# BERNSTEIN-YANG INVERSION, SAFEGCD

## HIGHSPEED CONSTANT-TIME INVERTER

**Cesare Huang**

Institute of Information Science, Academia Sinica

October 7, 2025

# Outline


**1 Introduction . . . . . 2**

**2 Definitions and Basic Construction . . . . . 3**

**3 A Naïve Inversion Algorithm . . . . . 7**

3.1 Bounds on Number of Divsteps . . . . . 10

3.2 Updating the Transition Matrix . . . . . 11

**4 Jumping: A Divide-and-Conquer Strategy for divsteps . . . . . 14**

**5 Advanced Optimizations . . . . . 15**

**6 Benchmark . . . . . 17**

# Introduction

- This is a breif introduction on safegcd. A constant-time efficient gcd algorithm.
- This work builds on [BY19].

### Definition 1 (Divstep)

*Given $f, g$ and $\delta$, one division step (divstep) updates the triple*

$$(\delta, f, g) \longmapsto (\delta', f', g')$$

*according to the following rule:*

$$(\delta', f', g') = \begin{cases} (1-\delta,\ g,\ (g-f)/2), & \text{if } g \text{ is odd and } \delta > 0, \\ (1+\delta,\ f,\ (g+(g \bmod 2)f)/2), & \text{otherwise} \end{cases}$$

*We will work with initial value of $\delta = \frac{1}{2}$, so when writing programs, what actually performed is*

$$(\delta', f', g') = \begin{cases} (2-\delta,\ g,\ (g-f)/2), & \text{if } g \text{ is odd and } \delta > 0, \\ (2+\delta,\ f,\ (g+(g \bmod 2)f)/2), & \text{otherwise} \end{cases}$$

*with initlial value of $\delta = 1$.*

## Example 1

Traces of divsteps:

**Example a**

$f = 7$, $g = 3$, $\delta = 1$

| Step | $\delta$ | $f$ | $g$ |
|---|---|---|---|
| 0 | 1 | 7 | 3 |
| 1 | 1 | 3 | -2 |
| 2 | 3 | 3 | -1 |
| 3 | -1 | -1 | -2 |
| 4 | 1 | -1 | -1 |
| 5 | 1 | -1 | 0 |
| 6 | 3 | -1 | 0 |
| 7 | 5 | -1 | 0 |
| 8 | 7 | -1 | 0 |
| 9 | 9 | -1 | 0 |
| 10 | 11 | -1 | 0 |

$\Rightarrow f_{\text{final}} = -1$

**Example b**

$f = 9$, $g = 5$, $\delta = 1$

| Step | $\delta$ | $f$ | $g$ |
|---|---|---|---|
| 0 | 1 | 9 | 5 |
| 1 | 3 | 5 | -2 |
| 2 | 5 | 5 | -1 |
| 3 | 7 | -1 | -3 |
| 4 | -5 | -1 | -2 |
| 5 | -3 | -1 | -1 |
| 6 | -1 | -1 | 0 |
| 7 | 1 | -1 | 0 |
| 8 | 3 | -1 | 0 |
| 9 | 5 | -1 | 0 |
| 10 | 7 | -1 | 0 |

$\Rightarrow f_{\text{final}} = -1$

**Example c**

$f = 19$, $g = 9$, $\delta = 1$

| Step | $\delta$ | $f$ | $g$ |
|---|---|---|---|
| 0 | 1 | 19 | 9 |
| 1 | 1 | 9 | -5 |
| 2 | 1 | -5 | -7 |
| 3 | 1 | -7 | -1 |
| 4 | 1 | -1 | 3 |
| 5 | 1 | 3 | 2 |
| 6 | 3 | 3 | 1 |
| 7 | -1 | 1 | -1 |
| 8 | 1 | 1 | 0 |
| 9 | 3 | 1 | 0 |
| 10 | 5 | 1 | 0 |

$\Rightarrow f_{\text{final}} = 1$

All examples terminate with $g = 0$ and $f = \pm 1$.

