Lecture 05: Repeated Squaring

Algorithms, modular arithmetic, and elliptic curves

The puzzle: can a large exponent be a short computation?

The task

Compute the residue

$$17^{2020} \bmod 23.$$

Multiplying by 17 one copy at a time takes 2020 steps.

Guiding question

How much work can we save by reusing the powers already computed?

power	residue modulo 23
17^1	17
17^2	13
17^4	8
17^8	18
17^{16}	2

Each new row needs only one squaring.

We will return to the full calculation after explaining how binary digits select the rows.

Roadmap

By the end of this supplement, we will be able to

1. measure an exponent by the number of bits used to write it;
2. derive repeated squaring by halving the exponent;
3. use a reusable table to answer many queries with the same base;
4. compute $17^{2020} \bmod 23$ in the lecture's three ways; and
5. replace squaring by doubling in an elliptic-curve group.

The main algorithmic fact

An exponent with L binary digits can be evaluated using at most $2L$ group operations.

How to use this supplement

Core recap and clarification

- The distributed 17-slide Lecture 05 deck remains the required minimum.
- The first part follows its algorithms and all three approaches to the worked example.

Additional reading

- The second part discusses storage, bit complexity, and implementation issues.
- A worked elliptic-curve example explains the same algorithm in additive notation.

Recall the group and field notation from Lecture 03. New notation is introduced before it is used.

Baseline: [Maj26].

Recall: powers in a group

Let $(G, \circ)$ be a group, let e be its identity, and fix $g \in G$. For a nonnegative integer i, define

$$g^0 = e, \qquad g^i = \underbrace{g \circ \cdots \circ g}_{i \text{ copies}} \quad (i > 0).$$

For finite G, a generator is an element whose powers reach the whole group. In $(\mathbb{Z}_7^*, \times)$:

i	0	1	2	3	4	5	6
$3^i \bmod 7$	1	3	2	6	4	5	1
$2^i \bmod 7$	1	2	4	1	2	4	1

Thus 3 is a generator and 2 is not. Repeated squaring works for *either* element: the base need not generate the group.

A short input can describe an enormous exponent

The lecture's motivating scale

A t-bit prime p satisfies

$$2^{t-1} \leq p < 2^t.$$

For $t = 1000$, the number is enormous, but its representation still has only 1000 bits.

The computational task

Given $g \in \mathbb{Z}_p^*$ and $0 \leq i < p$, compute $g^i \bmod p$.
For $i \geq 1$, its bit length is

$$L = \lfloor \log_2 i \rfloor + 1 \leq t.$$

The case $i = 0$ simply returns e.

Efficiency should depend on the number of input bits, not the numerical value of the exponent.

First attempt: multiply one copy at a time

Exp(i); fixed base g

```
1 prod ← e
2 for j = 1, ..., i do
3     prod ← prod ∘ g
4 return prod
```

The output is right. After j iterations, prod $= g^j$. Thus the returned value is g^i.
The work is too large. The loop uses exactly i group operations. If i has L bits, then

$$2^{L-1} \leq i < 2^L.$$

This is exponential in the input length.

A group operation means one application of $\circ$. We will distinguish this from the cost of integer arithmetic in the additional reading.

The identity that removes one bit

For every $j \geq 0$, regrouping the copies of g gives

Even exponent: $i = 2j$

$$g^{2j} = g^j \circ g^j.$$

Compute the half-sized power, then square it.

Odd exponent: $i = 2j + 1$

$$g^{2j+1} = (g^j \circ g^j) \circ g.$$

Square the half-sized power, then multiply by one more g.

$$18 \longrightarrow 9 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1 \longrightarrow 0.$$

Each step replaces i by $\lfloor i/2 \rfloor$. Associativity justifies the regrouping; commutativity is not required.

Second attempt: save and reuse the smaller answer

Exp(i); fixed g and e

```
1 if i = 0 then
2     return e
3 j ← ⌊i/2⌋
4 α ← Exp(j)
5 if i is even then
6     return α ∘ α
7 else
8     return (α ∘ α) ∘ g
```

One smaller problem

There is only one recursive call at each positive exponent.
Reuse its answer in one squaring, and multiply by g once more when i is odd.

The zero case stops the recursion and returns the identity without a group operation.

