EPFL

Re-Discovering ECM

Semester Project Presentation

Student:
Alessandro Colombo

Supervisor:
Tako Boris Fouotsa

14 June 2023

LASEC

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Motivations

Studying Factorization Methods

- Explore the problem of **integer factorization**
- Investigate variants of ECM with curves of **predefined order:**

➡ ECM with supersingular curves

$$\#E(\mathbb{F}_p) = p + 1$$

➡ ECM with anomalous curves

$$\#E(\mathbb{F}_p) = p$$

RSA Security

Integer factorization problem

- Given a composite integer n, return a prime factor p of n
- Believed to be *hard:*
  - Massive effort to find solutions efficiently
  - No **classical** *polynomial-time* algorithm known
  - Best classical algorithm known NFS (approximatively) $L_n(1/3,\ 2)$
- RSA instance of the problem:
  - ➡ $n = pq$, with p,q ***large*** primes (e.g., 1024/2048 bits)

Special-purpose Factorization Methods

Pollard p-1 and Williams p+1

Theorems on factorization and primality testing

By J. M. POLLARD

(*Mathematics Department, Plessey Telecommunications Research, Taplow Court, Taplow, Maidenhead, Berkshire*)

(*Received* 8 *April* 1974)

1. *Introduction.* This paper is concerned with the problem of obtaining theoretical estimates for the number of arithmetical operations required to factorize a large integer n or test it for primality. One way of making these problems precise uses a multi-tape Turing machine (e.g. (1), although we require a version with an input tape). At the start of the calculation n is written in radix notation on one of the tapes, and the machine is to stop after writing out the factors in radix notation or after writing one of two symbols denoting 'prime' or 'composite'. There are, of course, other definitions which could be used; but the differences between these are unimportant for our purpose.

The exercise of proving theoretical estimates of this kind differs considerably from that of devising practical algorithms for immediate use. On the one hand, an algorithm devised in order to prove a theoretical result may well be very badly suited to practical use, and indeed there is no shortage of difficulties associated with those described here. On the other hand, in one respect one could claim that the theoretical exercise is the more difficult. Contrary to some inventors of practical algorithms we insist that the algorithm shall always terminate in the stated number of operations, and that a proof shall be given that it does so.

To see the force of this, consider the problem of calculating the least positive quadratic non-residue of a large prime. This is easily done in practice by finding the quadratic character (mod p) of 2, 3, ... by means of the Law of Quadratic Reciprocity and stopping at the first non-residue. From the practical point of view (2) the number of operations may be said to be $O(p^{\epsilon})$ or even $O((\log p)^c)$ for some c; but apparently the best theoretical estimate we can give is $O(p^{b+\epsilon})$ with $b = 1/(4\sqrt{e})$, quoting Burgess's estimate (3) for the least positive non-residue. This particular difficulty concerning the distribution of the quadratic residues has occurred in other algorithms, e.g. Berlekamp (4).

A $p + 1$ Method of Factoring

By H. C. Williams

Abstract. Let N have a prime divisor p such that $p + 1$ has only small prime divisors. A method is described which will allow for the determination of p, given N. This method is analogous to the $p - 1$ method of factoring which was described in 1974 by Pollard. The results of testing this method on a large number of composite numbers are also presented.

1. Introduction. In 1974 Pollard [8] introduced a method of factorization which has since been called the $p - 1$ factorization technique. Actually, the test was known to D. N. and D. H. Lehmer many years before this but it was never published because, without a fast computer, it was not possible to determine how effective it would be in practice. For the convenience of the reader we give a brief description of this test.

Suppose N is a number to be factored and that N has a prime factor p such that

$$p = \left(\prod_{i=1}^{k} q_i^{\alpha_i} \right) + 1, \tag{1.1}$$

where q_i is the ith prime and $q_i^{\alpha_i} \leqslant B_1$. Let $q_i^{\beta_i}$ be that power of q_i such that $q_i^{\beta_i} \leqslant B_1$ and $q_i^{\beta_i + 1} > B_1$ and put

$$R = \prod_{i=1}^{k} q_i^{\beta_i}. \tag{1.2}$$

Clearly, $p - 1 \mid R$ and since $a^{p-1} \equiv 1 \pmod{p}$ when $(N, a) = 1$, we have $a^R \equiv 1 \pmod{p}$. Thus, $p \mid (N, a^R - 1)$.

