}$:** Put $Z = 1$ to get $f^{(h)}(X,Y,1) = f(X,Y)$. These are the points on $C$.
- **Points at infinity on $C^{(h)}$:** Put $Z = 0$ and solve for $f^{(h)}(X,Y,0) = 0$. These points do not belong to $C$.

# Examples of Projective Curves



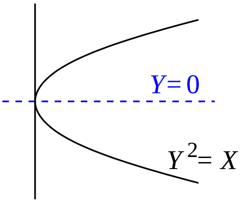

Straight Line



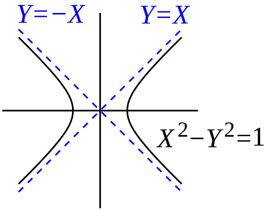
Circle

- **Straight line:** $aX + bY + cZ = 0$.
  - Finite points: Solutions of $aX + bY + c = 0$.
  - Points at infinity: Solve for $aX + bY = 0$.
    If $b \neq 0$, we have $Y = -(a/b)X$. So $[1, -(a/b), 0]$ is the only point at infinity.
    If $b = 0$, we have $aX = 0$, that is, $X = 0$. So $[0, 1, 0]$ is the only point at infinity.
- **Circle:** $(X - aZ)^2 + (Y - bZ)^2 = r^2 Z^2$.
  - Finite points: Solutions of $(X - a)^2 + (Y - b)^2 = r^2$.
  - Points at infinity: Solve for $X^2 + Y^2 = 0$.
    For $K = \mathbb{R}$, the only solution is $X = Y = 0$, so there is no point at infinity.
    For $K = \mathbb{C}$, the solutions are $Y = \pm \mathrm{i} X$, so there are two points at infinity: $[1, \mathrm{i}, 0]$ and $[1, -\mathrm{i}, 0]$.

# Examples of Projective Curves (contd.)



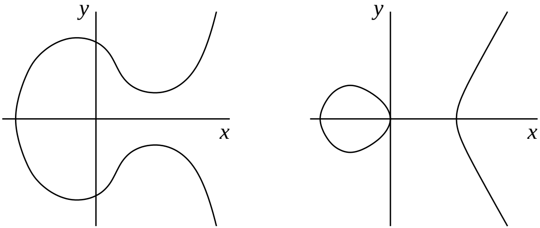

Parabola




Hyperbola

- **Parabola:** $Y^2 = XZ$.
  - Finite points: Solutions of $Y^2 = X$.
  - Points at infinity: Solve for $Y^2 = 0$.
    $Y = 0$, so $[1,0,0]$ is the only point at infinity.
- **Hyperbola:** $X^2 - Y^2 = Z^2$.
  - Finite points: Solutions of $X^2 - Y^2 = 1$.
  - Points at infinity: Solve for $X^2 - Y^2 = 0$.
    $Y = \pm X$, so there are two points at infinity: $[1,1,0]$ and $[1,-1,0]$.

# Examples of Projective Curves (contd.)




- **Elliptic curve:** $Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$.
  - Finite points: Solutions of $Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$.
  - Points at infinity: Solve for $X^3 = 0$.
    $X = 0$, that is, $[0, 1, 0]$ is the only point at infinity.

# Elliptic-Curve Arithmetic in Projective Coordinates

Consider the simple Weierstrass equation $E : y^2 = x^3 + ax + b$.
Let $P = [h_1, k_1, l_1]$ and $Q = [h_2, k_2, l_2]$ in projective coordinates.
We want to compute $P + Q = [h, k, l]$ and $2P = [h', k', l']$.

- The slope of the line passing through $P$ and $Q$ is

$$\lambda = \frac{\frac{k_2}{l_2} - \frac{k_1}{l_1}}{\frac{h_2}{l_2} - \frac{h_1}{l_1}} = \frac{k_2 l_1 - k_1 l_2}{h_2 l_1 - h_1 l_2}.$$

- Therefore,

$$\frac{h}{l} = \lambda^2 - \frac{h_1}{l_1} - \frac{h_2}{l_2} = \frac{l_1 l_2 (k_2 l_1 - k_1 l_2)^2 - (h_2 l_1 - h_1 l_2)^2 (h_1 l_2 + h_2 l_1)}{l_1 l_2 (h_2 l_1 - h_1 l_2)^2},$$

and

$$\frac{k}{l} = \lambda \left( \frac{h_1}{l_1} - \frac{h}{l} \right) - \frac{k_1}{l_1}.$$

Substituting the values of $\lambda$ and $h/l$ gives an explicit expression for $k/l$.

- These expressions are too clumsy.

# Elliptic-Curve Addition in Projective Coordinates

Practical solution: Collect common subexpressions.

$$
\begin{aligned}
T_1 &= k_2 l_1 - k_1 l_2, \\
T_2 &= h_2 l_1 - h_1 l_2, \\
T_3 &= T_2^2, \\
T_4 &= T_2 T_3, \\
T_5 &= l_1 l_2 T_1^2 - T_4 - 2 h_1 l_2 T_3, \\
h &= T_2 T_5, \\
k &= T_1 (h_1 l_2 T_3 - T_5) - k_1 l_2 T_4, \\
l &= l_1 l_2 T_4.
\end{aligned}
$$

Further optimization possible by storing $h_1 l_2$, $k_1 l_2$ and $l_1 l_2$ in temporary variables.

# Elliptic-Curve Doubling in Projective Coordinates

The projective coordinates $h', k', l'$ of $2P$ can be computed by the following formulas.

$$
\begin{aligned}
T_1 &= 3h_1^2 + al_1^2, \\
T_2 &= k_1 l_1, \\
T_3 &= h_1 k_1 T_2, \\
T_4 &= T_1^2 - 8T_3, \\
T_5 &= T_2^2, \\
h' &= 2T_2 T_4, \\
k' &= T_1(4T_3 - T_4) - 8k_1^2 T_5, \\
l' &= 8T_2 T_5.
\end{aligned}
$$

# Projective Coordinates and Scalar Multiplication

- Computing the affine coordinates requires a division in the field. (Recall the computation of the slope $\lambda$.)
- Division could be much costlier than multiplication and squaring in the field.
- Projective addition and doubling formulas do not use any division.
- At the end of the loop, the sum is converted from $[h,k,l]$ to $(h/l,k/l)$ by a single division.
- Projective coordinates increase the number of multiplication and squaring operations substantially.
- In some situations, speedup is reported with projective coordinates.

# Mixed Coordinates

- The left-to-right multiplication conditionally adds $P$ to $S$.
- The windowed variant adds $aP$ to $S$ for a small $a$.
- $P$ is available in affine coordinates.
- The small multiples of $P$ can be computed in affine coordinates.
- Adding $S = [h_1, k_1, l_1]$ and $aP = (h_2, k_2)$ is same as adding $[h_1, k_1, l_1]$ and $[h_2, k_2, 1]$.
- Since $l_2 = 1$, the addition algorithm can be simplified, and many operations can be saved.
- For example,

$$T_1 = k_2 l_1 - k_1 l_2$$

now becomes

$$T_1 = k_2 l_1 - k_1.$$

# Generalized Projective Coordinates

- Let $c, d$ be positive integers. Assume that $\gcd(c,d) = 1$.
- Define an equivalence relation on $K^3 \setminus \{(0,0,0)\}$ as $(h,k,l) \sim (h',k',l')$ if and only if $h' = \lambda^c h$, $k' = \lambda^d k$, and $l' = \lambda l$ for some non-zero $\lambda \in K$.
- Call the equivalence class of $(h,k,l)$ as $[h,k,l]_{c,d}$.
- Identify the finite point $(h,k)$ with $[h,k,1]_{c,d}$.
- Identify the finite point $[h,k,l]_{c,d}$ with $(h/l^c, k/l^d)$.
- Homogenization requires replacing $x$ by $X/Z^c$ and $y$ by $Y/Z^d$.
- Give the weight $c$ to $X$, the weight $d$ to $Y$, and the weight 1 to $Z$.
- Each non-zero term in the homogenization is of the same total weight.

# Generalized Projective Coordinates: Examples

- The standard projective coordinates correspond to $c = d = 1$.
- **Jacobian Coordinates:** The weights are $c = 2$ and $d = 3$.
- **López–Dahab Coordinates:** The weights are $c = 1$ and $d = 2$.
- For certain curves, generalized coordinates reduce the operation counts for point addition and doubling.
- The use of mixed coordinates can produce further speedup.

# Montgomery Ladders

- A modification of the left-to-right scalar multiplication.
- Two points $S$ and $T$ are computed in the loop.
- Invariance: $T = S + P$.

---

- Initialize $S = \mathcal{O}$ and $T = P$.
- For $i = s, s-1, s-2, \ldots, 1, 0$, repeat:
- If $(n_i = 0)$ /* Update $(S,T)$ to $(2S, 2S+P) = (2S, S+T)$ */
- Assign $T = S + T$ and $S = 2S$.
- else /* Update $(S,T)$ to $(2S+P, 2S+2P) = (S+T, 2T)$ */
- Assign $S = S + T$ and $T = 2T$.
- Return $S$.

---

- The Montgomery ladder is resistant to side-channel attacks.
- The Montgomery ladder is unlikely to be adaptable to windowed variants.

# Montgomery Ladders (Contd)

- Consider the curve $E : y^2 = x^3 + ax + b$.
- Let $P = (h_1, k_1)$, $Q = (h_2, k_2)$, $P + Q = (h_3, k_3)$, and $P - Q = (h_4, k_4)$. Suppose $P \neq Q$. The addition formula gives

$$\begin{aligned} (h_1 - h_2)^2 h_3 &= (h_1 + h_2)(h_1 h_2 + a) + 2b - 2k_1 k_2, \\ (h_1 - h_2)^2 h_4 &= (h_1 + h_2)(h_1 h_2 + a) + 2b + 2k_1 k_2. \end{aligned}$$

  Multiply these two formulas and substitute $k_1^2 = h_1^3 + ah_1 + b$ and $k_2^2 = h_2^3 + ah_2 + b$ to get

$$h_3 h_4 (h_1 - h_2)^2 = (h_1 h_2 - a)^2 - 4b(h_1 + h_2).$$

  Given $h_1, h_2, h_4$ alone, one can compute $h_3$.
- The $x$-coordinate $h_5$ of $2P$ can be computed from $h_1$ alone:

$$4h_5(h_1^3 + ah_1 + b) = (h_1^2 - a)^2 - 8bh_1.$$

# Montgomery Ladders (Contd)

- We always have $S - T = -P$. Moreover, $x(-P) = x(P)$.
- There is no need to compute any $y$-coordinate in the Montgomery ladder.
- Denote $kP = (x_k, y_k)$. Therefore, $P = (x_1, y_1)$ is known.
- The Montgomery loop computes $x_n = x(S)$ and $x_{n+1} = x(T)$. From these, the $y$-coordinate of $S = nT$ is computed as

$$y_n = \frac{(x_1 + x_n)(x_1 x_n + a) + 2b - (x_1 - x_n)^2 x_{n+1}}{2y_1}.$$

- Each iteration needs one addition and one doubling.
- Montgomery ladders are particularly attractive for curves of the form

$$By^2 = x^3 + Ax^2 + x.$$

  Projective coordinates help for these curves.
- Every curve of the form $y^2 = x^3 + ax + b$ (like a curve of large prime order) cannot be converted to the Montgomery form.