Why the recursive algorithm computes g^i

Proof idea. Induct on i; each recursive call has a strictly smaller exponent.

Claim

For every $g \in G$ and every integer $i \geq 0$, the algorithm returns g^i.

Base case. For $i = 0$, the output is $e = g^0$.

Induction step. Let $j = \lfloor i/2 \rfloor < i$. The induction hypothesis gives $\alpha = \mathrm{Exp}(j) = g^j$. Therefore

$$i = 2j: \quad \alpha \circ \alpha = g^{2j} = g^i,$$

$$i = 2j + 1: \quad (\alpha \circ \alpha) \circ g = g^{2j+1} = g^i.$$

Both branches return the required power. □

Boundary check: $\mathrm{Exp}(1) = (e \circ e) \circ g = g$.

Why repeated squaring is efficient

For $i \geq 1$, let L be its bit length and let $\mathrm{wt}(i)$ denote the number of 1's in its binary representation.

Exact count for the displayed recursive code

$$\underbrace{L}_{\text{squarings}} + \underbrace{\mathrm{wt}(i)}_{\text{extra multiplications}} \leq 2L.$$

Each call removes the last bit. Every positive call squares, and a removed 1 bit causes one extra multiplication by g.

The speedup comes from reuse

Compute $\alpha = \mathrm{Exp}(j)$ once. Writing two separate calls to $\mathrm{Exp}(j)$ would duplicate the work instead of reusing it.

Bit operations: parity and division by two

For a nonnegative integer i, & is bitwise AND and $\gg$ is a right shift.

Test the last bit

$$i \,\&\, 1 = \begin{cases} 0, & i \text{ even}, \\ 1, & i \text{ odd}. \end{cases}$$

For example,

$$10010 \,\&\, 00001 = 00000.$$

Remove the last bit

$$i \gg 1 = \lfloor i/2 \rfloor.$$

For example,

$$10010 \gg 1 = 01001,$$

so 18 becomes 9.

Use $j \leftarrow i \gg 1$ in the recursive code. Test the last bit of the *original* i to choose the even or odd branch.

Binary expansion tells us which powers to use

Write the exponent as

$$i = \sum_{j=0}^{L-1} b_j 2^j, \qquad b_j \in \{0, 1\}.$$

The bit b_0 is the least significant bit. For $i = 18$:

bit position j	4	3	2	1	0
place value 2^j	16	8	4	2	1
bit b_j	1	0	0	1	0

$$18 = (10010)_2 = 16 + 2, \qquad g^{18} = g^{16} \circ g^2.$$

Prepare g^{2^j} for each position j. Include that power precisely when $b_j = 1$.

Binary exponentiation: [MvOV97], p. 71.

Final attempt, part 1: global preprocessing

Fix a base g and a capacity $t \geq 1$. The table supports every $0 \leq i < 2^t$.

Build the reusable table

1 $\alpha_0 \leftarrow g, \quad c_0 \leftarrow 1$
2 **for** $j = 1, \ldots, t-1$ **do**
3 $\quad \alpha_j \leftarrow \alpha_{j-1} \circ \alpha_{j-1}$
4 $\quad c_j \leftarrow c_{j-1} \ll 1$

What is stored?

$$\alpha_j = g^{2^j}, \qquad c_j = 2^j.$$

The left shift doubles c_j; squaring doubles the exponent in α_j.

The table costs $t-1$ squarings and stores t group elements. Reuse it while the base and the group stay fixed.

Fixed-base preprocessing: [BS23], Appendix A.2.4(7), p. 1098.

Final attempt, part 2: select the needed powers

Exp(i) using the table

```
1 prod ← e
2 for j = 0, ..., t − 1 do
3     if i < c_j then
4         break
5     if (i & c_j) ≠ 0 then
6         prod ← prod ∘ α_j
7 return prod
```

The mask. Since $c_j = 2^j$, the test asks whether $b_j = 1$.

The update. If so, include $\alpha_j = g^{2^j}$ in the product.

The stopping rule. If $i < 2^j$, this bit and every higher bit are zero.

The exponent i stays unchanged throughout this loop; only the bit position j advances.

Why the table algorithm computes g^i

Proof idea. Track the part of the binary expansion already processed.