The algorithm now proceeds as follows. For a given B_1 put

$$R = r_1 r_2 r_3 \cdots r_m,$$

(for example, $m = k$, $r_i = q_i^{\alpha_i}$), $a_0 = a$, where $(a, N) = 1$ and define

$$a_i \equiv a_{i-1}^{r_i} \pmod{N} \ (i = 1, 2, 3, \ldots, m).$$

- Efficient under special conditions (e.g., p-1 or p+1 are B-powersmooth)
- Not useful to factor random RSA moduli

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Elliptic Curves

Some properties

- Short Weierstrass (affine) equation:

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

- Abelian group:

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

- Hasse's bound:

$$\#E(\mathbb{F}_p) = p + 1 - t, \text{ with } |t| \leq 2\sqrt{p}$$

- Isomorphism and *j*-invariant:

$$j(E) = \frac{1728 \cdot 4a^3}{4a^3 + 27b^2}$$

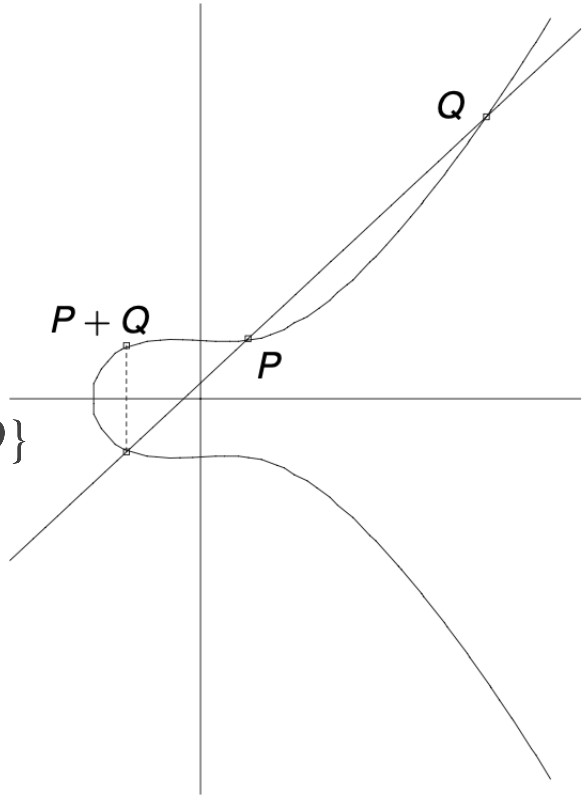

Image source: COM401, slide 358

Elliptic Curve Factorization Method

Principles

- Same principle of Pollard p-1 but:
  - Instead of working with $(\mathbb{F}_p^\times, \times)$ work with $(E(\mathbb{F}_p), +)$

- Sample a random elliptic curve E and $P \in E(\mathbb{Z}_n)$

- If $\#E(\mathbb{F}_p)$ is B-powersmooth, then $\#E(\mathbb{F}_p) \mid m$:
  - $[m]P = O$ over $\mathbb{F}_p$
  - $[m]P \neq O$ over $\mathbb{F}_q$

$$m = \prod_{p_i \leq B} p_i^{\lfloor \log_{p_i} B \rfloor}$$

➡ Some inversion error occurs over $\mathbb{Z}_n$!

Elliptic Curve Factorization Method

The algorithm

Algorithm 1: ECM

input : An RSA Modulus $n = pq$, a bound B, and a time limit w
output: The factorization of n or **failed**

```
Compute m = ∏_{i=1}^{k} p_i^{⌊log_{p_i}(B)⌋} and split it into M_s = [m_1, ..., m_j]
while elapsed time < w do
    Pick a, x_p, y_p randomly from Z_n
    Set b = y_p^2 − x_p^3 − a x_p
    Set E : y^2 = x^3 + ax − b (mod n), and P = (x_p, y_p) ∈ E
    foreach m_i ∈ M_s do
        try
            P = [m_i]P
        if a divisor D is not invertible in Z_n then
            return gcd(D, n)
        end
    end
end
return failed
```