# PART 4

## *PAIRING ON ELLIPTIC CURVES*

# Weil Pairing

Let $E$ be an elliptic curve defined over a finite field $K = \mathbb{F}_q$.
Take a positive integer $m$ coprime to $p = \operatorname{char} K$.
Let $\mu_m$ denote the set of $m$-th roots of unity in $\overline{K}$.
We have $\mu_m \subseteq \mathbb{F}_{q^k}$, where $k = \operatorname{ord}_m(q)$ is called the **embedding degree**.
Let $E[m]$ be those points in $E = E(\overline{K})$, whose orders divide $m$.

**Weil pairing** is a function $e_m : E[m] \times E[m] \to \mu_m$.

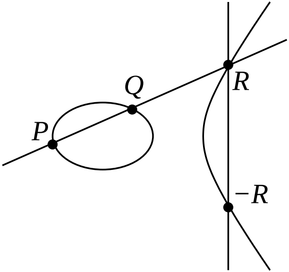
- **Bilinearity:**

$$\begin{aligned} e_m(P+Q,R) &= e_m(P,R)e_m(Q,R), \\ e_m(P,Q+R) &= e_m(P,Q)e_m(P,R). \end{aligned}$$

- **Alternation:** $e_m(P,P) = 1$.
- **Skew symmetry:** $e_m(Q,P) = e_m(P,Q)^{-1}$.
- **Non-degeneracy:** If $P \neq \mathcal{O}$, then $e_m(P,Q) \neq 1$ for some $Q \in E[m]$.
- If $m$ is a prime and $P \neq \mathcal{O}$, then $e_m(P,Q) = 1$ if and only if $Q$ lies in the subgroup generated by $P$ (that is, $Q = aP$ for some integer $a$).

# Line Functions

To compute the equation of the line $L_{P,Q}$ or the vertical line $L_{R,-R}$.




- If $P = Q = \mathcal{O}$, return 1.
- If $P = \mathcal{O}$, return $x - x(Q)$.
- If $Q = \mathcal{O}$, return $x - x(P)$.
- If $P = -Q$, return $x - x(P)$.
- Now, let $P = (h_1, k_1)$ and $Q = (h_2, k_2)$.
- If $P = Q$, take $\lambda = \dfrac{3h_1^2 + a}{2k_1}$, else take $\lambda = \dfrac{k_2 - k_1}{h_2 - h_1}$.
- Set $\mu = \lambda h_1 - k_1$.
- Return $y - \lambda x + \mu$.

# The Functions $f_{n,P}$ ($n \in \mathbb{Z}$, $P \in E(\bar{K})$)

- These are rational functions unique up to multiplication by elements of $\bar{K}^*$.
- $f_{n,P}$ satisfy the recurrence relation:

$$\begin{aligned} f_{0,P} &= f_{1,P} = 1, \\ f_{n+1,P} &= \left(\frac{L_{P,nP}}{L_{(n+1)P,-(n+1)P}}\right) f_{n,P} \text{ for } n \geqslant 1, \\ f_{-n,P} &= \frac{1}{f_{n,P}} \text{ for } n \geqslant 1. \end{aligned}$$

- The rational functions $f_{n,P}$ also satisfy

$$f_{n+n',P} = f_{n,P}\, f_{n',P} \times \left(\frac{L_{nP,n'P}}{L_{(n+n')P,-(n+n')P}}\right).$$

- In particular, for $n = n'$, we have

$$f_{2n,P} = f_{n,P}^2 \times \left(\frac{L_{nP,nP}}{L_{2nP,-2nP}}\right).$$

- The function $f_{n,P}$ is usually kept in the factored form.
- The value of $f_{n,P}$ at some point $Q$ is usually needed.

# Miller's Algorithm for Computing $f_{n,P}$

- **Input:** A point $P \in E$ and a positive integer $n$.
- **Output:** The rational function $f_{n,P}$.
- **Steps**
  - Let $n = (n_s n_{s-1} \ldots n_1 n_0)_2$ be the binary representation of $n$ with $n_s = 1$.
  - Initialize $f = 1$ and $U = P$.
  - For $i = s-1, s-2, \ldots, 1, 0$, do the following:
    - /* Doubling */
      Update $f = f^2 \times \left(\frac{L_{U,U}}{L_{2U,-2U}}\right)$ and $U = 2U$.
    - /* Conditional adding */
      If $(n_i = 1)$, update $f = f \times \left(\frac{L_{U,P}}{L_{U+P,-(U+P)}}\right)$ and $U = U + P$.
  - Return $f$.
- **Note:** One may supply a point $Q \in E$ and wish to compute the value $f_{n,P}(Q)$ (instead of the function $f_{n,P}$). In that case, the functions $L_{U,U}/L_{2U,-2U}$ and $L_{U,P}/L_{U+P,-(U+P)}$ should be evaluated at $Q$ before multiplication with $f$.

# Weil Pairing and the Functions $f_{n,P}$

Let $P, Q \in E[m]$, and we want to compute $e_m(P, Q)$.

- Choose a point $T$ not equal to $\pm P, -Q, Q - P, \mathcal{O}$.
- We have $e_m(P,Q) = \dfrac{f_{m,Q}(T)\ f_{m,P}(Q-T)}{f_{m,P}(-T)\ f_{m,Q}(P+T)}$.
- If $P \neq Q$, then we also have $e_m(P,Q) = (-1)^m\ \dfrac{f_{m,P}(Q)}{f_{m,Q}(P)}$.
- Miller's algorithm for computing $f_{n,P}(Q)$ can be used.
- All these invocations of Miller's algorithm have $n = m$.
- So a single double-and-add loop suffices.
- For efficiency, one may avoid the division operations in Miller's loop by separately maintaining polynomial expressions for the numerator and the denominator of $f$. After the loop terminates, a single division is made.

# Miller's Algorithm for Computing $e_m(P,Q)$

- If $(P = Q)$, return 1.
- Let $m = (1m_{s-1} \ldots m_1 m_0)_2$ be the binary representation of $m$.
- Initialize $f_{num} = f_{den} = 1$, $U = P$, and $V = Q$.
- For $i = s-1, s-2, \ldots, 1, 0$, repeat:

  /* Doubling */
  - Update numerator $f_{num} = f_{num}^2 \times L_{U,U}(Q) \times L_{2V,-2V}(P)$.
  - Update denominator $f_{den} = f_{den}^2 \times L_{2U,-2U}(Q) \times L_{V,V}(P)$.
  - Update $U = 2U$ and $V = 2V$.

  /* Conditional adding */
  - If $(m_i = 1)$, then execute the following three lines:
    - Update numerator $f_{num} = f_{num} \times L_{U,P}(Q) \times L_{V+Q,-(V+Q)}(P)$.
    - Update denominator $f_{den} = f_{den} \times L_{U+P,-(U+P)}(Q) \times L_{V,Q}(P)$.
    - Update $U = U + P$ and $V = V + Q$.
- /* End of for loop */
- Return $(-1)^m f_{num}/f_{den}$.

# Weil Pairing: Example

- Take $E : Y^2 = X^3 + 3X$ defined over $\mathbb{F}_{43}$.
- This is supersingular with $|E(\mathbb{F}_{43})| = 44$, and $E(\mathbb{F}_{43}) \cong \mathbb{Z}_{22} \oplus \mathbb{Z}_2$.
- Take $m = 11$. The embedding degree for this choice is $k = 2$.
- We work in the field $\mathbb{F}_{43^2} = \mathbb{F}_{1849} = \mathbb{F}_{43}(\theta)$, where $\theta^2 + 1 = 0$.
- $\mathbb{F}^*_{43^2}$ contains all the 11-th roots of unity: $1$, $2+13\theta$, $2+30\theta$, $7+9\theta$, $7+34\theta$, $11+3\theta$, $11+40\theta$, $18+8\theta$, $18+35\theta$, $26+20\theta$, and $26+23\theta$.
- $E(\mathbb{F}_{43^2}) \cong \mathbb{Z}_{44} \oplus \mathbb{Z}_{44}$ contains $E[11] \cong \mathbb{Z}_{11} \oplus \mathbb{Z}_{11}$.
- $P = (1,2)$ and $Q = (-1, 2\theta)$ generate $E[11]$.
- Let us compute $e_m(P,Q)$ for $P := P = (1,2)$ and $Q := 4P + 5Q = (15 + 22\theta, 5 + 14\theta)$.
- $11 = (1011)_2$.
- Initialization: $f = f_{num}/f_{den} = 1/1$, $U = P$, and $V = Q$.

# Miller Iteration for $i = 2$

## Doubling

- $\Lambda_1 = L_{U,U}/L_{2U,-2U} = \dfrac{y + 20x + 21}{x + 32}$
- $\Lambda_2 = L_{2V,-2V}/L_{V,V} = \dfrac{x + (36 + 21\theta)}{y + (12 + 35\theta)x + (26 + 14\theta)}$
- $f = f^2 \times \dfrac{\Lambda_1(Q)}{\Lambda_2(P)} = \dfrac{34 + 37\theta}{28 + \theta}$
- $U = 2P = (11, 26)$ and $V = 2Q = (7 + 22\theta, 28 + 7\theta)$

## Addition

- $m_2 = 0$, so addition is skipped.