Pad with zeros so that $i = \sum_{r=0}^{t-1} b_r 2^r$.

Invariant at the start of position j

$$\text{prod} = g^{\sum_{r=0}^{j-1} b_r 2^r}.$$

Initialization. At $j = 0$, the sum is empty and prod $= e = g^0$.

Update. If $b_j = 0$, do nothing. If $b_j = 1$, multiplication by g^{2^j} adds exactly that term to the exponent.

Termination. All bits have been included. An early break is also valid because every omitted bit is zero. Thus prod $= g^i$. □

Keep preprocessing and per-exponent work separate

Assume $i \geq 1$ has $L \leq t$ bits and $\mathrm{wt}(i)$ nonzero bits.

Method	One-time setup	Group operations for i
First attempt	none	i
Recursive second attempt	none	$L + \mathrm{wt}(i) \leq 2L$
Preprocessed final attempt	$t - 1$ squarings	$\mathrm{wt}(i) \leq L$

One exponent

Choose $t = L$. Including setup, the count is

$$(L - 1) + \mathrm{wt}(i) \leq 2L - 1.$$

Many exponents, one base

Reuse the same table. Each new exponent needs at most L multiplications, plus a scan of its bits.

Compute $17^{2020} \bmod 23$.

1. Square directly

Write 2020 in binary. Compute successive squares and multiply the selected powers.

2. Use $x^p \equiv x$

Since 23 is prime, replace factors 17^{23} by 17.
Then square.

3. Use $x^{p-1} \equiv 1$

Since $23 \nmid 17$, reduce the exponent modulo 22.
Then square.

These are the three approaches from the lecture. The next slides recall the precise hypotheses of the two arithmetic identities.

Solution 1: repeated squaring directly

$$2020 = (11111100100)_2 = 1024 + 512 + 256 + 128 + 64 + 32 + 4.$$

2^j	$17^{2^j} \bmod 23$	Use?
1	17	no
2	13	no
4	8	yes
8	18	no
16	2	no
32	4	yes

2^j	$17^{2^j} \bmod 23$	Use?
64	16	yes
128	3	yes
256	9	yes
512	12	yes
1024	6	yes

$$17^{2020} \equiv 8 \cdot 4 \cdot 16 \cdot 3 \cdot 9 \cdot 12 \cdot 6 \equiv 3 \pmod{23}.$$

Exponent reduction will shorten this calculation without changing the answer.

Solution 2: first use $x^p \equiv x \pmod{p}$

Fermat's identity, including the zero case

If p is prime, then $x^p \equiv x \pmod{p}$ for every integer x.

First reduction. Since $2020 = 23 \cdot 87 + 19$,

$$17^{2020} = (17^{23})^{87} 17^{19} \equiv 17^{87} 17^{19} = 17^{106} \pmod{23}.$$

Second reduction. Since $106 = 23 \cdot 4 + 14$,

$$17^{106} = (17^{23})^{4} 17^{14} \equiv 17^{4} 17^{14} = 17^{18} \pmod{23}.$$

Replacing a factor 17^{23} by 17 removes 22 from the exponent, not 23.

The lecture's homework fact: [MvOV97], Fact 2.127(iii), p. 69.

Solution 3: reduce the exponent modulo $p-1$

Fermat's little theorem: the nonzero hypothesis matters

If p is prime and $p \nmid x$, then $x^{p-1} \equiv 1 \pmod{p}$.

Here $2020 = 22 \cdot 91 + 18$, so

$$17^{2020} = (17^{22})^{91} 17^{18} \equiv 1^{91} 17^{18} = 17^{18} \pmod{23}.$$

The general reduction rule

For $i \geq 0$ and $p \nmid x$, write $i = q(p-1) + r$, where $0 \leq r < p-1$. Then

$$x^i = (x^{p-1})^q x^r \equiv x^r \pmod{p}.$$

The lecture's homework fact: [MvOV97], Fact 2.127(i), p. 69.

Finish the computation: $18 = 16 + 2$

Power	Calculation	Residue
17^1		17
17^2	$17^2 = 289$	13
17^4	$13^2 = 169$	8
17^8	$8^2 = 64$	18
17^{16}	$18^2 = 324$	2

All residues are modulo 23.