- Repeat the above algorithm until $\#E(F_p)$ is B-powersmooth

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Basic Idea

Turn ECM into a p+1 method

- Supersingular curves:
  - Trace of Frobenius $t \equiv 0 \pmod{p}$
  - From Hasse's bound: $\#E(\mathbb{F}_p) = p + 1$

- Let E be supersigular over $\mathbb{F}_p$ and $P \in E(\mathbb{F}_p)$:
  - If p+1 is B-powersmooth
  - ➡ Then $[m]P = O$ over $\mathbb{F}_p$!

$$m = \prod_{p_i \leq B} p_i^{\lfloor \log_{p_i} B \rfloor}$$

Generating Supersingular Curves

Main ingredients

- ***Endomorphism ring***:

  Set of all the endomorphisms of an elliptic curve E, it is denoted by $End(E)$ (e.g, multiply-by-n map). We always have $\mathbb{Z} \subseteq End(E)$.

- ***Complex Multiplication(CM)***:

  The endomorphism ring of an Elliptic curve E defined over a field K with char(K)=0 is either isomorphic to $\mathbb{Z}$ or to an order O_{-D} in an imaginary quadratic field. In the latter case, we say E has CM by O_{-D}

- ***Hilbert class polynomial:***

  Special polynomial $H_{-D}(x) \in \mathbb{Z}[x]$, its roots are exactly the *j*-invariants of elliptic curves having CM by an order O_{-D} in an imaginary quadratic field $\mathbb{Q}(\sqrt{-d})$.

Generating Supersingular Curves

CM theory

- Let E be an elliptic curve having CM by an order O_{-D} of $Q(\sqrt{-d})$:
  - If the prime p does not *split* in $Q(\sqrt{-d})$
  - ➡ Then E is **supersingular** over $\mathbb{F}_p$
- This condition is **equivalent** to the Legendre symbol[1] $(-D/p) = 0, -1$

- **Idea**: generate a list of curves with CM by finding roots of the Hilbert polynomial $H_{-D}(x)$ for k different values of $-D$,
  - ➡ **At least** one of them will satisfy the condition with: $prob_{succ} = 1 - (0.5)^k$

[1] As p is of cryptographic size, in practice we will never have $(-D/p) = 0$

Generating Supersingular Curves

Problem

We don't know p !

1. Extracting **roots** of $H_{-D}(x) \pmod{n}$ is **hard**:

   Solution: only consider $-D_s$ such that $deg(H_{-D}(x)) = 1$

   ➡ We get a single root j **independent** of n

2. Once $E(j)$ is **fixed**, defining a point $P \in E(j) \pmod{n}$ is **hard**:

   Consider $E(j) : y^2 = x^3 + a(j)x - a(j)$, with $a(j) = \dfrac{27j}{4(1728 - j)} \in \mathbb{Q}$

   ➡ $P = (1,1)$ is always available in $E(j)$!

Generating Supersingular Curves

Candidate discriminants, Hilbert class polynomial and j-invariants

$-D = -d \cdot f^2$	$H_{-D}(x)$	j
$-3 = -3 \cdot 1$	x	0
$-12 = -3 \cdot 4$	$x - 54000$	54000
$-27 = -3 \cdot 9$	$x + 12288000$	-12288000
$-4 = -4 \cdot 1$	$x - 1728$	1728
$-16 = -4 \cdot 4$	$x - 287496$	287496
$-7 = -7 \cdot 1$	$x + 3375$	-3375
$-28 = -7 \cdot 4$	$x - 16581375$	16581375
$-8 = -8 \cdot 1$	$x - 8000$	8000
$-11 = -11 \cdot 1$	$x + 32768$	-32768
$-19 = -19 \cdot 1$	$x + 884736$	-884736
$-43 = -43 \cdot 1$	$x + 884736000$	-884736000
$-67 = -67 \cdot 1$	$x + 147197952000$	-147197952000
$-163 = -163 \cdot 1$	$x + 262537412640768000$	-262537412640768000

- Group discriminant having the same $-d$ (Legendre symbol doesn't change)

ECM p+1

The algorithm

Algorithm 2: ECM with Supersingular Curves

---

input : An RSA modulus $n = pq$ and a bound B
output: The factorization of n or **failed**

