New Diffie–Hellman Speed Records

Chitchanok Chuengsatiansup

Technische Universiteit Eindhoven

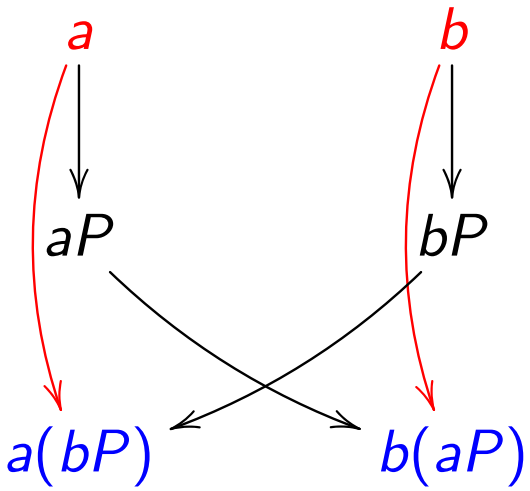
March 23rd, 2015

Diffie–Hellman Key Exchange

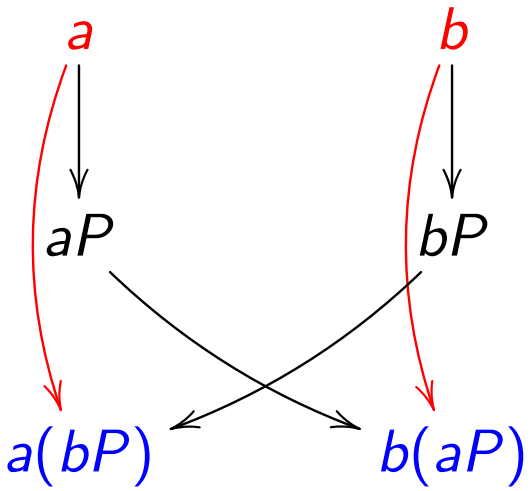
a b

Diffie–Hellman Key Exchange

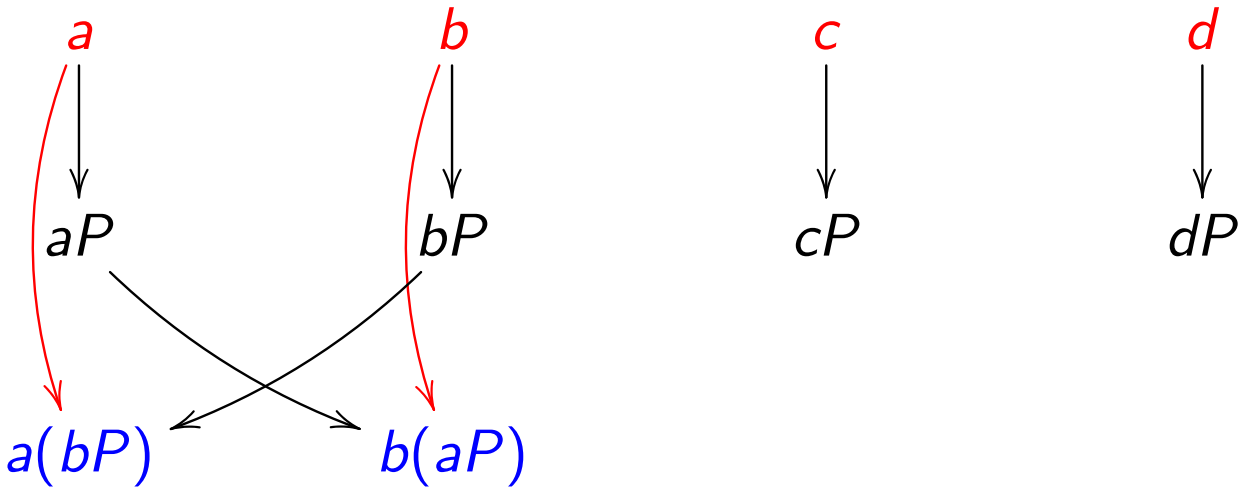
Diffie–Hellman Key Exchange