# Miller Iteration for $i = 1$

**Doubling**

- $\Lambda_1 = L_{U,U}/L_{2U,-2U} = \dfrac{y + 31x + 20}{x + 7}$
- $\Lambda_2 = L_{2V,-2V}/L_{V,V} = \dfrac{x + (2 + 26\theta)}{y + (18 + 22\theta)x + (29 + 2\theta)}$
- $f = f^2 \times \dfrac{\Lambda_1(Q)}{\Lambda_2(P)} = \dfrac{12 + 15\theta}{25 + 18\theta}$
- $U = 4P = (36, 18)$ and $V = 4Q = (41 + 17\theta, 6 + 6\theta)$

**Addition**

- $\Lambda_1 = L_{U,P}/L_{U+P,-(U+P)} = \dfrac{y + 2x + 39}{x + 33}$
- $\Lambda_2 = L_{V+Q,-(V+Q)}/L_{V,Q} = \dfrac{x + (41 + 8\theta)}{y + (28 + 9\theta)x + (31 + 9\theta)}$
- $f = f^2 \times \dfrac{\Lambda_1(Q)}{\Lambda_2(P)} = \dfrac{25 + 15\theta}{28 + 20\theta}$
- $U = 5P = (10, 16)$ and $V = 5Q = (2 + 35\theta, 30 + 18\theta)$

# Miller Iteration for $i = 0$

**Doubling**

- $\Lambda_1 = L_{U,U}/L_{2U,-2U} = \dfrac{y+8x+33}{x+42}$
- $\Lambda_2 = L_{2V,-2V}/L_{V,V} = \dfrac{x+(28+21\theta)}{y+(19+16\theta)x+(19+16\theta)}$
- $f = f^2 \times \dfrac{\Lambda_1(Q)}{\Lambda_2(P)} = \dfrac{10+22\theta}{12+28\theta}$
- $U = 10P = (1,41)$ and $V = 10Q = (15+22\theta, 38+29\theta)$

**Addition**

- $\Lambda_1 = L_{U,P}/L_{U+P,-(U+P)} = \dfrac{x+42}{1}$
- $\Lambda_2 = L_{V+Q,-(V+Q)}/L_{V,Q} = \dfrac{1}{x+(28+21\theta)}$
- $f = f^2 \times \dfrac{\Lambda_1(Q)}{\Lambda_2(P)} = \dfrac{12\theta}{18+32\theta}$
- $U = 11P = \mathcal{O}$ and $V = 11Q = \mathcal{O}$

# Weil Pairing: Example

We have $e_m(P,Q) = (-1)^{11}\left(\frac{12\theta}{18+32\theta}\right) = 26+20\theta$. This is indeed an 11-th root of unity.

- If $P,Q$ are linearly dependent, we have $e_m(P,Q) = 1$.
- The Miller loop may encounter a *division by zero* error in this case.
- Use the alternative formula

$$e_m(P,Q) = \frac{f_{m,Q}(T)\ f_{m,P}(Q-T)}{f_{m,P}(-T)\ f_{m,Q}(P+T)}$$

for a randomly chosen point $T$.

# Tate Pairing

Let $E$ be an elliptic curve defined over $K = \mathbb{F}_q$ with $p = \operatorname{char} K$.
Let $m$ be a positive integer coprime to $p$.
Let $k = \operatorname{ord}_m(q)$ (the **embedding degree**), and $L = \mathbb{F}_{q^k}$.
Let $E[m] = \{P \in E(\overline{K}) \mid mP = \mathcal{O}\}$, and $mE(L) = \{mP \mid P \in E(L)\}$.
Let $(L^*)^m = \{a^m \mid a \in L^*\}$ be the set of $m$-th powers in $L^*$.

- Let $P$ be a point in $E[m]$, and $Q$ a point in $E(L)$.
- The **Tate pairing** is a function

$$\langle\ ,\ \rangle_m : E[m] \times E(L)/mE(L) \to L^*/(L^*)^m$$

that maps a pair of points $P, Q$ to $\langle P, Q\rangle_m$.

- $Q$ should be regarded as a point in $E(L)/mE(L)$.
- The value of $\langle P, Q\rangle_m$ is unique up to multiplication by an $m$-th power of a non-zero element of $L$, that is, $\langle P, Q\rangle_m$ is unique in $L^*/(L^*)^m$.

# Properties of Tate Pairing

- **Bilinearity:**

$$\begin{aligned} \langle P+Q,R\rangle_m &= \langle P,R\rangle_m \langle Q,R\rangle_m, \\ \langle P,Q+R\rangle_m &= \langle P,Q\rangle_m \langle P,R\rangle_m. \end{aligned}$$

- **Non-degeneracy:** For every $P \in E[m]$, $P \neq \mathscr{O}$, there exists $Q$ with $\langle P,Q\rangle_m \neq 1$. For every $Q \notin mE(L)$, there exists $P \in E[m]$ with $\langle P,Q\rangle_m \neq 1$.
- The Weil pairing is related to the Tate pairing as

$$e_m(P,Q) = \frac{\langle P,Q\rangle_m}{\langle Q,P\rangle_m}$$

  up to $m$-th powers.
- Let $k = \mathrm{ord}_m(q)$ be the embedding degree. The Tate pairing can be made unique by exponentiation to the power $(q^k-1)/m$:

$$\hat{e}_m(P,Q) = (\langle P,Q\rangle_m)^{\frac{q^k-1}{m}}$$

  $\hat{e}_m(P,Q)$ is called the **reduced Tate pairing**. The reduced pairing continues to exhibit bilinearity and non-degeneracy.

# Computing the Tate Pairing

- Take a point $T \neq P, -Q, P-Q, \mathscr{O}$.
- We have $\langle P, Q \rangle_m = \dfrac{f_{m,P}(Q+T)}{f_{m,P}(T)}$.
- If $P$ and $Q$ are linearly independent, then $\langle P, Q \rangle_m = f_{m,P}(Q)$.
- Miller's algorithm is used to compute $\langle P, Q \rangle_m$.
- A single double-and-add loop suffices.
- For efficiency, the numerator and the denominator in $f$ may be updated separately. After the loop, a single division is made.
- If the reduced pairing is desired, then a **final exponentiation** to the power $(q^k - 1)/m$ is made on the value returned by Miller's algorithm.

# Weil vs. Tate Pairing

- The Miller loop for Tate pairing is more efficient than that for Weil pairing.
- The reduced Tate pairing demands an extra exponentiation.
- Let $k = \text{ord}_m(q)$ be the embedding degree, and $L = \mathbb{F}_{q^k}$.
- Tate pairing requires working in the field $L$.
- Let $L'$ be the field obtained by adjoining to $L$ the coordinates of all the points of $E[m]$.
- Weil pairing requires working in the field $L'$.
- $L'$ is potentially much larger than $L$.
- **Special case:** $m$ is a prime divisor of $|E(K)|$ with $m \not| q$ and $m \not| (q-1)$. Then, $L' = L$. So it suffices to work in the field $L$ only.
- For cryptographic applications, Tate pairing is used more often that Weil pairing.
- One takes $\mathbb{F}_q$ with $|q|$ about 500–2000 bits and $k \leqslant 12$. Larger embedding degrees are impractical for implementation.

# Distortion Maps

Let $m$ be a prime divisor of $|E(K)|$.
Let $P$ be a generator of a subgroup $G$ of $E(K)$ of order $m$.
**Goal:** To define a pairing of the points in $G$.

- If $k = 1$ (that is, $L = K$), then $\langle P, P\rangle_m \neq 1$.
- **Bad news:** If $k > 1$, then $\langle P, P\rangle_m = 1$.
  But then, by bilinearity, $\langle Q, Q'\rangle_m = 1$ for all $Q, Q' \in G$.
- **A way out:** If $k > 1$ and $Q \in L$ is linearly independent of $P$ (that is, $Q \notin G$), then $\langle P, Q\rangle_m \neq 1$.
- Let $\phi : E(L) \to E(L)$ be an endomorphism of $E(L)$ with $\phi(P) \notin G$.
  $\phi$ is called a **distortion map**.
- Define the **distorted Tate pairing** of $P, Q \in G$ as $\langle P, \phi(Q)\rangle_m$.
- Since $\phi(P)$ is linearly independent of $P$, we have $\langle P, \phi(P)\rangle_m \neq 1$.
- Since $\phi$ is an endomorphism, bilinearity is preserved.
- **Symmetry:** We have $\langle Q, \phi(Q')\rangle_m = \langle Q', \phi(Q)\rangle_m$ for all $Q, Q' \in G$.
- Distortion maps exist only for supersingular curves.

# Twists

Let $E$ be defined by the short Weierstrass equation $Y^2 = X^3 + aX + b$.
Let $d \geqslant 2$, and $v \in \mathbb{F}_q^*$ a $d$-th power non-residue.

- Consider the curve $E' : Y^2 = X^3 + v^{4/d}aX + v^{6/d}b$ (defined over $\mathbb{F}_{q^d}$).
- If $d = 2$, then $E'$ is defined over $\mathbb{F}_q$ itself.
- $E'$ is called a **twist of $E$ of degree $d$**.
- $E$ and $E'$ are isomorphic over $\mathbb{F}_{q^d}$. An explicit isomorphism is given by the map $\phi_d : E' \to E$ taking $(h,k) \mapsto (v^{-2/d}h, v^{-3/d}k)$.
- Let $m$ be a prime divisor of $|E(\mathbb{F}_q)|$, $G$ a subgroup of order $m$ in $E(\mathbb{F}_{q^k})$, and $G'$ a subgroup of order $m$ in $E'(\mathbb{F}_{q^k})$. Let $P, P'$ be generators of $G$ and $G'$. Suppose that $\phi_d(P')$ is linearly independent of $P$.
- For $d = 2$ (**quadratic twist**), a natural choice is $G \subseteq E(\mathbb{F}_q)$ and $G' \subseteq E'(\mathbb{F}_q)$.
- Define a pairing of points $Q \in G$ and $Q' \in G'$ as $\langle Q, \phi_d(Q') \rangle_m$.
- This is called the **twisted Tate pairing**.

# Pairing-Friendly Curves

- **Requirement for efficient computation:** Small embedding degree $k$.
- For general curves, $k$ is quite high ($|k| \approx |m|$).
- Only some specific types of curves qualify as pairing-friendly.
- **Supersingular curves**
  - By Hasse's Theorem, $|E(\mathbb{F}_q)| = q + 1 - t$ with $|t| \leqslant 2\sqrt{q}$.
  - If $p|t$, we call $E$ a **supersingular curve**.
  - Curves of the form $Y^2 + aY = X^3 + bX + c$ are supersingular over fields of characteristic 2.
  - Supersingular curves have small embedding degrees. The only possibilities are $1, 2, 3, 4, 6$.
  - If $\mathbb{F}_q$ is a prime field with $q \geqslant 5$, the only possibility is $k = 2$.
  - Non-supersingular curves are called **ordinary curves**.
  - It is difficult to locate ordinary curves with small embedding degrees.