## Definition 2

*Transition matrices describe how the pair* $(f, g)$ *evolves during* **Divstep***(s).*

$$\begin{pmatrix} f' \\ g' \end{pmatrix} = T \begin{pmatrix} f \\ g \end{pmatrix}$$

*where*

$$T = \begin{cases} \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}, & g \text{ odd and } \delta > 0, \\ \begin{bmatrix} 1 & 0 \\ \frac{g \bmod 2}{2} & \frac{1}{2} \end{bmatrix}, & g \text{ otherwise.} \end{cases}$$

*Let* $(\delta_n, f_n, g_n) =$ **Divstep**$^n(\delta, f, g)$*, then we will use the following notations:*

$$\begin{bmatrix} f_{n+1} \\ g_{n+1} \end{bmatrix} = T_{n+1:n} \begin{bmatrix} f_n \\ g_n \end{bmatrix}, \quad \begin{bmatrix} f_m \\ g_m \end{bmatrix} = T_{m:n} \begin{bmatrix} f_n \\ g_n \end{bmatrix}, \quad T_{m:n} = T_{m:m-1} T_{m-1:m-2} \cdots T_{n+1:n}, \quad A_m = T_{m:0}$$

## Example 2

Transition and accumulated matrices for Example a ($f = 7, g = 3, \delta = 1$).

| Step | $\delta$ | $f$ | $g$ |
|---|---|---|---|
| 0 | 1 | 7 | 3 |
| 1 | 1 | 3 | -2 |
| 2 | 3 | 3 | -1 |
| 3 | -1 | -1 | -2 |
| 4 | 1 | -1 | -1 |
| 5 | 1 | -1 | 0 |

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{16} & -\frac{3}{16} \\ \frac{3}{32} & -\frac{7}{32} \end{bmatrix} \begin{bmatrix} 7 \\ 3 \end{bmatrix}$$

**Step 1.** $\begin{bmatrix} 3 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 7 \\ 3 \end{bmatrix}$, $A_1 = T_{1:0} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$;

**Step 2.** $\begin{bmatrix} 3 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 3 \\ -2 \end{bmatrix}$, $A_2 = T_{2:1}A_1 = \begin{bmatrix} 0 & 1 \\ -\frac{1}{4} & \frac{1}{4} \end{bmatrix}$;

**Step 3.** $\begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 3 \\ -1 \end{bmatrix}$, $A_3 = T_{3:2}A_2 = \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{8} & -\frac{3}{8} \end{bmatrix}$;

**Step 4.** $\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} -1 \\ -2 \end{bmatrix}$, $A_4 = T_{4:3}A_3 = \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{16} & -\frac{3}{16} \end{bmatrix}$;

**Step 5.** $\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $A_5 = T_{5:4}A_4 = \begin{bmatrix} -\frac{1}{16} & -\frac{3}{16} \\ \frac{3}{32} & -\frac{7}{32} \end{bmatrix}$.

# A Naïve Inversion Algorithm

1. From the example in the last page, we get:

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{16} & -\frac{3}{16} \\ \frac{3}{32} & -\frac{7}{32} \end{bmatrix} \begin{bmatrix} 7 \\ 3 \end{bmatrix}$$

2. Extract the first row:

$$-\frac{1}{16} \cdot 7 + \frac{-3}{16} \cdot 3 = -1.$$

3. Multiply both sides by 16:

$$16 \cdot -\frac{1}{16} \cdot 7 + 16 \cdot \frac{-3}{16} \cdot 3 = 16 \cdot (-1).$$

4. Reduce modulo 7:

$$16 \cdot \frac{-3}{16} \cdot 3 = 16 \cdot (-1) \pmod 7.$$

5. Hence

$$(16^{-1} \bmod 7) \cdot 16 \cdot \frac{-3}{16} \cdot 3 = (-1) \pmod 7.$$

6. The rational coefficient

$$-\frac{3}{16}$$

encodes the modular-inverse relationship:

$$(16^{-1} \bmod 7) \cdot 16 \cdot \frac{-3}{16} = -(3^{-1} \bmod 7).$$

# A Naïve Inversion Algorithm

For the case $(\delta = 1, f = 9, g = 5)$, we have

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{3}{32} & -\frac{1}{32} \\ * & * \end{bmatrix} \begin{bmatrix} 9 \\ 5 \end{bmatrix}$$