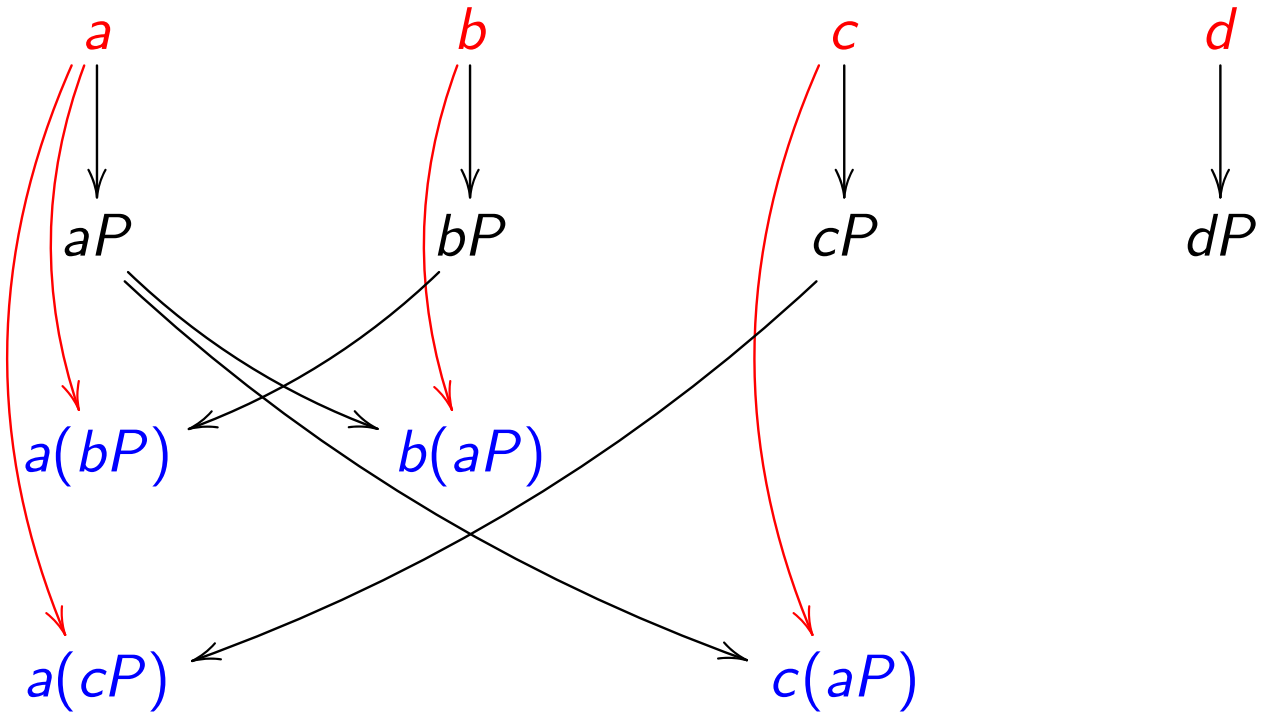
Diffie–Hellman Key Exchange



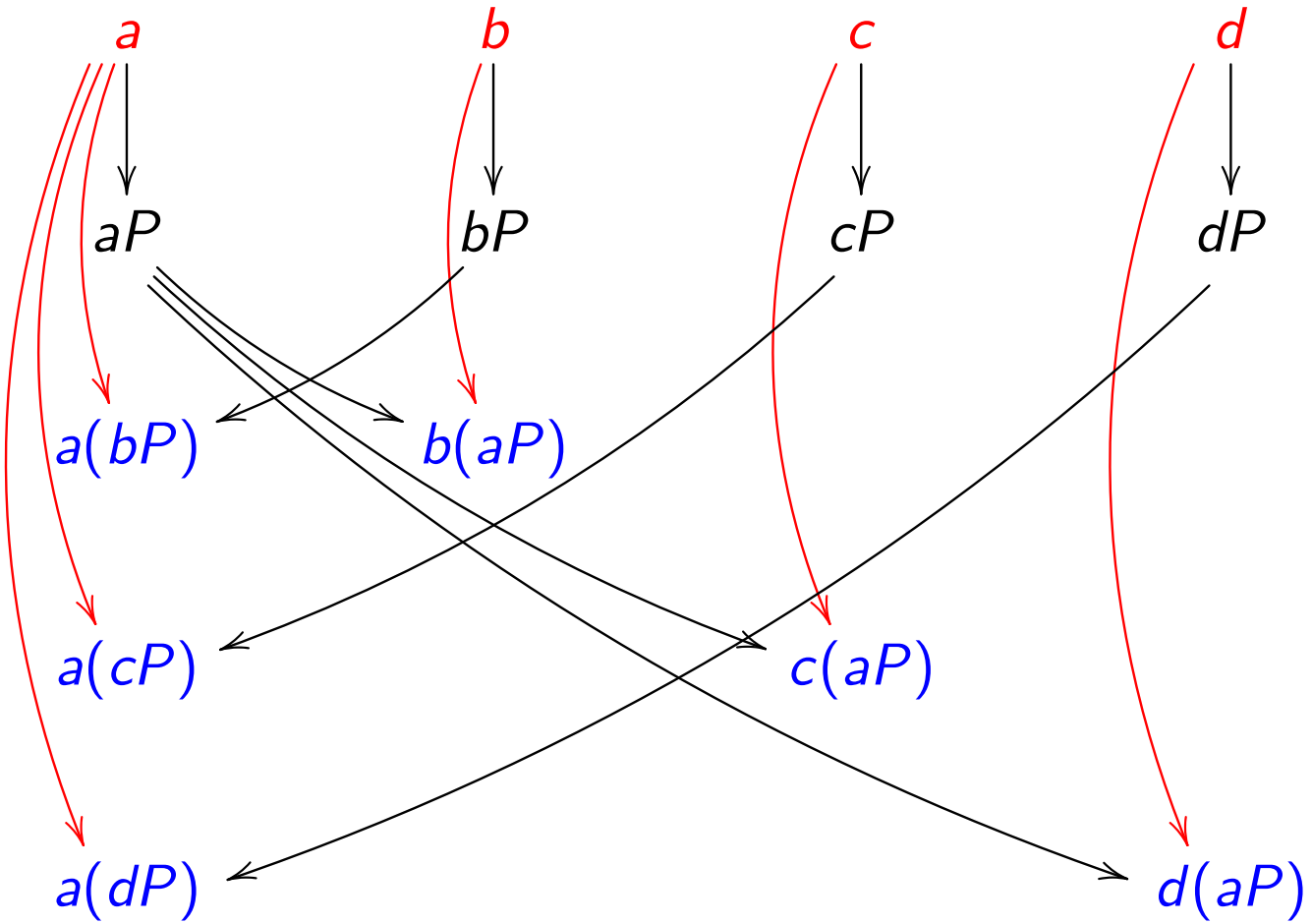
Diffie–Hellman Key Exchange