Set $J_s = [54000, 287496, -3375, 8000, -32768, -884736, -884736000, -147197952000, -262537412640768000]$
foreach $j \in J_s$ **do**
  Set $a(j) = \frac{27j}{4(1728-j)}$
  Set $E : y^2 = x^3 + a(j)x - a(j)$ and $P = (1,1) \in E$
  Set $\bar{E} : E \pmod{n}$ and $\bar{P} = P \pmod{n}$
  if `ECM` *executed on* $\bar{E}$,$\bar{P}$, *and* B *succeeds* **then**
    return p,q
  end
end
return failed

- When p+1 is B-powersmooth, succeeds with $prob_{succ} = 1 - (0.5)^9 \simeq 0.998$

Performance Comparison

Williams p+1

➡Tested on 1024 bits RSA moduli n=pq such that p+1 is B-powersmooth for $B = 2^{10}$:

```python
def ecm_factor(n, B, curves, anomalous=False, w=30, test=False):
    ks = []
    # for anomalous curves we just multiply the point by n
    if anomalous:
        ks.append(Integer(n))
    # for other curves we use the standard ECM method
    else:
        k = 1
        # populate list ks with all pi^alphai
        for p in prime_range(B):
            ki = p
            k *= p
            while ki < B:
                ki *= p
                if (ki <= B):
                    k *= p
            ks.append(k)
            k = 1
    # try to factor with all the input curves
    for i in range(len(curves)):
        P = curves[i][1]
        if len(P) == 2:
            P.append(1)
        a, b = curves[i][0][0] % n, curves[i][0][1] % n
        if (test):
            print(f"Trying curve {i} ...")
        t = time.time()
        for ki in ks:
            try:
                P = double_add(a, b, P, ki, n)
```

```python
def williams_factor(N, B1, rep = 5):
    Zn = Integers(N)
    p0s = [Integers(2**10).random_element() for i in range(rep)]
    ri = 1
    Ris = []

    tresh = 2**800
    count = 1

    for p in prime_range(B1):
        ri *= p
        tmp = p
        while(tmp < B1):
            tmp *= p
            if(tmp<B1):
                ri *= p

        #we limit the calls to lucas_q1 t0 improve efficiency
        if(ri>tresh):
            count+=1
            p0s[0] = lucas_q1(ri, Zn(p0s[0])) #lucas(rj, lucas(ri,p0)) = lucas(ri*rj, p0)
            d = gcd(p0s[0]-2,N)
            if(d!=0 and d!=1 and d!=N):
                return d
            Ris.append(ri)
            ri=1

    #if the algo failed with the first p0 we try with the others (may have (p0/delta) = 1)
    for i in range(1, rep):
        for j in range(len(Ris)):
            p0s[i] = lucas_q1(Ris[j], Zn(p0s[i]))
```

- Runtime between 30s and 3/4 min
- Factors **some** of the moduli

- Runtime between <1s and 4/5s
- Factors **all** moduli (10 rep)

Performance Comparison

Williams p+1

➡Tested on 1024 bits RSA moduli n=pq such that p+1 is B-powersmooth for $B = 2^{10}$:

```
def ecm_factor(n, B, curves, anomalous=False, w=30, test=False):
    ks = []
    # for anomalous curves we just multiply the point by n
    if anomalous:
        ks.append(Integer(n))
    # for other curves we use the standard ECM method
    else:
        k = 1
        # populate list ks with all pi^alphai
        for p in prime_range(B):
            ki = p
            k *= p
            while ki < B:
                ki *= p
                if (ki <= B):
                    k *= p
            ks.append(k)
            k = 1
    # try to factor with all the input curves
    for i in range(len(curves)):
        P = curves[i][1]
        if len(P) == 2:
            P.append(1)
        a, b = curves[i][0][0] % n, curves[i][0][1] %
        if (test):
            print(f"Trying curve {i} ...")
        t = time.time()
        for ki in ks:
            try:
                P = double_add(a, b, P, ki, n)
```

```
def williams_factor(N, B1, rep = 5):
    Zn = Integers(N)
    p0s = [Integers(2**10).random_element() for i in range(rep)]
    ri = 1
    Ris = []

    tresh = 2**800
    count = 1

    for p in prime_range(B1):
        ri *= p
        tmp = p
        while(tmp < B1):
            tmp *= p
            if(tmp<B1):
                ri *= p

        #we limit the calls to lucas_q1 t0 improve efficiency
        if(ri>tresh):
            count+=1
            p0s[0] = lucas_q1(ri, Zn(p0s[0])) #lucas(rj, lucas(ri,p0)) = lucas(ri*rj, p0)
            d = gcd(p0s[0]-2,N)
            if(d!=0 and d!=1 and d!=N):
                return
            Ris.append(ri)
            ri=1

    #if the algo failed with the first p0 we try with the others (may have (p0/delta) = 1)
    for i in range(1, rep):
        for j in range(len(Ris)):
            p0s[i] = lucas_q1(Ris[j], Zn(p0s[i]))
```