# Supersingular Curves: Examples

- $E : Y^2 = X^3 + a$ defined over $\mathbb{F}_p$ with an odd prime $p \equiv 2 \pmod 3$.
  Embedding degree: $k = 2$.
- $E : Y^2 = X^3 + aX$ defined over $\mathbb{F}_p$ with an odd prime $p \equiv 3 \pmod 4$.
  Embedding degree: $k = 2$.
- $E : Y^2 + Y = X^3 + X + a$ with $a = 0$ or $1$ defined over $\mathbb{F}_{2^d}$ with odd $d$.
  Embedding degree: $k = 4$.
- $E : Y^2 = X^3 - X \pm 1$ defined over $\mathbb{F}_{3^d}$ with $2, 3 \nmid d$.
  Embedding degree: $k = 6$.
- $E : Y^2 = X^3 + a$ defined over $\mathbb{F}_{p^2}$ with a prime $p \equiv 5 \pmod 6$ and with $a \in \mathbb{F}_{p^2}$ a square but not a cube.
  Embedding degree: $k = 3$.
- Let $E$ be a supersingular curve defined over $\mathbb{F}_p$ with $p \geqslant 5$. Then, $E$ as a curve over $\mathbb{F}_{p^n}$ with even $n$ is again supersingular.
  Embedding degree: $k = 1$.

# How to Find Ordinary Pairing-Friendly Curves

- Let $k$ be a positive integer, and $\Delta$ a small positive square-free integer.
- Search for integer-valued polynomials $t(x), m(x), q(x) \in \mathbb{Q}[x]$ to represent a family of elliptic curves of embedding degree $k$ and discriminant $\Delta$. The triple $(t, m, q)$ should satisfy the following:

1. $q(x) = p(x)^n$ for some $n \in \mathbb{N}$ and $p(x) \in \mathbb{Q}[x]$ representing primes.
2. $m(x)$ is irreducible with a positive leading coefficient.
3. $m(x) | q(x) + 1 - t(x)$.
4. $m(x) | \Phi_k(t(x) - 1)$, where $\Phi_k$ is the $k$-th cyclotomic polynomial.
5. There are infinitely many integers $(x, y)$ satisfying $\Delta y^2 = 4q(x) - t(x)^2$.

- If $y$ in Condition 5 can be parametrized by a polynomial $y(x) \in \mathbb{Q}[x]$, the family is called **complete**, otherwise it is called **sparse**.
- For obtaining ordinary curves, we require $\gcd(q(x), m(x)) = 1$.
- The **complex multiplication method** is used to obtain specific examples of elliptic curves $E$ over $\mathbb{F}_q$ with $E(\mathbb{F}_q)$ having a subgroup of order $m$.

# Some Families of Ordinary Pairing-Friendly Curves

- Some sparse families of ordinary pairing-friendly curves are:
  - **MNT (Miyaji–Nakabayashi–Takano) curves:** These are curves of prime orders with embedding degrees 3, 4 or 6.
  - **Freeman curves:** These curves have embedding degree 10.
- Some complete families of ordinary pairing-friendly curves are:
  - **BN (Barreto–Naehrig) curves:** These curves have embedding degree 12 and discriminant 3.
  - **SB (Scott–Barreto) curves**
  - **BLS (Barreto–Lynn–Scott) curves**
  - **BW (Brezing–Weng) curves**

# Efficient Implementations of Pairing

- **Denominator elimination:** Applicable to Tate pairing.
- Let the embedding degree $k = 2d$ be even.
- $f_{n,P}(Q)$ is computed by Miller's algorithm, where $Q = (h,k)$ with $h \in \mathbb{F}_{q^d}$.
- The denominators $L_{2U,-2U}(Q)$ and $L_{U+P,-(U+P)}(Q)$ correspond to vertical lines, evaluate to elements of $\mathbb{F}_{q^d}$, and can be discarded.
- The final exponentiation guarantees correct computation of Tate pairing.

---

- **BMX (Blake-Murty-Xu) refinements** use 2-bit windows in Miller's loop.

---

- **Loop reduction:** With clever modifications to Tate pairing, the number of iterations in the Miller loop can be substantially reduced.
- A typical reduction is by a factor of 2.
- **Examples**
  - **$\eta$ and $\eta_T$ pairings** (for supersingular curves)
  - **Ate pairing** (for ordinary curves)
  - **R-ate pairing**

# PART 5

## *PAIRING-BASED CRYPTOGRAPHY*

# Intractable Problems (Contd)

Let $G$ be a finite cyclic additive group with a generator $P$, and $G'$ a finite cyclic multiplicative group. We assume that $|G| = r$ is a prime. Suppose that $e : G \times G \to G'$ is an efficiently computable pairing.

- **Decisional Diffie–Hellman Problem (DDHP):** Given $aP, bP, zP \in G$ (but not $a$, $b$ and $z$), decide whether $zP = abP$, that is, whether $z \equiv ab \pmod{r}$.
- The existence of the pairing function $e$ makes the DDHP in $G$ easy. In fact, $z \equiv ab \pmod{r}$ if and only if $e(aP, bP) = e(P, zP)$. In this case, $G$ is called a **Gap Diffie–Hellman (GDH) group**.
- In a GDH group, given $aP, bP$, it is easy to compute $e(P, P)^{ab} = e(aP, bP)$.

# The Problems That Are Intractable in Presence of Pairing

- **Bilinear Diffie–Hellman Problem (BDHP):** Given $P, aP, bP, cP \in G$, $P \neq 0$, compute $e(P,P)^{abc}$.
- **Decisional Bilinear Diffie–Hellman Problem (DBDHP):** Given $P, aP, bP, cP, zP \in G$, $P \neq 0$, decide whether $e(P,P)^{abc} = e(P,P)^z$, that is, $z \equiv abc \pmod{r}$.
- **Bilinear Diffie–Hellman Assumption:** The pairing map does not make these problems computationally easy.
- However, we require the DLP/DHP to be difficult in $G$.
  - If one of $a, b, c$ is known, $e(P,P)^{abc} = e(bP,cP)^a = e(aP,cP)^b = e(aP,bP)^c$ can be computed.
  - If one of $bcP, acP, abP$ is known, $e(P,P)^{abc} = e(aP,bcP) = e(bP,acP) = e(cP,abP)$ can be computed.
- **Example:** Elliptic-curve groups with Weil pairing.
- Extensions possible for $e : G_1 \times G_2 \to G_3$ (**Co-BDHP**, **Co-DBDHP**).

# Identity-Based Encryption (IBE)

- Original concept proposed by Shamir in 1984.
- The first realization proposed in 2001 by Boneh and Franklin.
- The Boneh–Franklin IBE uses pairing.

---

- Conventional encryption and signature schemes (like RSA, DSA) use public-key certificates.
- Every use of a public key requires validating the public key using a certificate from a trusted **Certification Authority (CA)**.
- An identity-based scheme uses a public identity (like e-mail ID) of an entity as the public key, which does not require validation.
- A trusted authority is still needed as a **Key Generation Center (KGC)** or **Public Key Generator (PKG)**.
- The KGC is needed only once during the registration of an entity.

# Boneh–Franklin IBE: Setup Phase

- **Domain parameters**
  - Groups $G, G'$ of prime order $r$
  - A generator $P$ of $G$
  - An efficiently computable bilinear map $e : G \times G \to G'$
- **Keys of PKG**
  - **Master Secret Key (MSK):** $s \in_R \mathbb{Z}_r^*$
  - **Public Key:** $P_{PKG} = sP$.
- **Hash functions**
  - $H_1 : \{0,1\}^* \to G$
  - $H_2 : G' \to \{0,1\}^n$ for some suitable $n$
- $r, G, G', e, P, P_{PKG}, n, H_1, H_2$ are made public
- $s$ is kept secret
- $s$ cannot be retrieved from $P_{PKG} = sP$ (DLP assumption)

# Boneh–Franklin IBE: Key-generation Phase

- The KGC sets up keys for an entity Bob.
  - Bob's public identity: `bob@p.b.cr`
  - Bob's public key: $P_{Bob} = H_1(\texttt{bob@p.b.cr})$.
  - Bob's private key: $D_{Bob} = sP_{Bob}$.
- The KGC transfers $D_{Bob}$ to Bob securely.
- Anybody can compute $P_{Bob}$.
- Bob cannot compute $s$ from $D_{Bob}$ (DLP assumption).

# Boneh–Franklin IBE: Encryption Phase

Alice plans to send an $n$-bit message $M$ to Bob.

- Alice computes Bob's hashed identity $P_{Bob} = H_1(\texttt{bob@p.b.cr}) \in G$.
- Alice computes $g = e(P_{Bob}, P_{PKG}) \in G'$.
- Alice chooses a random element $a \in \mathbb{Z}_r^*$.
- Alice computes the ciphertext $C = (aP, M \oplus H_2(g^a)) \in G \times \{0,1\}^n$.

---

- $a$ is the session secret.
- $H_2(g^a)$ is used as a mask to hide the message.
- Anybody can send messages to Bob.
- No certificates are required.

# Boneh–Franklin IBE: Decryption Phase

Bob plans decrypts a ciphertext $C = (U, V) \in G \times \{0,1\}^n$.

- Bob computes the element $g' = e(D_{Bob}, U) \in G'$.
- Bob computes the mask $H_2(g')$.
- Bob retrieves the message $M = V \oplus H_2(g')$.

---

**Correctness**

- $g' = e(D_{Bob}, U) = e(D_{Bob}, aP) = e(sP_{Bob}, aP) = e(P_{Bob}, P)^{sa} = e(P_{Bob}, sP)^a = e(P_{Bob}, P_{PKG})^a = g^a$

---

**Security**

- An eavesdropper knows $P$, $U = aP$, $P_{Bob} = bP$ and $P_{PKG} = sP$.
- The mask is $e(P,P)^{abs}$.
- Intractability of the BDHP guarantees security against eavesdroppers.
- Alice knows $a$ and can compute the mask.
- Bob knows $bsP$ and can compute the mask.