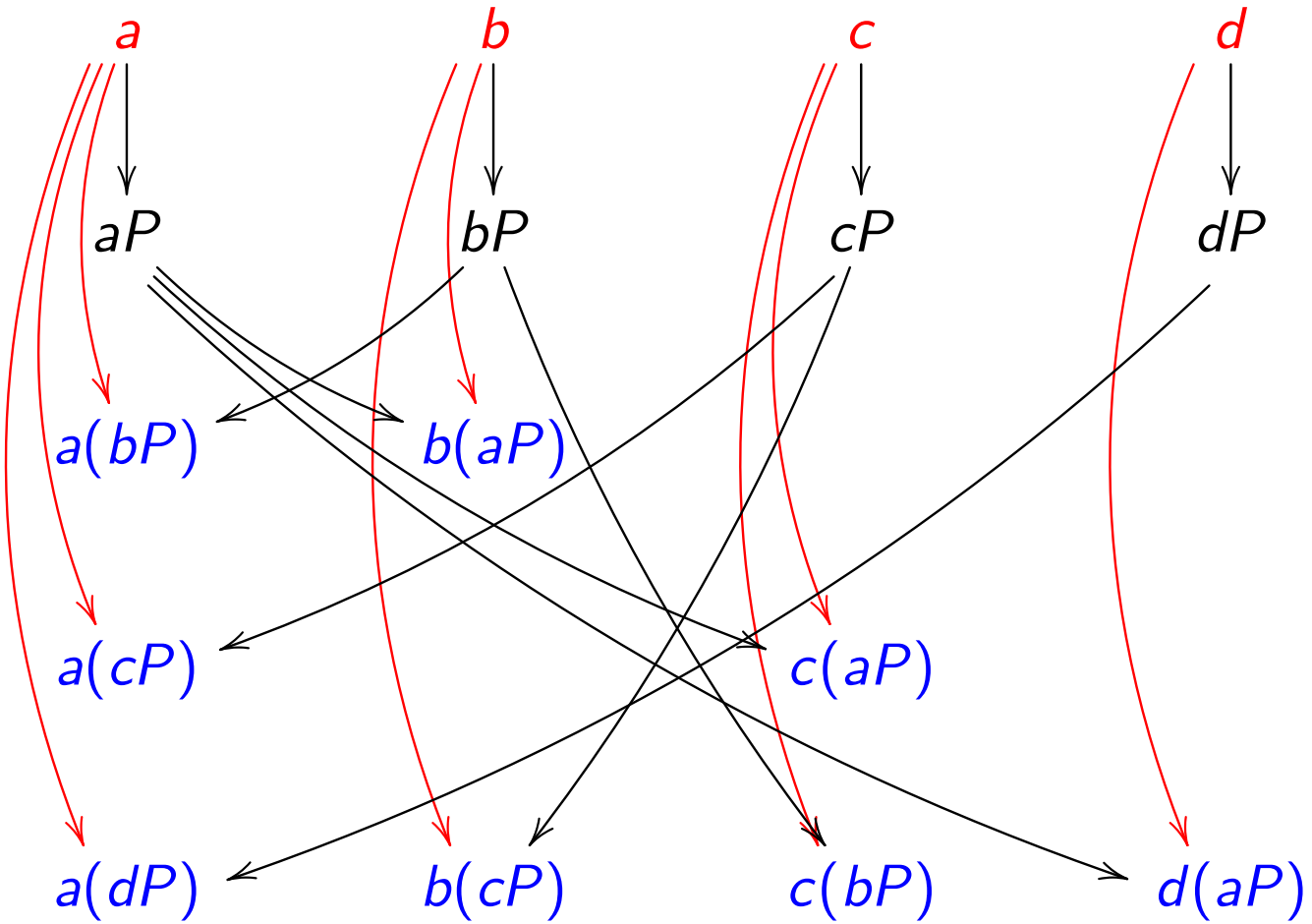
Diffie–Hellman Key Exchange



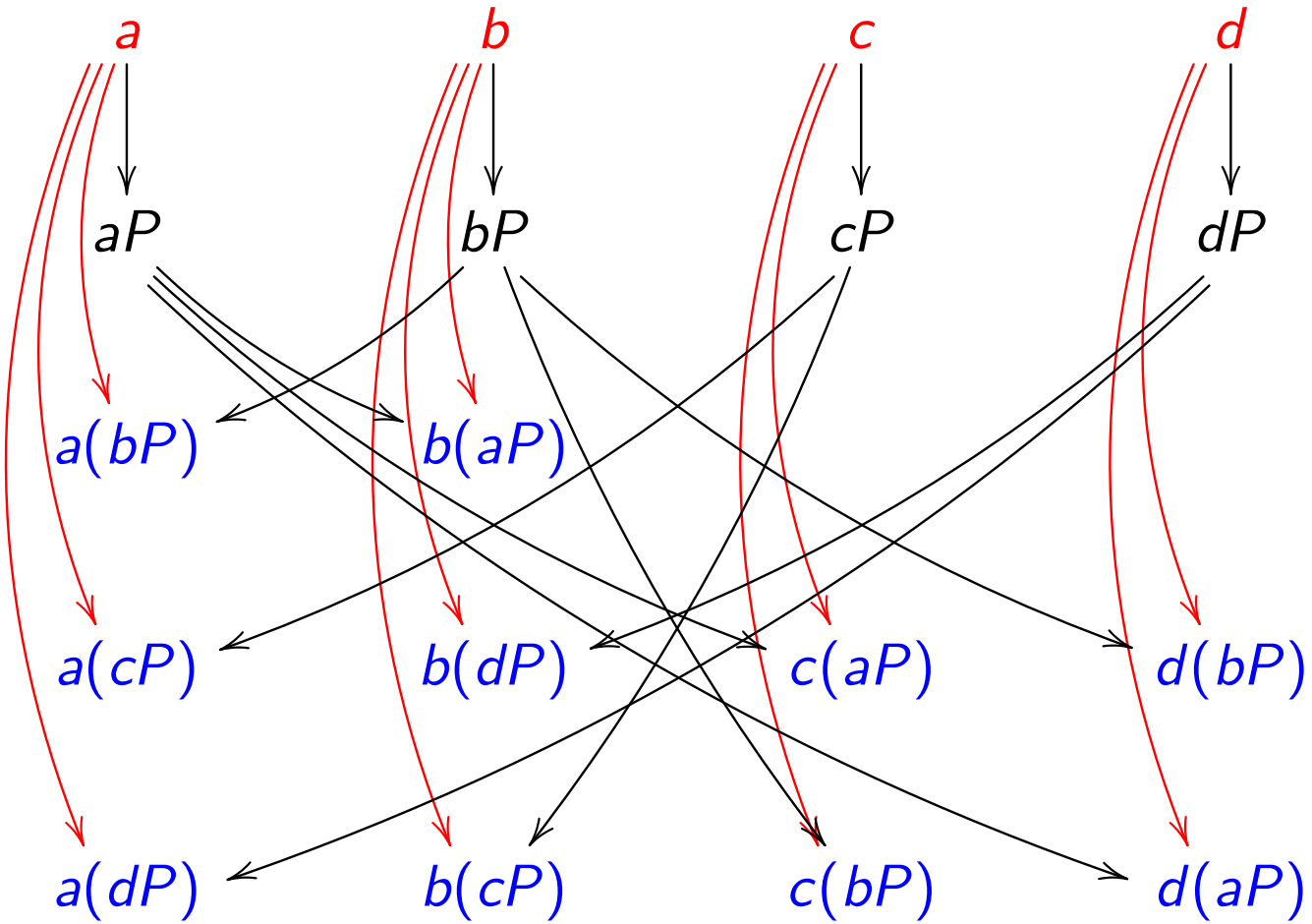
Diffie–Hellman Key Exchange



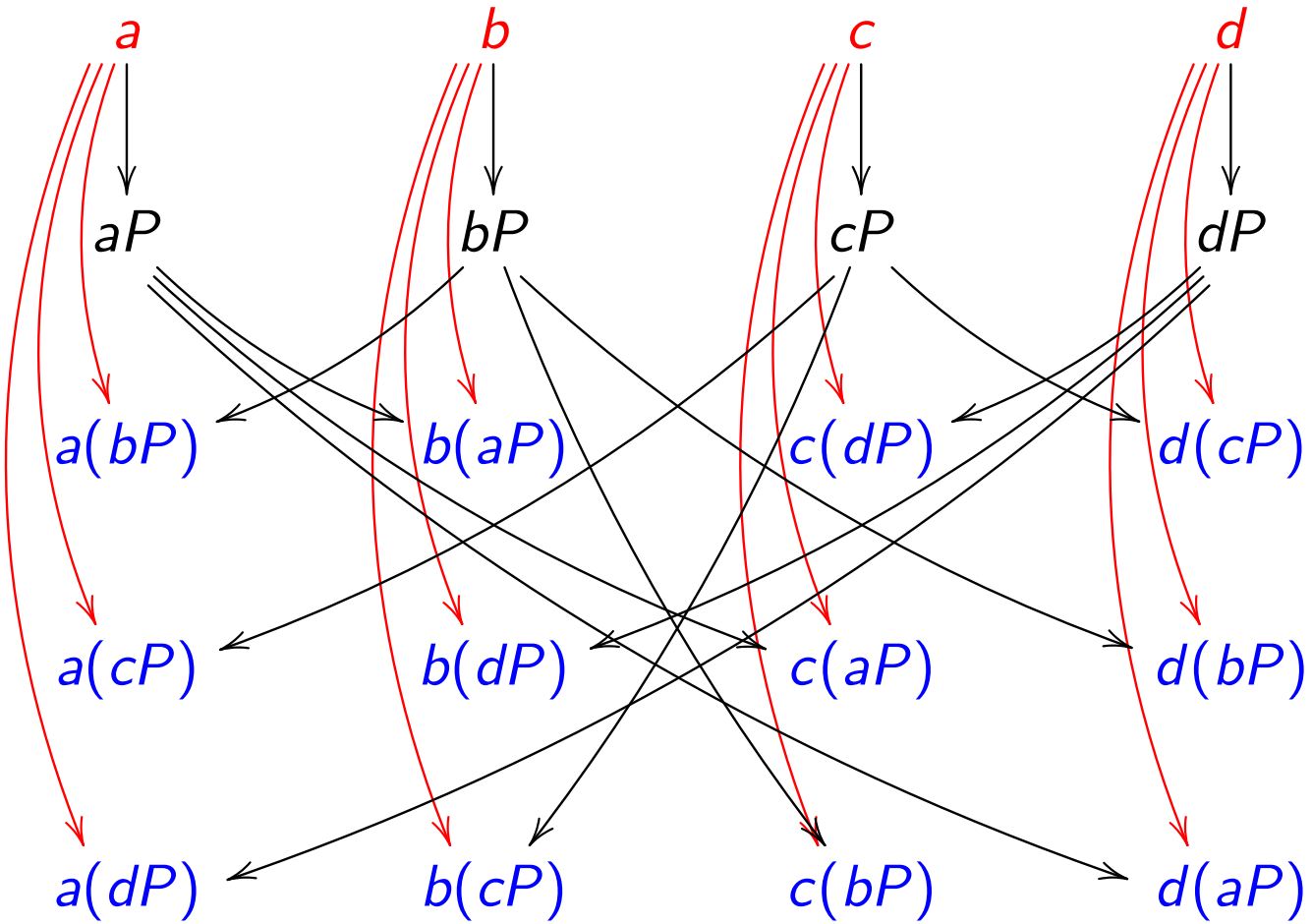