Faster methods already exist!

- Runtime between 30s and 3/4 min
- Factors **some** of the moduli

- Runtime between <1s and 4/5s
- Factors **all** moduli (10 rep)

LASEC

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Basic Idea

ECM with anomalous curves

- Anomalous curves:
  - Trace of Frobenius $t \equiv 1 \pmod{p}$
  - From Hasse's bound: $\#E(\mathbb{F}_p) = p$

- Let E be anomalous over $\mathbb{F}_p$ and $P \in E(\mathbb{F}_p)$:

➡ $[n]P = O$ over $\mathbb{F}_p$!

Generating Anomalous Curves

CM theory

- We define the quadratic twist of an elliptic curve $E/\mathbb{F}_p$ as:
  - $\tilde{\mathbb{E}} : cy^2 = x^3 + ax + b$, with c non-quadratic residue of $\mathbb{F}_p$
- Every $x \in \mathbb{F}_p$ is either x-coordinate of a point in E or in $\tilde{E}$
- Assume:
  - E has complex multiplication by an order O_{-D}
  - $4p = 1 + D(2m+1)^2$, *with* $m \in Z_{\geq 0}$

➡ Then we either have: $\#E(\mathbb{F}_p) = p$, $\#\tilde{E}(\mathbb{F}_p) = p + 2$ or the opposite.

- **Idea**: generate a list of candidate anomalous curves by solving the Hilbert polynomial $H_{-D}(x)$ for different values of $-D$

Generating Anomalous Curves

Same issue, different solutions

We don’t know p !

Two strategies:

1. Restrict to discriminants for which $deg(H_{-D}(x)) = 1$ (as in supersingular case)
2. Go to *number fields* and have **no limitation** on the discriminant:
   - Work with a **symbolic** root *j* of $H_{-D}(x) \pmod{n}$:
     ➡ $E(j)$ is now defined over $\mathbb{Q}(j) = \mathbb{Q}[x]/H_{-D}(x)$
   - Switch from XY to XZ notation:
     - Sample random $x \in \mathbb{Z}_n$
     ➡ Each P = (x : 1) is either on E or on $\tilde{E}$ with 1/2 probability

ECM Anomalous

The algorithm

- Polynomial time algorithm
- Succeeds when $4p = 1 + D(2m+1)^2$
- Induced arithmetic is fast only if $|-D|$ is small
- For random RSA moduli, $(4p-1)/D$ is a square with negligible probability!

Algorithm 3: Anomalous ECM

input : An RSA Modulus $n = pq$, a list H_s of Hilbert polynomials and an iteration bound K

output: The factorization of n or **failed**

```
foreach H_{-D}(x) ∈ H_s do
    Set R[j] = (Z/nZ)[x]/H_{-D}(x)
    Set a(j) = 27j / (4(1728 − j)) ∈ R[j]
    Set Ē : y^2 = x^3 + a(j)x − a(j)
    for i = 1 to K do
        Pick x_P ∈ Z_n randomly
        Set P̄ = (x_P : 1) ∈ Ē
        try
            P̄ = [n]P̄
        if a divisor D is not invertible in Z_n then
            return gcd(D, n)
        end
    end
end
return failed
```