# SOK Two-Party Key Agreement

- Proposed by Sakai, Ohgishi and Kasahara (2000).
- **Setup phase:** As in Boneh-Franklin IBE $(r, G, G', P, s, P_{PKG}, e, n, H_1)$
- **Key-generation phase:**
  - Alice: Public key $P_{Alice} = H_1(\texttt{alice@p.b.cr})$, private key $D_{Alice} = sP_{Alice}$.
  - Bob: Public key $P_{Bob} = H_1(\texttt{bob@p.b.cr})$, private key $D_{Bob} = sP_{Bob}$.
- **Key-agreement phase:**
  - Alice computes $S_{Alice} = e(D_{Alice}, P_{Bob})$.
  - Bob computes $S_{Bob} = e(P_{Alice}, D_{Bob})$.
- **Correctness:** $S_{Alice} = e(D_{Alice}, P_{Bob}) = e(sP_{Alice}, P_{Bob}) = e(P_{Alice}, P_{Bob})^s = e(P_{Alice}, sP_{Bob}) = e(P_{Alice}, D_{Bob}) = S_{Bob}$.
- **Security:** $P$, $P_{Alice} = aP$, $P_{Bob} = bP$ and $P_{PKG} = sP$ are known to everybody. The task is to compute $e(P,P)^{abs}$. Alice knows $D_{Alice} = asP$ and Bob knows $D_{Bob} = bsP$, so they can compute $e(P,P)^{abs}$. An eavesdropper cannot compute this quantity (BDHP assumption).

# One-Round Three-Party Key Agreement

- Proposed by Joux (2004).
- **Setup phase:** Same as before $(r, G, G', P, e)$.
- **Key-agreement phase:**
  - Alice chooses $a \in_R \mathbb{Z}_r^*$ and broadcasts $aP$ to Bob and Carol.
  - Bob chooses $b \in_R \mathbb{Z}_r^*$ and broadcasts $bP$ to Alice and Carol.
  - Carol chooses $c \in_R \mathbb{Z}_r^*$ and broadcasts $cP$ to Alice and Bob.
  - Alice computes $e(bP, cP)^a = e(P, P)^{abc}$.
  - Bob computes $e(aP, cP)^b = e(P, P)^{abc}$.
  - Carol computes $e(aP, bP)^c = e(P, P)^{abc}$.
- **Security:** A passive eavesdropper knows $P, aP, bP, cP$ only and cannot compute $e(P, P)^{abc}$ (BDHP assumption).

# Paterson's Identity-Based Signatures

- First IBS scheme was proposed and realized by Shamir (1984).
- Many pairing-based IBS schemes are known.
- Paterson's IBS scheme (2002) is an adaptation of ElGamal signatures.
- **Setup phase:** Domain parameters $r, G, G', P, e$ and PKG's keys $s$ and $P_{PKG} = sP$ are as earlier. Hash functions: $H_1 = \{0,1\}^* \to G$, $H_2 : \{0,1\}^* \to \mathbb{Z}_r$ and $H_3 : G \to \mathbb{Z}_r$.
- **Key-generation phase:**
  - Bob's public key is $P_{Bob} = H_1(\texttt{bob@p.b.cr})$
  - Bob's private key is $D_{Bob} = sP_{Bob}$

# Paterson's Identity-Based Signatures (Contd)

- **Signing:** Bob's signature on message $M$ is $(S, T)$, where:

$$\begin{aligned} d' &\in_R \mathbb{Z}_r, \\ S &= d'P, \\ T &= d'^{-1}(H_2(M)P - H_3(S)D_{Bob}). \end{aligned}$$

- **Verification:** Bob's signature $(S, T)$ on $M$ is verified if and only if

$$e(P,P)^{H_2(M)} = e(S,T)e(P_{pub},P_{Bob})^{H_3(S)}.$$

- **Correctness:** $H_2(M)P = d'T + H_3(S)D_{Bob} = d'T + H_3(S)sP_{Bob}$, so

$$\begin{aligned} e(P,P)^{H_2(M)} &= e(P,H_2(M)P) = e(P,d'T + H_3(S)sP_{Bob}) \\ &= e(P,d'T)e(P,H_3(S)sP_{Bob}) = e(d'P,T)e(sP,P_{Bob})^{H_3(S)} \\ &= e(S,T)e(P_{pub},P_{Bob})^{H_3(S)}. \end{aligned}$$

- **Security:** Similar to ElGamal signatures.

# BLS Short Signatures

- Proposed by Boneh, Lynn and Shacham (2004).
- Uses pairing, but not identity-based.
- Smaller signatures than DSA or ECDSA at the same security level.
- **Setup phase:**
  - Groups $G_1, G_2, G_3$ of prime order $r$ (with $G_1 \neq G_2$)
  - Pairing map $e : G_1 \times G_2 \to G_3$
  - A generator $Q$ of $G_2$
  - Hash function $H : \{0,1\}^* \to G_1$
- **Key-generation phase:**
  - Bob's private key: $d \in_R \mathbb{Z}_r$
  - Bob's public key: $Y = dQ \in G_2$
- **Notes:**
  - Does not involve a PKG
  - $G_1 = G_2$ may fail to give same security as DSA

# BLS Short Signatures (Contd)

- **Signing:** Bob's signature on $M$ is $S = dH(M)$.
- **Verification:** Check whether $e(S,Q) = e(H(M),Y)$.
- **Correctness:** $e(S,Q) = e(dH(M),Q) = e(H(M),dQ) = e(H(M),Y)$.
- **Security:**
  - Signature verification is easy, since the Co-DDHP is easy for $G_1, G_2$.
  - Signature forging is difficult, since the Co-DHP is difficult.
  - Any pair of gap Diffie–Hellman (GDH) groups $G_1, G_2$ can be used to implement the BLS scheme.

# References


- Blake, Seroussi and Smart, *Advances in Elliptic Curve Cryptography*, Cambridge, 2005.
- Boneh and Franklin, *Identity Based Encryption from the Weil Pairing*, Crypto 2001.
- Boneh, Lynn and Shacham, *Short Signatures from the Weil Pairing*, Jl of Cryptology, 2004.
- Das, *Computational Number Theory*, CRC Press, 2013.
- Charlap and Robbins, *An Elementary Introduction to Elliptic Curves*, CRD Report, 1988.
- Charlap and Coley, *An Elementary Introduction to Elliptic Curves II*, CCR Report, 1990.
- Cohen, Frey, Avanzi, Doche, Lange, Nguyen and Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, CRC Press, 2006.
- Enge, *Elliptic Curves and Their Applications to Cryptography*, Kluwer, 1999.
- Freeman, Scott and Teske, *A Taxonomy of Pairing-Friendly Elliptic Curves*, Jl of Cryptology, 2010.
- Hankerson, Menezes and Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.
- Joux, *A One-Round Protocol for Tripartite Diffie–Hellman*, ANTS-4, 2004.
- Martin, *Introduction to Identity-Based Encryption*, Artech House, 2008.
- Miller, *The Weil Pairing, and Its Efficient Calculation*, Jl of Cryptology, 2004.
- Paterson, *ID-Based Signatures from Pairings on Elliptic Curves*, Electronics Letters, 2002.
- Sakai, Ohgishi and Kasahara, *Cryptosystems Based on Pairing*, SCIS 2000.

# Thanks for Your Attention!

For future: abhij@cse.iitkgp.ernet.in

## PART 6

# *ECDSA BATCH VERIFICATION*

# ECDSA Revisited: Parameters

- We work over the prime field $\mathbb{F}_q$.
- $E : y^2 = x^3 + ax + b$ is an elliptic curve defined over $\mathbb{F}_q$.
- Assume that $n = |E(\mathbb{F}_q)|$ is prime.
- $P$ is an arbitrary point of order $n$ in $E(\mathbb{F}_q)$.
- $|n - q - 1| \leqslant 2\sqrt{q}$.
- If $n < q$, an integer reduced modulo $n$ may have two modulo $q$ values. The fraction of such integers is very small. So we ignore this.

- **Signer's permanent key**
  - Private key $d \in_R \mathbb{Z}_n$.
  - Public key $Q = dP$.
  - DL assumption: It is infeasible to compute $d$ from $P$ and $Q$.

# ECDSA Signatures Revisited

- **Signature generation**
  - $k \in_R [1, n-1]$ (the session key)
  - $R = kP$
  - $r = x(R) \pmod{n}$
  - $s = k^{-1}(m + dr) \pmod{n}$, where $m = H(M)$
  - $(M, r, s)$ is the signed message

- **Signature verification**
  - $w = s^{-1} \pmod{n}$
  - $u = mw \pmod{n}$
  - $v = rw \pmod{n}$
  - $R = uP + vQ \in E(\mathbb{F}_q)$
  - Accept if and only if $x(R) = r \pmod{n}$

# ECDSA Signatures: Examples

For illustration, we work with an artificially small example.

- $q = 991$
- $E : y^2 = x^3 + x + 23$ defined over $\mathbb{F}_{991}$
- $n = |E(\mathbb{F}_{991})| = 997$
- $P = (1,5) \in E(\mathbb{F}_{991})$ is a point of order 997

- Private key $d = 737$
- Public key $Q = dP = (272,437)$

# ECDSA Signatures: Examples

| Example 1 | Example 2 | Example 3 |
|---|---|---|
| $m_1 = 123$ | $m_2 = 561$ | $m_3 = 288$ |
| Signature generation | | |
| $k_1 = 523$ | $k_2 = 755$ | $k_3 = 593$ |
| $R_1 = k_1P = (476, 617)$ | $R_2 = k_2P = (183, 212)$ | $R_3 = k_3P = (149, 56)$ |
| $r_1 = 476$ | $r_2 = 183$ | $r_3 = 149$ |
| $s_1 = 549$ | $s_2 = 528$ | $s_3 = 569$ |
| Signature verification | | |
| $w_1 = s_1^{-1} = 385$ | $w_2 = s_2^{-1} = 338$ | $w_3 = s_3^{-1} = 198$ |
| $u_1 = m_1w_1 = 496$ | $u_2 = m_2w_2 = 188$ | $u_3 = m_3w_3 = 195$ |
| $v_1 = r_1w_1 = 809$ | $v_2 = r_2w_2 = 40$ | $v_3 = r_3w_3 = 589$ |
| $R_1 = u_1P + v_1Q = (476, 617)$ | $R_2 = u_2P + v_2Q = (183, 212)$ | $R_3 = u_3P + v_3Q = (149, 56)$ |

- Signature generation needs one scalar multiplication.
- Signature verification needs two scalar multiplications.
- Practical improvements:
  - Use double scalar multiplication.
  - $P$ is a system-wide fixed parameter.
  - If $Q$ is fixed too, use double fixed-base scalar multiplication.