Diffie–Hellman Key Exchange



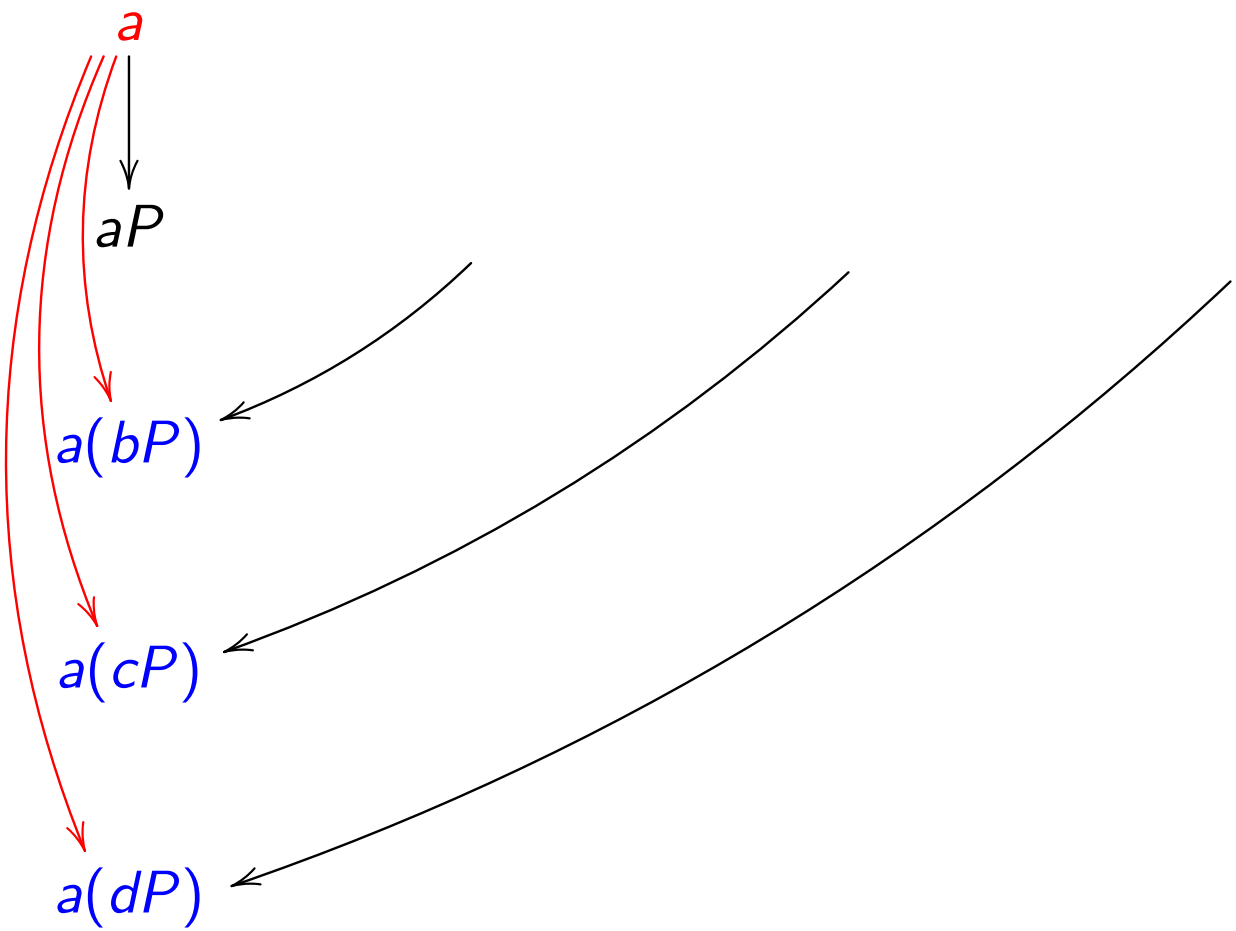
Diffie–Hellman Key Exchange



Diffie–Hellman Key Exchange