Then

$$-\frac{3}{32} \cdot 9 + \frac{-1}{32} \cdot 5 = -1.$$

Multiply both sides by 32, reduce modulo 9 and multiply by $32^{-1} \bmod 9$, we get

$$(32^{-1} \bmod 9) \cdot 32 \cdot \frac{-1}{32} \cdot 5 = -1 \pmod 9.$$

So

$$(32^{-1} \bmod 9) \cdot 32 \cdot \frac{-1}{32} = -(5^{-1} \bmod 9).$$

For the case $(\delta = 1, f = 19, g = 9)$, we have

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{64} & \frac{5}{64} \\ * & * \end{bmatrix} \begin{bmatrix} 19 \\ 9 \end{bmatrix}$$

Then

$$\frac{1}{64} \cdot 19 + \frac{5}{64} \cdot 9 = 1.$$

Multiply both sides by 64, reduce modulo 19 and multiply by $64^{-1} \bmod 19$, we get

$$(64^{-1} \bmod 19) \cdot 64 \cdot \frac{5}{64} \cdot 9 = 1 \pmod{19}.$$

So

$$(64^{-1} \bmod 19) \cdot 64 \cdot \frac{5}{64} = (9^{-1} \bmod 19).$$

# A Naïve Inversion Algorithm

We can then write down a naïve algorithm to compute the modular inverse:

1. Initialize $(\delta, f, g) = (1, p, x)$, where $p$ is the modulus and $x$ is the integer to invert.
2. Perform enough **divstep**s such that $f_n = \pm 1$ and $g_n = 0$. Also accmumulate the transition matrix $A_n$ such that
$$\begin{bmatrix} f_n \\ g_n \end{bmatrix} = A_n \begin{bmatrix} p \\ x \end{bmatrix} = \begin{bmatrix} u & v \\ r & s \end{bmatrix} \begin{bmatrix} p \\ x \end{bmatrix}$$
Note that it is guaranteed that $u, v, r, s$ are multiples of $2^{-n}$.
3. Extract the first row of $A_n$:
$$u \cdot p + v \cdot x = f_n = \pm 1.$$
4. Multiply both sides by $2^n$, take modulo $p$, and multiply by $2^{-n} \bmod p$:
$$(2^{-n} \bmod p) \cdot 2^n \cdot v \cdot x = f_n = \pm 1 \bmod p.$$
5. Return $\pm(2^{-n} \bmod p) \cdot 2^n \cdot v$ as the inverse of $x \bmod p$.

**Theorem 1 ([BY19], [Sip25])**

*If $M \geq 2301079$, the for all $0 \leq g \leq f \leq M$ start with $\delta = 1$, the number of divstep iterations is bounded by*

$$m = \left\lfloor \frac{3787(\log_2 M) + 2166}{1644} \right\rfloor$$

*If $(\delta_n, f_n, g_n) = \textbf{Divstep}^n(\delta_0, f, g)$ and $n \geq m$, then $g_n = 0$ and $f_n = \pm \gcd(f, g)$.*
For the prime $p = 2^{255} - 19$, if we want to invert any $x$ such that $0 < x < p$, we have to perform $m = 588$ divsteps. We will then focus on the implementation of 25519 inverter. Other primes common in cryptographic engineering and the numbers of divsteps required are:

| Name | Prime | Divsteps needed |
|---|---|---|
| Curve25519 | $2^{255} - 19$ | 588 |
| Ed25519 | $2^{255} - 19$ | 588 |
| secp256k1 | $2^{256} - 2^{32} - 977$ | 591 |
| P-256 (secp256r1) | $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ | 591 |
| P-384 (secp384r1) | $2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$ | 885 |
| P-521 (secp521r1) | $2^{521} - 1$ | 1201 |
| SM2 (sm2p256v1) | $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 + (2^{32} - 1)$ | 591 |

# A Naïve Inversion Algorithm

## Updating the Transition Matrix

Here, we make a crucial observation on computing the accumulation transition matrix $A_n$.