Select the powers 16 and 2

$$\begin{aligned} 17^{18} &= 17^{16} \cdot 17^2 \\ &\equiv 2 \cdot 13 \\ &\equiv 26 \\ &\equiv 3 \pmod{23}. \end{aligned}$$

All three approaches agree

$$17^{2020} \equiv 17^{18} \equiv 3 \pmod{23}.$$

Two reductions; two different moduli

Values: reduce modulo p

In $\mathbb{Z}_{23}^*$, store 17^2 as 13 rather than 289.
The group operation is multiplication followed by reduction modulo 23.

Exponents: modulo $p-1$

For a nonzero base modulo 23, reduce exponents modulo 22.
Reducing 2020 modulo 23 gives 19, but

$$17^{19} \equiv 5 \not\equiv 3 \pmod{23}.$$

The nonzero-base hypothesis is essential. For $x = 23$, we have $x^{22} \equiv 0 \pmod{23}$, whereas replacing the exponent by 0 gives $x^0 = 1$.

Core takeaways

1. **Input size:** measure an exponent by its binary length L.
2. **Reuse:** compute the smaller power once, then square it.
3. **Binary selection:** the 1 bits identify the powers to multiply.
4. **Preprocessing:** separate table construction from each new query.
5. **Exponent reduction:** modulo prime p, a nonzero base permits reduction modulo $p - 1$.

The complete worked example

$$2020 \equiv 18 \pmod{22}, \qquad 17^{18} \equiv 2 \cdot 13 \equiv 3 \pmod{23}.$$

From the core material to additional reading

Algorithmic perspective

- a loop that generates powers without storing a table;
- group-operation counts versus bit-operation costs;
- what the execution of an algorithm may reveal.

The same idea in another group

- additive notation and repeated doubling;
- a complete calculation of $[13]P$ on an elliptic curve;
- coordinate reduction versus scalar reduction.

Everything from this point onward is supplementary to the minimum Lecture 05 deck.

A loop without a stored table

Streaming repeated squaring

```
1 prod ← e, a ← g, k ← i
2 while k > 0 do
3     if (k & 1) = 1 then
4         prod ← prod ∘ a
5     k ← k ≫ 1
6     if k > 0 then
7         a ← a ∘ a
8 return prod
```

Idea. Generate $g, g^2, g^4, \ldots$ only as needed.

Storage. Keep only a constant number of group elements; no table or recursive stack is needed.

Work, for $i > 0$.

$$(L-1) + \mathrm{wt}(i) \leq 2L - 1.$$

The last unused squaring is skipped.

Compare [MvOV97], Algorithm 2.143, p. 71.

Proof idea. The completed product times the power still needed stays fixed.

Invariant at each loop entry

$$\text{prod} \circ a^k = g^i.$$

Initialization. $\text{prod} = e$, $a = g$, and $k = i$.

Update. Write $k = 2q + b$ with $b \in \{0, 1\}$. After reading the last bit, $\text{prod}' = \text{prod} \circ a^b$. If $q > 0$, the new base is $a' = a \circ a$, and

$$\text{prod}' \circ (a')^q = \text{prod} \circ a^b \circ a^{2q} = \text{prod} \circ a^k = g^i.$$

If $q = 0$, then $k = b = 1$ and $\text{prod}' = \text{prod} \circ a = g^i$.

Termination. The returned product is g^i. When $i = 0$, the loop is skipped and the output is e.□

Group operations versus bit operations

Let t be the modulus bit length and L the exponent bit length.

One group operation

Let $C(t)$ be the bit cost of multiplying and reducing modulo a t-bit modulus. Classical arithmetic gives

$$C(t) = O(t^2).$$

The exponentiation

Scanning the exponent bits once gives a bit cost of

$$O(L\,C(t) + L).$$

Hence $O(Lt^2)$ with classical arithmetic, or $O(t^3)$ if $L \leq t$.

The $O(L)$ bound counts group operations, not bit operations.

[MvOV97], Table 2.5, p. 72; [BS23], Appendix A.2.4, p. 1098.

The same algorithm in additive notation

In an additive group $(G, +)$, write $[i]P$ for i copies of P added together. The zero multiple is $[0]P = 0_G$, the identity.