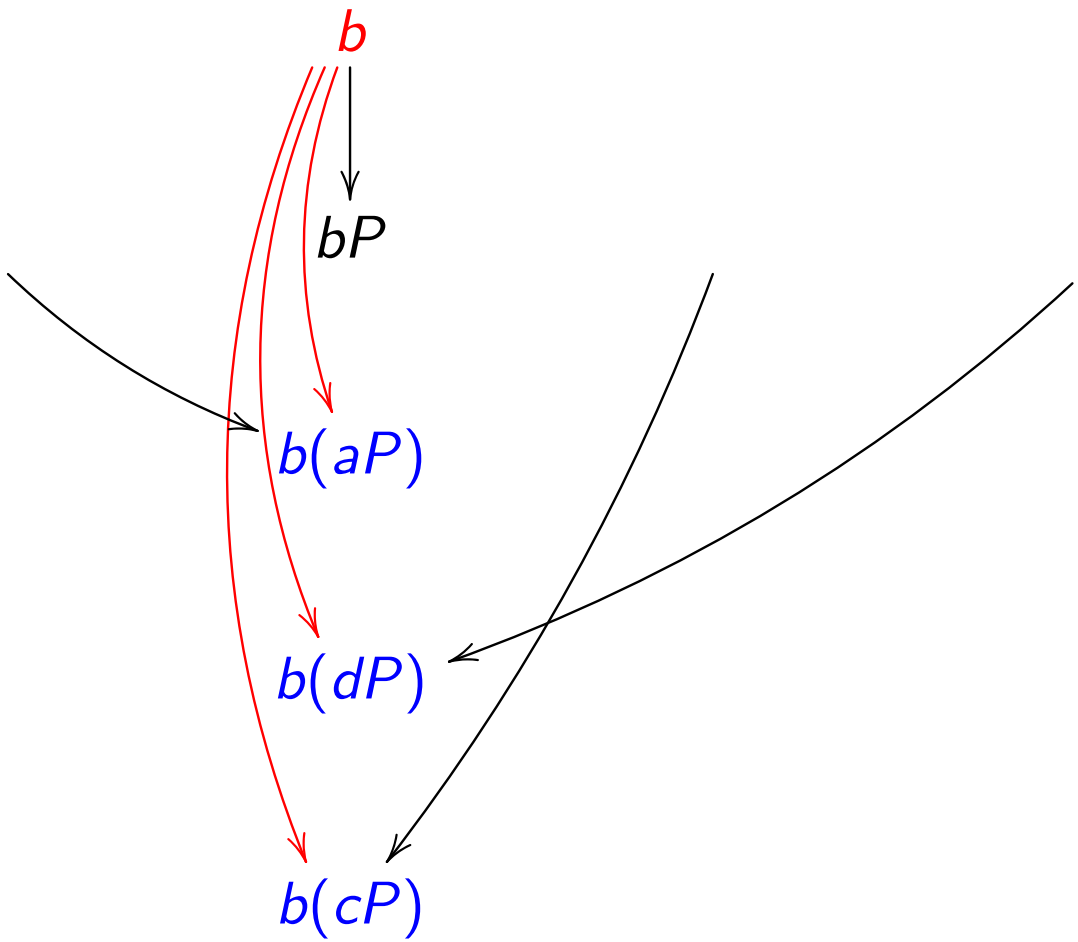


Diffie–Hellman Key Exchange



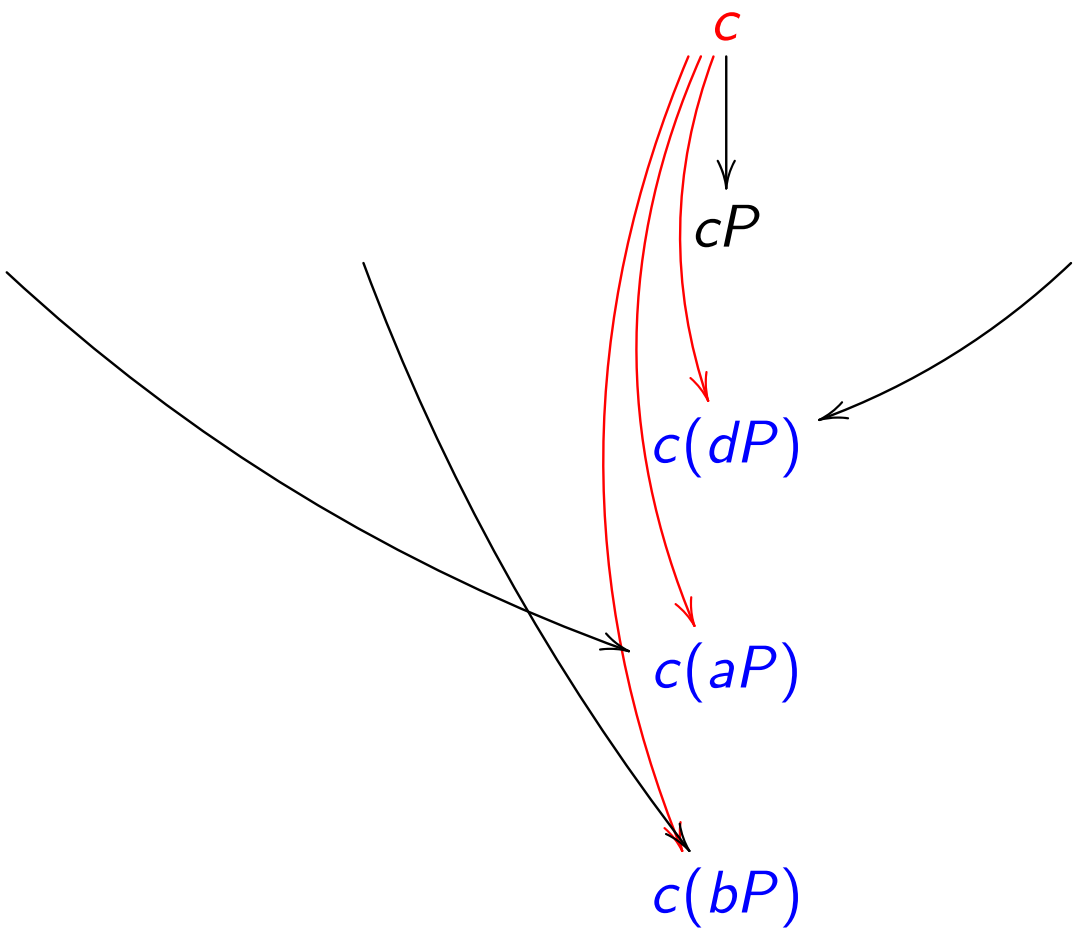
main DH challenge: make variable-base scalar mult as fast as possible