# Batch Verification

- Verify multiple signatures together at a time less than the total individual verification time
- Applicable when most of the available signatures are valid
- Useful in resource-constrained and/or real-time systems
- Security issue: One or more invalid signatures in a batch may go unnoticed
- The attacker may inject carefully crafted forged signatures in a batch
- Safeguards needed against such attacks

---

- To verify a batch of $t$ ECDSA signatures $(r_1, s_1)$, $(r_2, s_2)$, $\ldots$, $(r_t, s_t)$.
- $R_i = (x_i, y_i)$, so $r_i = x_i \pmod{n}$. We assume that $x_i = r_i$ for all $i$.
- $Q$ is fixed in a batch but varies across different batches, so precomputations based on $Q$ may be ineffective, particularly for small batches

# The Problem in ECDSA Batch Verification

- The $i$-th verification equation is $R_i = u_i P + v_i Q$.
- These equations can be combined as

$$\sum_{i=1}^{t} R_i = \left(\sum_{i=1}^{t} u_i\right) P + \left(\sum_{i=1}^{t} v_i\right) Q.$$

- This boils down to only *two* scalar multiplication for a batch of any size $t$.
- But how do we compute the left hand side $\sum_{i=1}^{t} R_i$?
- ECDSA signatures present only the $x$-coordinates $x_i = r_i = x(R_i)$.
- ECDSA$^*$: A non-standard variant of ECDSA in which the entire points $R_i$ are included (instead of only $r_i$) in the signatures.
- For ECDSA$^*$, the above algorithm works without any problem.

# A Naive Approach to Solve the Problem

- $y_i^2 = x_i^3 + ax_i + b \pmod{q}$.
- $y_i$ is a modular square root of the right hand side.
- Square-root computations are costly.
- In general, there are two square roots of $x_i^3 + ax_i + b$.
- Try all of the $2^t$ combinations of the *signs* of the square roots. If any of the combinations satisfies the verification equation, accept.
- Checking $2^{t-1}$ combinations actually suffices. There are $2^{t-1}$ possibilities of the $x$-coordinates of $\pm R_1 \pm R_2 \pm \cdots \pm R_t$.
- ECDSA#: A non-standard variant of ECDSA in which an extra bit is appended to an ECDSA signature for identifying the correct square root.
- For ECDSA#, only one of the $2^t$ combinations need to be checked.
- The naive approach is usually the fastest batch-verification algorithm for ECDSA#.

# The Naive Algorithm: Example

- Consider the three signatures $(476, 549), (183, 528), (149, 569)$.
- The square roots of $476^3 + 476 + 23$ are $374, 617$. Take $R_1 = (476, 374)$.
- The square roots of $183^3 + 183 + 23$ are $212, 779$. Take $R_2 = (183, 212)$.
- The square roots of $149^3 + 149 + 23$ are $56, 935$. Take $R_3 = (149, 56)$.
- The right hand side of the verification equation is $(539, 347)$.
- We have the following elliptic-curve sums:
  - $R_1 + R_2 + R_3 = (117, 895)$.
  - $R_1 + R_2 - R_3 = (342, 505)$.
  - $R_1 - R_2 + R_3 = (990, 608)$.
  - $R_1 - R_2 - R_3 = (539, 644) = -(539, 347)$.
- Therefore, $-R_1 + R_2 + R_3 = (539, 347)$, and the batch is verified.

# What about Standard ECDSA Signatures?

- To avoid the time for $t$ modular square-root computations
- Replace this by something faster
- Eliminate the *unknown* $y$-coordinates $y_i = y(R_i)$
- Three elimination possibilities
  - Linearization
  - Algebraic elimination
  - Use of summation polynomials
- The first two methods are based on symbolic manipulations, where $y_1, y_2, \ldots, y_t$ are treated as symbols satisfying $y_i^2 = x_i^3 + ax_i + b \pmod{q}$
- The third method is based on resultant computations
- Analyses and experiments reveal significant practical improvements
- Open question: Can we make elimination faster than $O(2^t)$ time?

# Algorithm S1: Elimination by Linearization

- The verification equation is $\sum_{i=1}^{t} R_i = (\sum_{i=1}^{t} u_i)\, P + (\sum_{i=1}^{t} v_i)\, Q$.
- **Stage 1:** Compute the right hand side numerically by a double scalar multiplication (fixed-base if applicable). Let this point be $(\alpha, \beta)$.
- **Stage 2:** Compute the left hand side symbolically, and express the symbolic sum as a pair $(R_x, R_y)$ of polynomials in $y_1, y_2, \ldots, y_t$. The largest $y_i$-degree in both $R_x$ and $R_y$ is 1 (since $y_i^2$ can be substituted by the explicit value $x_i^3 + ax_i + b$). Moreover, $R_x$ consists non-zero terms of even total degrees, and $R_y$ consists of non-zero terms of odd total degrees.
- **Stage 3:** We have $R_x(y_1, y_2, \ldots, y_t) = \alpha$. By successively squaring this equation or multiplying by even-degree monomials, generate a system of equations, each linear with respect to the even-degree monomials.
- **Stage 4:** Solve the system to get the values of all even-degree monomials.
- **Stage 5:** Use $R_y(y_1, y_2, \ldots, y_t) = \beta$ to solve for individual $y_i$ values.
- **Stage 6:** Check whether $y_i^2 = x_i^3 + ax_i + b \pmod{q}$ for all $i$.

# Algorithm S1: Example

- The verification equation is $(476, y_1) + (183, y_2) + (149, y_3) = (539, 347)$.
- First compute $(h_3, k_3) = (476, y_1) + (183, y_2)$:
  - $\lambda = (y_2 - y_1)/(183 - 476) = 115y_1 + 876y_2$.
  - $\lambda^2 = 342y_1^2 + 307y_1y_2 + 342y_2^2 = 307y_1y_2 + 478$.
  - $h_3 = \lambda^2 - x_1 - x_2 = 307y_1y_2 + 810$.
  - $k_3 = \lambda(x_1 - h_3) - y_1 = 371y_1^2y_2 + 620y_1y_2^2 + 238y_1 + 752y_2 = 580y_1 + 42y_2$.
- Then compute $(h_4, k_4) = (h_3, k_3) + (149, y_3)$:
  - $$\lambda = (y_3 - k_3)/(149 - h_3) = (411y_1 + 949y_2 + y_3)/(684y_1y_2 + 330)$$ $$= (411y_1 + 949y_2 + y_3)(684y_1y_2 - 330)/(684^2y_1^2y_2^2 - 330^2)$$ $$= 987y_1y_2y_3 + 904y_1 + 57y_2 + 906y_3.$$
  - $$h_4 = \lambda^2 - h_3 - x_3 = 16y_1^2y_2^2y_3^2 + 696y_1^2y_2y_3 + 632y_1^2 + 535y_1y2^2y_3$$ $$+ 680y_1y_2y_3^2 + 676y_1y_2 + 916y_1y_3 + 276y_2^2 + 220y_2y_3 + 288y_3^2 + 32$$ $$= 524y_1y_2 + 332y_1y_3 + 58y_2y_3 + 497.$$
  - $k_4 = \lambda(h_3 - h_4) - k_3 = 342y_1y_2y_3 + 227y_1 + 491y_2 + 152y_3$.
- Thus, we have:
  - $524y_1y_2 + 332y_1y_3 + 58y_2y_3 + 497 = 539$.
  - $342y_1y_2y_3 + 227y_1 + 491y_2 + 152y_3 = 347$.

# Algorithm S1: Example (Contd)

- First equation: $524y_1y_2 + 332y_1y_3 + 58y_2y_3 = 82$.
- Generate the second equation:
  - Multiplying by $y_1y_2$ gives $524y_1^2y_2^2 + 332y_1^2y_2y_3 + 58y_1y_2^2y_3 = 82y_1y_2$.
  - This simplifies to $949y_1y_2 + 422y_1y_3 + 572y_2y_3 = 158$.
- Generate the third equation:
  - Multiplying by $y_1y_3$ gives $949y_1^2y_2y_3 + 422y_1^2y_3^3 + 572y_1y_2y_3^2 = 158y_1y_3$.
  - This simplifies to $82y_1y_2 + 833y_1y_3 + 847y_2y_3 = 445$.
- The linearized system is: $\begin{pmatrix} 524 & 332 & 58 \\ 949 & 422 & 572 \\ 82 & 833 & 847 \end{pmatrix} \begin{pmatrix} y_1y_2 \\ y_1y_3 \\ y_2y_3 \end{pmatrix} = \begin{pmatrix} 42 \\ 158 \\ 445 \end{pmatrix}$.
- The solution of this system is: $y_1y_2 = 983$, $y_1y_3 = 858$, $y_2y_3 = 971$.

# Algorithm S1: Example (Contd)

- We also have $342y_1y_2y_3 + 227y_1 + 491y_2 + 152y_3 = 347$.
- Multiply by $y_1$ to get $342y_1^2y_2y_3 + 227y_1^2 + 491y_1y_2 + 152y_1y_3 = 347y_1$.
- Simplification gives $347y_1 = 43$, that is, $y_1 = 617$.
- $y_2 = (y_1y_2)/y_1 = 212$.
- $y_3 = (y_1y_3)/y_1 = 56$.
- Therefore, $y_1^2 = 145$, $y_2^2 = 349$, and $y_3^2 = 163$.
- Moreover, $x_1^3 + x_1 + 23 = 145$, $x_2^3 + x_2 + 23 = 349$, and $x_3^3 + x_3 + 23 = 163$.

# Algorithm S1: Remarks

- This is perhaps not too impressive.
- This is too much computation.
- We have to deal with all even-degree monomials in $y_1, y_2, \ldots, y_t$.
- There are $2^{t-1} - 1$ of them.
- Solving the dense linearized system needs $O(2^{3t})$ field operations.
- But this is the beginning.
- We at least have an understanding of the potentials of symbolic computations.

# Algorithm S1′: Reduction in Monomial Count

- Need to reduce the number of monomials in the linearized system.
- Numerically compute the right hand side of the batch-verification equation. Let this point be $(\alpha, \beta)$.
- Let $\tau = \lceil t/2 \rceil$. Rewrite the verification equation as:

$$\sum_{i=1}^{\tau} R_i = (\alpha, \beta) - \sum_{i=\tau+1}^{t} R_i.$$

- Compute both sides of the rewritten equation symbolically.
- Linearize by successive squaring.
- The variables in the linearized system are all even-degree square-free monomials in $y_1, y_2, \ldots, y_\tau$, and all square-free monomials in $y_{\tau+1}, y_{\tau+2}, \ldots, y_t$.
- Does $O(t^{3/2})$ field operations—still poorer than naive exhaustive search.