Repeated squaring

$$g^{2j} = g^j \circ g^j,$$
$$g^{2j+1} = (g^j \circ g^j) \circ g.$$

Multiply group elements; square the running base.

Repeated doubling

$$[2j]P = [j]P + [j]P,$$
$$[2j+1]P = ([j]P + [j]P) + P.$$

Add group elements; double the running base.

In $(\mathbb{Z}_{23}, +)$, for example, $[18]5 = [16]5 + [2]5 \equiv 11 + 10 \equiv 21 \pmod{23}$.

Elliptic curves: squaring becomes doubling

Work over $\mathbb{F}_{17} = \mathbb{Z}/17\mathbb{Z}$. The curve and the chosen point are

$$E: \; y^2 = x^3 + 2x + 2, \qquad P = (5, 1).$$

The solutions, together with the identity $\mathcal{O}$ (the point at infinity), form a group under elliptic-curve point addition.

The task: compute a scalar multiple

$$[13]P = \underbrace{P + \cdots + P}_{13 \text{ copies}}, \qquad [0]P = \mathcal{O}.$$

Roadmap. Build $P, [2]P, [4]P, [8]P$ by doubling. Then add the terms selected by $13 = (1101)_2$.

Here $+$ is not coordinatewise addition. We recall its formulas next. The point satisfies $1^2 \equiv 5^3 + 2 \cdot 5 + 2 \pmod{17}$.

Elliptic-curve groups: [BS23], §15.2, pp. 617–618.

Elliptic-curve example I: one doubling in full

All arithmetic is modulo 17; division uses a modular inverse.

The doubling rule

For $Q = (x, y)$ with $y \neq 0$, write $[2]Q = (x_D, y_D)$. Then

$$\lambda = (3x^2 + 2)(2y)^{-1},$$
$$x_D = \lambda^2 - 2x,$$
$$y_D = \lambda(x - x_D) - y.$$

Apply it to $P = (5, 1)$

Since $2^{-1} = 9$, we get

$$\lambda \equiv 77 \cdot 9 \equiv 13,$$
$$x_D \equiv 13^2 - 10 \equiv 6,$$
$$y_D \equiv 13(5 - 6) - 1 \equiv 3.$$

Thus $[2]P = (6, 3)$.

Doubling acts on points, not on their individual coordinates. The exceptional rules are $[2]\mathcal{O} = \mathcal{O}$ and $[2](x, 0) = \mathcal{O}$.

Coordinate formulas: [BS23], §15.2, p. 618.

Elliptic-curve example II: build the doubling table

Apply the same rule to the point just obtained. All arithmetic is modulo 17.

Input Q	$(2y)^{-1}$	slope λ	Output $[2]Q$
$P = (5, 1)$	9	$77 \cdot 9 \equiv 13$	$(6, 3)$
$[2]P = (6, 3)$	3	$110 \cdot 3 \equiv 7$	$(3, 1)$
$[4]P = (3, 1)$	9	$29 \cdot 9 \equiv 6$	$(13, 7)$

Coordinates of $[4]P$

$$x_D \equiv 7^2 - 12 \equiv 3,$$
$$y_D \equiv 7(6 - 3) - 3 \equiv 1.$$

Coordinates of $[8]P$

$$x_D \equiv 6^2 - 6 \equiv 13,$$
$$y_D \equiv 6(3 - 13) - 1 \equiv 7.$$

Three doublings; then select $[8]P$, $[4]P$, and P, since $13 = 8 + 4 + 1$.

Elliptic-curve example III: select and add

Adding points with different x-coordinates

For $A = (x_1, y_1)$ and $B = (x_2, y_2)$ with $x_1 \neq x_2$, write $A + B = (x_3, y_3)$. Then

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1},$$
$$x_3 = \lambda^2 - x_1 - x_2, \qquad y_3 = \lambda(x_1 - x_3) - y_1.$$

First compute $[12]P = [8]P + [4]P = (13, 7) + (3, 1)$. Since $3 - 13 \equiv 7$ and $7^{-1} = 5$ modulo 17,

$$\lambda \equiv (1 - 7)(3 - 13)^{-1} \equiv 11 \cdot 5 \equiv 4,$$
$$x_3 \equiv 4^2 - 13 - 3 \equiv 0,$$
$$y_3 \equiv 4(13 - 0) - 7 \equiv 11.$$

Thus $[12]P = (0, 11)$**.** One selected point, P, remains.