Diffie–Hellman Key Exchange



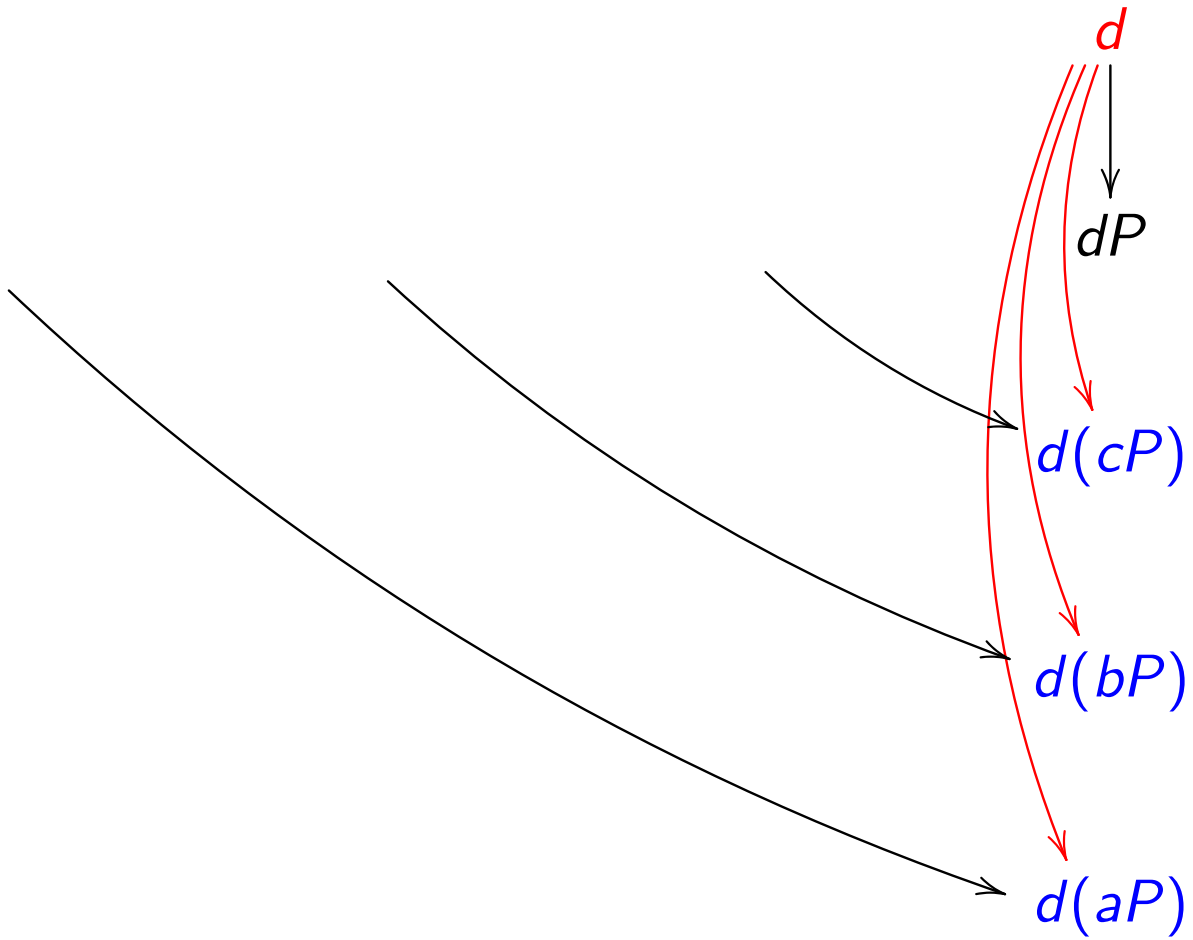
main DH challenge: make variable-base scalar mult as fast as possible