## Algorithm S1′: Example

- Rewrite the verification equation as $(476, y_1) + (183, y_2) = (539, 347) + (149, -y_3)$.
- Compute the left hand side as $(h_3, k_3)$ as in S1. We have:
  - $h_3 = 307y_1y_2 + 810$, and
  - $k_3 = 580y_1 + 42y_2$.
- Compute the right hand side as $(h_4, k_4)$:
  - $\lambda = (347 + y3)/(539 - 149) = 836y_3 + 720$.
  - $\lambda^2 = (2 \times 836 \times 720)y_3 + (836^2 y_3^2 + 720^2) = 766y_3 + 741$.
  - $h_4 = \lambda^2 - 539 - 149 = 766y_3 + 53$.
  - $k_4 = l(149 - h_4) + y_3 = 801y_3^2 + 453y_3 + 741 = 453y_3 + 492$.
- Equate the two sides:
  - $307y_1y_2 + 810 = 766y_3 + 53$.
  - $580y_1 + 42y_2 = 453y_3 + 492$.

# Algorithm S1′: Example (Contd)

- Now, we have two variables $y_1 y_2$ and $y_3$.
- First equation: $307 y_1 y_2 + 810 = 766 y_3 + 53$.
- Second equation: Square the first equation to get $849 y_1 y_2 + 768 = 925 y_3 + 645$.
- The linearized system is: $\begin{pmatrix} 307 & 225 \\ 849 & 66 \end{pmatrix} \begin{pmatrix} y_1 y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 234 \\ 868 \end{pmatrix}$.
- Solve this to get $y_1 y_2 = 983$ and $y_3 = 56$.
- We also have $580 y_1 + 42 y_2 = 453 y_3 + 492$. Multiply both sides by $y_1$ to get $(453 y_3 + 492) y_1 = 580 y_1^2 + 42 y_1 y_2$, that is, $y_1 = 617$.
- $y_2 = (y_1 y_2)/y_1 = 212$.

# Algorithm S2: Algebraic Elimination

- The verification equation is $\sum_{i=1}^{t} R_i = (\sum_{i=1}^{t} u_i)\, P + (\sum_{i=1}^{t} v_i)\, Q$.
- **Stage 1:** Compute the right hand side $(\alpha, \beta)$ numerically.
- **Stage 2:** Compute the left hand side symbolically as a pair $(R_x(y_1, y_2, \ldots, y_t), R_y(y_1, y_2, \ldots, y_t))$ of polynomials with square-free monomials.
- **Stage 3:** Set $\phi = R_x - \alpha$. For $i = 1, 2, \ldots, t$, repeat:
  - Write $\phi = u(y_{i+1}, y_{i+2}, \ldots, y_t) + y_i v(y_{i+1}, y_{i+2}, \ldots, y_t)$.
  - Set $\phi$ to $(u - y_i v)\phi = u^2 + y_i^2 v^2$.
  - Substitute all $y_j^2$ for $j = i, i+1, \ldots, t$.
- Accept the batch if and only if $\phi$ is reduced to zero.

# Algorithm S2: Example

- Consider the same example $(476, y_1) + (183, y_2) + (149, y_3) = (539, 347)$.
- As in Algorithm S1, the left hand side has the $x$-coordinate $524y_1y_2 + 332y_1y_3 + 58y_2y_3 + 497$.
- Set $\phi = 524y_1y_2 + 332y_1y_3 + 58y_2y_3 + 497 - 539 = 524y_1y_2 + 332y_1y_3 + 58y_2y_3 + 949 = (524y_2 + 332y_3)y_1 + (58y_2y_3 + 497)$.
- Update $\phi$ to $(524y_2 + 332y_3)^2y_1^2 - (58y_2y_3 + 497)^2 = 600y_2^2y_3^2 + 95y_2^2 + 809y_2y_3 + 623y_3^2 + 218 = 809y_2y_3 + 324$.
- Update $\phi$ to $(809y_3)^2y_2^2 - 324^2 = 0$.

# Algorithm S2′: Faster Variant of S2

- Compute $(\alpha, \beta)$ as in Algorithm S2.
- Let $\tau = \lceil t/2 \rceil$. Rewrite the verification equation as $\sum_{i=1}^{\tau} R_i = (\alpha, \beta) - \sum_{i=\tau+1}^{t} R_i$.
- Compute the two sides of the rewritten equation symbolically. Let $R_x^{(1)}(y_1, y_2, \ldots, y_\tau)$ and $R_x^{(2)}(y_{\tau+1}, y_{\tau+2}, \ldots, y_t)$ be the $x$-coordinates of the two sides.
- Set $\phi = R_x^{(1)} - R_x^{(2)}$.
- Eliminate $y_1, y_2, \ldots, y_t$ from $\phi$ as in Algorithm S2.
- Accept the batch if and only if $\phi$ is reduced to zero.

## Algorithm S2′: Example

- Rewrite the verification equation as

$$(476, y_1) + (183, y_2) = (539, 347) + (149, -y_3).$$

- Symbolic computation gives the $x$-coordinates of the two sides as $307y_1y_2 + 810$ and $766y_3 + 53$.
- Start with

$$\phi = (307y_1y_2 + 810) - (766y_3 + 53) = (307y_2)y_1 + (225y_3 + 757).$$

- Update $\phi$ to

$$(307y_2)^2y_1^2 - (225y_3 + 757)^2 = 215y_2^2 + 907y_3^2 + 254y_3 + 740 = 254y_3 + 641.$$

- Update $\phi$ to $254^2y_3^2 - 641^2 = 0$.

# Algorithms S2 and S2′: Remarks

- Elimination stage is made efficient.
- Much faster than Algorithms S1 and S1′.
- Practical for batch sizes up to six or seven.
- Theoretically poorer than naive exhaustive search by a factor of $t^2$. (Algorithm S1′ is poorer by a factor of $2^{t/2}$.)

# Algorithm SP

- This achieves a running time of $O(2^t)$ field operations.
- Summation polynomials (introduced by Semaev) are recursively defined as:

$$
\begin{aligned}
f_2(x_1, x_2) &= x_1 - x_2, \\
f_3(x_1, x_2, x_3) &= (x_1 - x_2)^2 {x_3}^2 - 2((x_1 + x_2)(x_1 x_2 + a) + 2b)x_3 + \\
&\quad ((x_1 x_2 - a)^2 - 4b(x_1 + x_2)), \\
f_t(x_1, x_2, \ldots, x_t) &= \mathrm{Res}_T(f_{t-k}(x_1, \ldots, x_{t-k-1}, T), f_{k+2}(x_{t-k}, \ldots, x_t, T)) \\
&\quad \text{for } t \geqslant 4 \text{ and for any } k \text{ in the range } 1 \leqslant k \leqslant t-3.
\end{aligned}
$$

- $\mathrm{Res}_T$ is the resultant of two polynomials with respect to the variable $T$.
- Let $x_1, x_2, \ldots, x_t \in \mathbb{F}_q$. Then, $f_t(x_1, x_2, \ldots, x_t) = 0$ if and only if there exist $y_1, y_2, \ldots, y_t \in \overline{\mathbb{F}}_p$ such that $(x_i, y_i)$ lie on the curve for all $i = 1, 2, \ldots, t$, and we have the following sum in the elliptic-curve group $E(\overline{\mathbb{F}}_p)$:

$$(x_1, y_1) + (x_2, y_2) + \cdots + (x_t, y_t) = \mathscr{O}.$$

# Algorithm SP (Contd)

- Write the verification equation as $\sum_{i=1}^{t}(x_i, y_i) + (\alpha, -\beta) = \mathcal{O}$.
- This is true if and only if $f_{t+1}(x_1, x_2, \ldots, x_t, \alpha) = 0$.
- Recursion tree for $t = 5$:

$$
\begin{array}{l}
f_6(x_1, x_2, x_3, x_4, x_5, \alpha) \\
\qquad \rightarrow f_4(x_1, x_2, x_3, T) \\
\qquad\qquad \rightarrow f_3(x_1, x_2, T_1) \\
\qquad\qquad \rightarrow f_3(x_3, T, T_1) \\
\qquad \rightarrow f_4(x_4, x_5, \alpha, T) \\
\qquad\qquad \rightarrow f_3(x_4, x_5, T_2) \\
\qquad\qquad \rightarrow f_3(\alpha, T, T_2)
\end{array}
$$

- Practical for batch sizes up to ten.
- Replace the last resultant calculation by a gcd computation for practical benefits.

# Algorithm SP: Example

- Write the verification equation as

$$(476, y_1) + (183, y_2) + (149, y_3) + (539, -347) = \mathscr{O}.$$

- Compute

$$\begin{aligned} & f_4(476, 183, 149, 539) \\ = \quad & \mathrm{Res}_T(f_3(476, 183, T), f_3(149, 539, T)) \\ = \quad & \mathrm{Res}_T(623T^2 + 569T + 114, 477T^2 + 970T + 658) \\ = \quad & 0. \end{aligned}$$

- In fact, $\gcd(623T^2 + 569T + 114, 477T^2 + 970T + 658) = T + 655.$

## Security Issues

- An attacker capable of forging ECDSA$^*$ (or ECDSA$^\#$) batches can trivially forge ECDSA batches too.
- Suppose that the attacker is capable of forging ECDSA batches that pass our batch-verification algorithms.
- The attacker can uniquely reconstruct the missing $y$-coordinates.
- The naive, S1 and S1$'$ algorithms indeed do so.
- S2 and S2$'$ can be extended to do the same task.
- For small batch sizes, these algorithms are feasible.
- So the attacker can forge ECDSA$^*$ (or ECDSA$^\#$) batches.
- Our algorithms do not compromise security—relative to straightforward ECDSA$^*$ batch verification.

- The security concerns do not end here.

## Need for Randomization

- An attacker can inject $k$ faulty signatures in a batch of size $t$.
- The attacker needs to arrange the following:
  - $R_1 + R_2 + \cdots + R_k = \mathscr{O}$.
  - $m_1 s_1^{-1} + m_2 s_2^{-1} + \cdots + m_k s_k^{-1} = 0 \pmod{n}$.
  - $r_1 s_1^{-1} + r_2 s_2^{-1} + \cdots + r_k s_k^{-1} = 0 \pmod{n}$.
- The effect of these $k$ forged signatures on both sides of the verification equation is zero.
- For example, the attacker may take $m_1 = m_2$, $r_1 = r_2$ and $s_1 = -s_2$. This corresponds to $R_2 = -R_1$.
- In general, the attacker first chooses $R_1, R_2, \ldots, R_k$, and fixes $r_1, r_2, \ldots, r_k$. The attacker then chooses $m_1, m_2, \ldots, m_k$. The attacker finally arranges any solution of the above two modulo $n$ congruences for $s_1^{-1}, s_2^{-1}, \ldots, s_k^{-1}$.
- Randomization destroys the above three relations with high probability.