1. Accumulation matrix $A$ satisfies the following relations:

$$\begin{bmatrix} f_n \\ g_n \end{bmatrix} = \underbrace{\begin{bmatrix} u_n & v_n \\ r_n & s_n \end{bmatrix}}_{=A_n} \begin{bmatrix} f_0 \\ g_0 \end{bmatrix}, \quad \begin{bmatrix} f_{n+1} \\ g_{n+1} \end{bmatrix} = \underbrace{\begin{bmatrix} u_{n+1} & v_{n+1} \\ r_{n+1} & s_{n+1} \end{bmatrix}}_{=A_{n+1}} \begin{bmatrix} f_0 \\ g_0 \end{bmatrix}$$

2. We need to update from $A_n$ to $A_{n+1}$ during each divstep.
3. Perform the operations done on $(f_n, g_n)$ to the pairs $(u_n, r_n)$ and $(v_n, s_n)$ will complete the update. Say, in this step, the single-step transition matrix is

$$\begin{bmatrix} f_{n+1} \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} f_n \\ g_n \end{bmatrix}$$

Then

$$\begin{bmatrix} u_{n+1} \\ r_{n+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} u_n \\ r_n \end{bmatrix}, \begin{bmatrix} v_{n+1} \\ s_{n+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} v_n \\ s_n \end{bmatrix}$$

will get you a correct accumulation matrix.

# A Naïve Inversion Algorithm

## Updating the Transition Matrix

- if $\delta > 0$ and $g$ is odd, then

$$\begin{aligned}
\delta' &= 2 - \delta \\
f' &= g & u' &= r & v' &= s \\
g' &= (g - f)/2 & r' &= (r - u)/2 & s' &= (s - v)/2
\end{aligned}$$

- othwewise

$$\begin{aligned}
\delta' &= 2 + \delta \\
f' &= f & u' &= u & v' &= v \\
g' &= (g + (g \bmod 2)f)/2 & r' &= (r + (g \bmod 2)u)/2 & s' &= (s + (g \bmod 2)v)/2
\end{aligned}$$

# A Naïve Inversion Algorithm

## Updating the Transition Matrix

But it remains a problem that, currently, our accumulation matrix are in the rational number entries:

$$\begin{bmatrix} 0 & 1 \\ -\frac{1}{4} & \frac{1}{4} \end{bmatrix}, \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{8} & -\frac{3}{8} \end{bmatrix}, \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{16} & -\frac{3}{16} \end{bmatrix}, \ldots$$

which is not friendly to the computer arithmetic.
A remedy: If we plan to do say, 20 disvsteps, we initialize the accumulation matrix as

$$\begin{bmatrix} (2^{20}) & 0 \\ 0 & (2^{20}) \end{bmatrix}$$

instead of identity. Then after 20 divsteps, we multiply the matrix by $(2^{-20} \bmod p)$ once. We can do the shifting in place of divided by 2, constant-time and efficient.
However, in 25519 inversion, we need to perform 588 divsteps. That means we have to initialize the matrix as $2^{588}I$, and this is, not plausible since a shifting costs 9 to 10 actual shinfting instructions in assembly. We need another divide-and-conquer strategy for this.

## JUMPING: A DIVIDE-AND-CONQUER STRATEGY FOR DIVSTEPS

- Idea: Deciding the branching of first $k$ steps requires only lowest $k$ bits of $f, g$.
- Accumulation matrices are multiplicative:

$$\begin{aligned} A_{600} &= T_{600:0} \\ &= T_{600:300}\, T_{300:0} \\ &= T_{600:540}\, T_{540:480} \cdots T_{60:0} \end{aligned}$$