ECM Anomalous

Testing the algorithm

```c
void ECM(uintmax_t B, Curve *C, mpz_t N)
{
    mpz_inits(lamb, a1, a2, NULL);
    uintmax_t ks[DIM], primes[DIM];
    uintmax_t k;
    erathostenes(primes, B);
    for (uintmax_t i = 0; i < DIM; i++)
    {
        uintmax_t ki = primes[i];
        k = primes[i];
        while (ki < B && ki > 0)
        {
            ki *= primes[i];
            if (ki <= B)
                k *= primes[i];
        }
        ks[i] = k;
    }
    printf("%lu, %lu\n", ks[0], ks[1]);

    Point R;
    mpz_init(R.x), mpz_init_set_ui(R.y, 1), R.z = 0;
    Point *P = &(C->point);

    for (uintmax_t j = 0; j < DIM; j++)
    {
        if (j % 2 == 0)
```

```python
def ECM_anomalous(n):
    d = 67
    n = ZZ(n)
    R.<x> = Zmod(n)[]
    S.<J> = R.quotient(hilbert_class_polynomial(-d))  # J is a symbolic roo
    a0, b0 = 27*J*inv(1728-J)/4, -27*J*inv(1728-J)/4  # y^2 = x^3 + a0*x +

    for i in range(1,10):
        P = J.parent().random_element()     # random point
        #print(J.parent())
        print(P)
        try:
            Q = xMUL(a0,b0,n, P)
            continue
        except ZeroDivisionError as e:      # probably found a divisor!
            vs = list(map(ZZ, re.findall('[0-9]+', e.args[0])))

        f = gcd(vs[0],n)
        if 1 < f < n:
            print(f'\x1b[34m{n} = {f} * {n//f}\x1b[0m');
            return f
    return None
```

- C implementation with “**known**” j-invariants
- Runtime in the order of 10^-3 seconds! (1024 bit moduli)

- Sage implementation with **symbolic** j-invariants
- Runtime depending on the **degree** of $H\_D(x)$

(e.g., <1s for $deg(H\_D(x)) = 1$, >30s for $deg(H\_D(x)) = 8$)

- That’s why we needed |-D| to be **small**!

Outline

- Introduction
- Elliptic Curve Factorization Method (ECM)
- ECM with Supersingular Curves
- ECM with Anomalous Curves
- Conclusion

Final Considerations

Our work

- We explored the topic of integer factorization and designed two variants of ECM:
  - ECM with supersingular curves:
    - Transforms ECM into a p+1 method
    - Slow compared to Williams p+1
  - ECM with anomalous curves:
    - Polynomial time algorithm
    - Exposes a restricted class of vulnerable RSA moduli

Final Considerations

Conclusions

- RSA is safe! (at least from us)

- During the last weeks, we discovered that approaches using ECM with anomalous curves already exist[1,2]

LASEC

[1] Q. Cheng, "A new special-purpose factorization algorithm," 2002.

[2] G. Vitto, "Factoring primes to factor moduli: Backdooring and distributed generation of semiprimes," 2021

EPFL

Thank You!

LASEC

EPFL

Questions?

LASEC

Pollard p-1

Principles

- Fermat's little theorem:

$$\forall a \in \mathbb{F}_p^\times : a^{p-1} \equiv 0 \pmod{p}$$

- Idea: use the multiplicative group $\mathbb{F}_p^\times$ to factor n

Let p-1 be B-powersmooth and $m = \prod_{p_i \leq B} p_i^{\lceil \log_{p_i} B \rceil}$

We see that $p - 1 \mid m$

Computing $gcd(a^m, n)$ yields the factorization of n

Quadratic Fields and Orders

Some properties

- Quadratic field:
  - Field with order two over $\mathbb{Q}$
  - Written as $\mathbb{Q}(\sqrt{d})$, where d is a fundamental discriminant
  - Each element $e \in \mathbb{Q}(\sqrt{d})$ expressed as $e = a + b\sqrt{d},$ with $a, b \in \mathbb{Q}$
  - *Imaginary* quadratic field => $d < 0$

- Order of a quadratic field $\mathbb{Q}(\sqrt{d})$:
  - Free Z-module of rank 2 containing an integral base of $\mathbb{Q}(\sqrt{d})$
  - Associated with a discriminant $D = f^2 d$