# What is Randomization?

- Choose random multipliers $\xi_1, \xi_2, \ldots, \xi_t$ during batch verification.
- Now, the attacker must arrange the following three relations *a priori*.
- $\xi_1 R_1 + \xi_2 R_2 + \cdots + \xi_k R_k = \mathcal{O}$.
- $\xi_1 m_1 s_1^{-1} + \xi_2 m_2 s_2^{-1} + \cdots + \xi_k m_k s_k^{-1} = 0 \pmod{n}$.
- $\xi_1 r_1 s_1^{-1} + \xi_2 r_2 s_2^{-1} + \cdots + \xi_k r_k s_k^{-1} = 0 \pmod{n}$.
- If $l$-bit randomizers are used, the probability of a successful attack is $2^{-l}$.
- One can take $l = |q|/2$ since square-root methods for solving the ECDLP imply only this much security.
- Another possibility: $l = 128$.

# Randomization of ECDSA Batches

- The verification equation now modifies to:

$$\sum_{i=1}^{t} \xi_i R_i = \left( \sum_{i=1}^{t} \xi_i u_i \right) P + \left( \sum_{i=1}^{t} \xi_i v_i \right) Q.$$

- The right hand side again poses no difficulty.
- The left hand side appears to be irreparably affected, because only the $x$-coordinates of $R_i$ are available.

- Rescue: Given only $x(R)$ and a multiplier $\xi$, the $x$-coordinate $x(\xi R)$ can be uniquely determined and *efficiently* computed.
- Replace the points $R_i$ by $\xi_i R_i$, and run the batch-verification algorithms. Now, the symbols $y_i$ are $y(\xi_i R_i)$.
- We need good algorithms to compute $x(\xi R)$ from $x(R)$ and $\xi$.

# Montgomery Ladders Revisited

- Suppose that $x(P_1) = h_1$, $x(P_2) = h_2$ and $x(P_1 - P_2) = h_4$ are known.
- We can compute $h_3 = x(P_1 + P_2)$ and $h_5 = x(2P_1)$ as:

$$\begin{aligned} h_3 h_4 (h_1 - h_2)^2 &= (h_1 h_2 - a)^2 - 4b(h_1 + h_2). \\ 4h_5(h_1^3 + ah_1 + b) &= (h_1^2 - a)^2 - 8bh_1. \end{aligned}$$

- Montgomery ladder for computing $x(\xi R)$:
  - Initialize $x(S) := x(R)$ and $x(T) := x(2R)$.
  - For $(i = l-2, l-3, \ldots, 1, 0)$ {
  - If $(\xi_i = 0)$, assign $x(T) := x(T+S)$ and $x(S) := x(2S)$,
  - else assign $x(S) := x(T+S)$ and $x(T) := x(2T)$.
  - }
  - Return $x(S)$
- Loop invariance: $T = S + R$.

# Montgomery Ladders: Example

- Take $R = (476, y)$ and $\xi = 97 = (1100001)_2$.
- Montgomery iterations:

| Bit position | Bit value | $S$ | $T$ | $x(S)$ | $x(T)$ |
|---|---|---|---|---|---|
| 6 | 1 | $R$ | $2R$ | 476 | 467 |
| 5 | 1 | $3R$ | $4R$ | 676 | 544 |
| 4 | 0 | $6R$ | $7R$ | 679 | 441 |
| 3 | 0 | $12R$ | $13R$ | 875 | 447 |
| 2 | 0 | $24R$ | $25R$ | 218 | 200 |
| 1 | 0 | $48R$ | $49R$ | 962 | 740 |
| 0 | 1 | $97R$ | $98R$ | 514 | 140 |

# Seminumeric Randomization

- Let $R = (r, y)$ with $r$ known and $y$ unknown.
- Any non-zero multiple $uR$ of $R$ can be expressed as $(h, ky)$, where $h$ and $k$ are field elements fully determined by $r$ and $u$.
  - For $R$ itself, $h = r$ and $k = 1$.
  - $-(h, ky) = (h, (-k)y)$.
  - Let $P_1 = (h_1, k_1 y)$ and $P_2 = (h_2, k_2 y)$ with $P_1 \neq \pm P_2$. Then, $P_3 = (h_3, k_3 y)$:

$$h_3 = \left(\frac{k_1 - k_2}{h_1 - h_2}\right)^2 (r^3 + ar + b) - h_1 - h_2, \text{ and } k_3 = \left(\frac{k_1 - k_2}{h_1 - h_2}\right)(h_1 - h_3) - k_1.$$

  - We have $P_4 = 2P_1 = (h_4, k_4 y)$:

$$h_4 = \left(\frac{3h_1^2 + a}{2k_1}\right)^2 \left(\frac{1}{r^3 + ar + b}\right) - 2h_1, \text{ and } k_4 = \left(\frac{3h_1^2 + a}{2k_1}\right)\left(\frac{h_1 - h_4}{r^3 + ar + b}\right) - k_1.$$

  - Represent the multiple $(h, ky)$ of $R$ by the pair $(h, k)$ of field elements.

# Seminumeric Randomization: Algorithm

- Precompute the field elements $r^3 + ar + b$ and $(r^3 + ar + b)^{-1}$.
- Initialize $S := (r, 1)$.
- For ($i = l - 2, l - 3, \ldots, 1, 0$) {
- Assign $S := 2S$ using seminumeric doubling.
- If ($\xi_i = 1$), assign $S := S + R$ using seminumeric addition.
- }
- Return $S$ (or the first component of $S$).

- This is slightly slower than scalar multiplication.

# Seminumeric Randomization: Example

- Take $R = (476, y)$ and $\xi = 97 = (1100001)_2$.
- Seminumeric iterations:

| Bit position | Bit value | Operation | $S$ | $h$ | $k$ |
|---|---|---|---|---|---|
| 6 | 1 | Init | $R$ | 476 | 1 |
| 5 | 1 | Double | $2R$ | 467 | 553 |
| | | Add | $3R$ | 676 | 704 |
| 4 | 0 | Double | $6R$ | 679 | 348 |
| 3 | 0 | Double | $12R$ | 875 | 82 |
| 2 | 0 | Double | $24R$ | 218 | 834 |
| 1 | 0 | Double | $48R$ | 962 | 57 |
| 0 | 1 | Double | $96R$ | 692 | 513 |
| | | Add | $97R$ | 514 | 643 |

# Comparison of Randomization Methods

- Montgomery ladders use one doubling and one addition in each iteration.
- The seminumeric method does addition only for one bits.
- No effective windowed variant is known for Montgomery ladders.
- The seminumeric method readily adapts to any windowed variant.
- Montgomery ladders are robust against simple side-channel attacks.
- Neither the Montgomery-ladder method nor the seminumeric method is known to have an effective multiple-scalar-multiplication algorithm.
- The seminumeric method is practically faster than Montgomery ladders except for very small randomizers.

# Overheads of Randomization

- Let SM be the time of one unwindowed full-length scalar multiplication.
- Randomization requires roughly $t$ half-length scalar multiplications.
- 4-NAF seminumeric half-length scalar multiplication takes $\frac{2}{5}$ SM time.
- Double scalar multiplication takes $\frac{7}{6}$ SM time on an average.
- Preparing each fixed-base precomputation table takes $\frac{2}{3}$ SM time.
- Double fixed-base scalar multiplication takes $\frac{1}{2}$ SM time on an average.
- Let BV denote the batch-verification time.
-

| Verification type | Time for verifying $t$ signatures |
|---|---|
| Individual (no fixed-base) | $\left(\frac{7t}{6}\right)\text{SM}$ |
| Individual (fixed-base) | $\left(\frac{4}{3}+\frac{t}{2}\right)\text{SM}$ |
| Batch without randomization | $\left(\frac{7}{6}\right)\text{SM}+\text{BV}$ |
| Batch with randomization | $\left(\frac{2t}{5}+\frac{7}{6}\right)\text{SM}+\text{BV}$ |

# Final Remarks

- For ECDSA$^{\#}$, it is preferable to use arbitrarily scalable naive batch verification, particularly for large batch sizes.
- For standard ECDSA, Algorithm SP with the seminumeric randomization method gives the best practical performance for $t \leqslant 10$.
- If enough memory is available, individual verification using fixed-base double scalar multiplication may outperform batch verification except for small batch sizes.

- It is fairly straightforward to adapt the batch-verification algorithms to other types of curves, like Koblitz curves and Edwards curves.

- It remains unsolved whether batch verification can be done in $o(2^t)$ time.
- No proposed batch-verification algorithm supplies speedup in the case of multiple signers, particularly when randomization is used.

# References for Part 6


- Sabyasachi Karati, Abhijit Das, Dipanwita Roychowdhury, Bhargav Bellur, Debojyoti Bhattacharya and Aravind Iyer, *Batch Verification of ECDSA Signatures*, 5th International Conference on Cryptology in Africa (AfricaCrypt 2012), Lecture Notes in Computer Science #7374, pp 1–18, Jul 10–12, 2012, Ifrane, Morocco.
- Sabyasachi Karati, Abhijit Das, Dipanwita Roychowdhury, Bhargav Bellur, Debojyoti Bhattacharya and Aravind Iyer, *New Algorithms for Batch Verification of Standard ECDSA Signatures*, Journal of Cryptographic Engineering, DOI: 10.1007/s13389-014-0082-x, Volume 4, Issue 4, pp 237–258, Springer-Verlag, November 2014 (online publication dated 26 July 2014).
- Sabyasachi Karati and Abhijit Das, *Faster Batch Verification of Standard ECDSA Signatures Using Summation Polynomials*, 12th International Conference on Applied Cryptography and Network Security (ACNS 2014), Lecture Notes in Computer Science #8479, pp 438–456, Jun 10–13, 2014, Lausanne, Switzerland.
- Sabyasachi Karati, Abhijit Das and Dipanwita Roychowdhury, *Randomized Batch Verification of Standard ECDSA Signatures*, 4th International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE 2014), Lecture Notes in Computer Science #8804, pp 237–255, Oct 18–22, 2014, Pune, India.
- Sabyasachi Karati and Abhijit Das, *Batch Verification of EdDSA Signatures*, 4th International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE 2014), Lecture Notes in Computer Science #8804, pp 256–271, Oct 18–22, 2014, Pune, India.

# Thanks for Your Attention!

For future: abhij@cse.iitkgp.ernet.in