- Our new algorithm can be:
  1. Initialize $(\delta_0, f_0, g_0)$ and $A_0 = I$
  2. Take the lowest 60 bits of $f_0, g_0$, store them in the $\bar{f}, \bar{g}$, respectively.
  3. Perform 60 divsteps on $\bar{f}, \bar{g}$ and get the transition matrix $(2^{60})T_{60:0}$ of 60 steps.
  4. Compute

$$A_{60} = T_{60:0} A_0, \quad \begin{bmatrix} f_{60} \\ g_{60} \end{bmatrix} = T_{60:0} \begin{bmatrix} f_0 \\ g_0 \end{bmatrix}$$

     (the operation on the left could be done with Montgomery Multiplication of $(2^{60})T_{60:0}$ and $A_0$, and that will automatically cancel out $2^{60}$ factor of $T_{60:0}$ if taking $R = 2^{60}$. Similarly, the operation on the right could be done with long multiplication of $(2^{60})T_{60:0}$ and $(f_0, g_0)$, and the lowest 60 bits of the linear combination result will be zero. Can just omit them and cancel out the $2^{60}$ factor.)
  5. Back to the step 2, replacing $f_0, g_0$ by $f_{60}, g_{60}$.

## Advanced Optimizations

- As mentioned above, we have to update the accumulate matrix in each divstep according to
  - if $\delta > 0$ and $g$ is odd, then

$$f' = g \qquad u' = r \qquad v' = s$$
$$g' = (g - f)/2 \qquad r' = (r - u)/2 \qquad s' = (s - v)/2$$

  - othwewise

$$f' = f \qquad u' = u \qquad v' = v$$
$$g' = (g + (g \bmod 2)f)/2 \qquad r' = (r + (g \bmod 2)u)/2 \qquad s' = (s + (g \bmod 2)v)/2$$

- Note that, $f$, $u$ and $v$ are always synchronized, and $g$, $r$, and $s$ are similalrly always synchronized.
- One can obviouly do these operations in vector instruction. But there is no conditional flag that is useful for us in vector register.
- What we have done is to squeeze $f, u, v$ into one integer register and $g, r, s$ into another integer register. Do the divstep by using conditional flags in intger registers.
- Concretely, let

$$fuv = (2^{42}) \cdot u + (2^{21}) \cdot v + f$$
$$grs = (2^{42}) \cdot r + (2^{21}) \cdot s + g$$

and do divstep on $(fuv, grs)$

# Advanced Optimizations

- Concretely, let

$$fuv = (2^{42}) \cdot u + (2^{21}) \cdot v + f$$
$$grs = (2^{42}) \cdot r + (2^{21}) \cdot s + g$$

  and do divstep on $(fuv, grs)$
- After 20 divsteps are done on $fuv, grs$, we can extract the 20-divstep transition matrix $u, v, r, s$ out.
- Accumulate 3 20-divstep transition matrix, then we get a 60-divstep transition matrix by matrix multiplication.
- Send the 60-divstep transition matrix to the vector registers and then perform big number arithmetic (Montgomery and long products)
- Once $f_0 g_0$ are updated, send them back to the scalar registers, and start next 60-divstep in scalar registers.
- Other vector instructions can be interleaved with scalar registers.

# BENCHMARK

1. Raspbery pi 5 (cortex-a76 core):
   1.1 Fermat Little Theorem: 29500 cycles
   1.2 Safegcd (before SLOTHY): 5200 cycles
   1.3 Safegcd (after SLOTHY): 3905 cycles
2. Mac M1 chip:
   2.1 Fermat Little Theorem: 8200 cycles
   2.2 Safegcd (after SLOTHY): 3200 cycles

# References I


[BY19] Daniel J. Bernstein and Bo-Yin Yang. “Fast constant-time gcd computation and modular inversion”. In: *Cryptographic Hardware and Embedded Systems – CHES 2019*. Ed. by Steven Galbraith and Shiho Moriai. Vol. 11603. Lecture Notes in Computer Science. Springer, 2019, pp. 340–368. DOI: 10.1007/978-3-030-26648-3_16.

[Sip25] Sipa. *safegcd-bounds*. `https://github.com/sipa/safegcd-bounds`. GitHub repository. 2025.