Addition formula: [BS23], §15.2, p. 618.

Elliptic-curve example IV: finish the computation

Add the remaining selected point:

$$[13]P = [12]P + P = (0, 11) + (5, 1).$$

Since $5^{-1} = 7$ in $\mathbb{F}_{17}$, the addition rule gives

$$\begin{aligned} \lambda &\equiv (1 - 11)(5 - 0)^{-1} \equiv 7 \cdot 7 \equiv 15, \\ x_3 &\equiv 15^2 - 0 - 5 \equiv 16, \\ y_3 &\equiv 15(0 - 16) - 11 \equiv 4 \pmod{17}. \end{aligned}$$

The double-and-add answer

$$[13](5, 1) = (16, 4).$$

Cost. Three doublings and two additions, starting the selected sum at $[8]P$.

Point check. $4^2 \equiv 16^3 + 2 \cdot 16 + 2 \equiv 16 \pmod{17}$.

Elliptic-curve example V: an independent check

Use a different chain of additions. Recall that $-(x, y) = (x, -y)$ in $\mathbb{F}_{17}$.

Operation	Slope λ	Result
$P + [2]P$	$(3-1)(6-5)^{-1} = 2$	$[3]P = (10, 6)$
$[3]P + [3]P$	$302 \cdot 12^{-1} \equiv 11$	$[6]P = (16, 13)$
$[8]P + [8]P$	$509 \cdot 14^{-1} \equiv 6$	$[16]P = (10, 11)$

Since $[16]P = (10, 11) = -[3]P$,

$$[19]P = [16]P + [3]P = \mathcal{O}.$$

A second route to the same answer

$$[13]P = -[6]P = -(16, 13) = (16, 4).$$

Inverse rule: [BS23], §15.2, p. 618.

The preceding calculation established $[19]P = \mathcal{O}$. Therefore

$$2020 = 19 \cdot 106 + 6 \quad \Longrightarrow \quad [2020]P = [6]P = (16, 13).$$

Coordinates: modulo 17

Point coordinates, slopes, and modular inverses are computed in the field $\mathbb{F}_{17}$.

Scalars: modulo 19

For multiples of this P, the identity $[19]P = \mathcal{O}$ justifies reducing the scalar modulo 19.

The field modulus and the scalar-reduction modulus play different roles, just as 23 and 22 did in the earlier example.

An implementation can reveal the work it performs

A side channel is information exposed by an execution, such as timing or power consumption, rather than just its returned value.

A data-dependent branch

Our teaching code multiplies when a bit is 1 and skips that multiplication when it is 0.
Its operation sequence depends on the exponent's bits.

A secret exponent

An observer who can distinguish these operations may learn information about the exponent.
The earlier proofs establish the returned value, not privacy of the execution.

This is teaching pseudocode, not a side-channel-resistant implementation.

Side-channel context: [BS23], §4.3.2, especially p. 126.

Reading guide

Required baseline

First read the distributed *Lecture 05: Repeated Squaring* deck. Its 17 slides contain the minimum course material.

Further reading

- **Binary exponentiation:** Menezes–van Oorschot–Vanstone, Algorithm 2.143 and Table 2.5, pp. 71–72.
- **Arithmetic and preprocessing:** Boneh–Shoup, Appendix A.2.4, p. 1098.
- **Elliptic-curve point addition:** Boneh–Shoup, §15.2, pp. 617–618.

Page numbers refer to the printed pages in the cited editions.

References

[BS23] Dan Boneh and Victor Shoup.
A Graduate Course in Applied Cryptography.
Author-maintained textbook, 2023.
Version 0.6, January 2023.
URL: https://toc.cryptobook.us/book.pdf.

[Maj26] **Hemanta K. Maji.**
Lecture 05: Repeated squaring.
CS 35500, Purdue University. Instructor's lecture slides, 2026.
Distributed lecture deck, 17 slides.

[MvOV97] **Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone.**
Handbook of Applied Cryptography.
CRC Press, 1997.
Chapter 2, Mathematical Background.
URL: https://cacr.uwaterloo.ca/hac/about/chap2.pdf.