Diffie–Hellman Key Exchange



main DH challenge: make variable-base scalar mult as fast as possible

Diffie–Hellman Key Exchange



main DH challenge: make variable-base scalar mult as fast as possible

Part I

Kummer strikes back: new DH speed records
ASIACRYPT 2014

Joint work with Daniel J. Bernstein, Tanja Lange, and Peter Schwabe

Ladder

Input: $Q, n = (n_{i-1}, \ldots, n_0)_2$
Output: $R_0 = nQ$

$R_0 \leftarrow 0Q; \quad R_1 \leftarrow 1Q$
if $n_i = 0$ **then**
  $R_0 \leftarrow 2R_0; \quad R_1 \leftarrow R_0 + R_1$
else
  $R_0 \leftarrow R_0 + R_1; \quad R_1 \leftarrow 2R_1$

Return R_0

Example: Compute $9Q$

$9 = 1001_2$

$(R_0, R_1) = (0Q, 1Q)$

$n_3 = 1 : \quad (1Q, 2Q)$

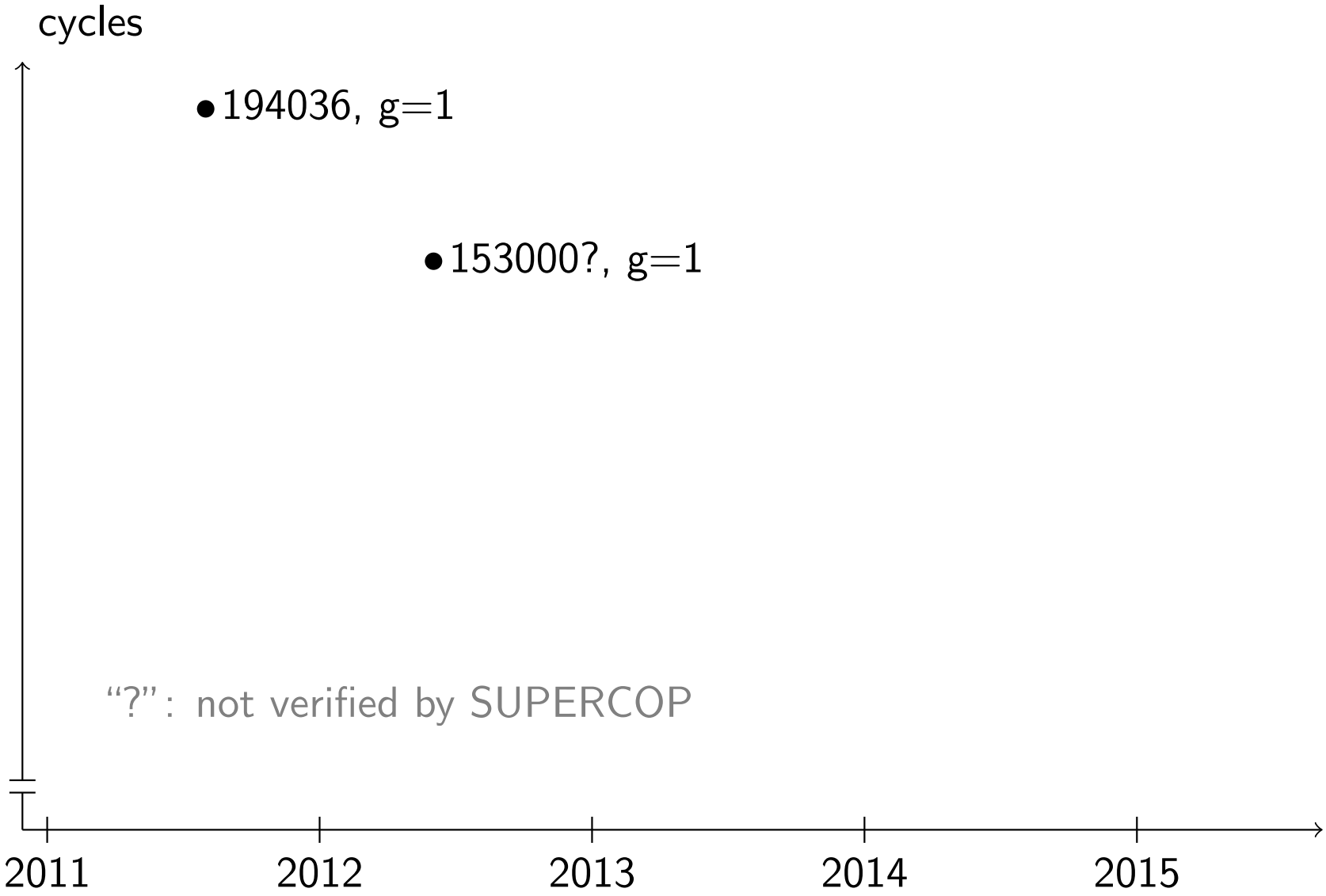
$n_2 = 0 : \quad (2Q, 3Q)$

$n_1 = 0 : \quad (4Q, 5Q)$

$n_0 = 1 : \quad (9Q, 10Q)$

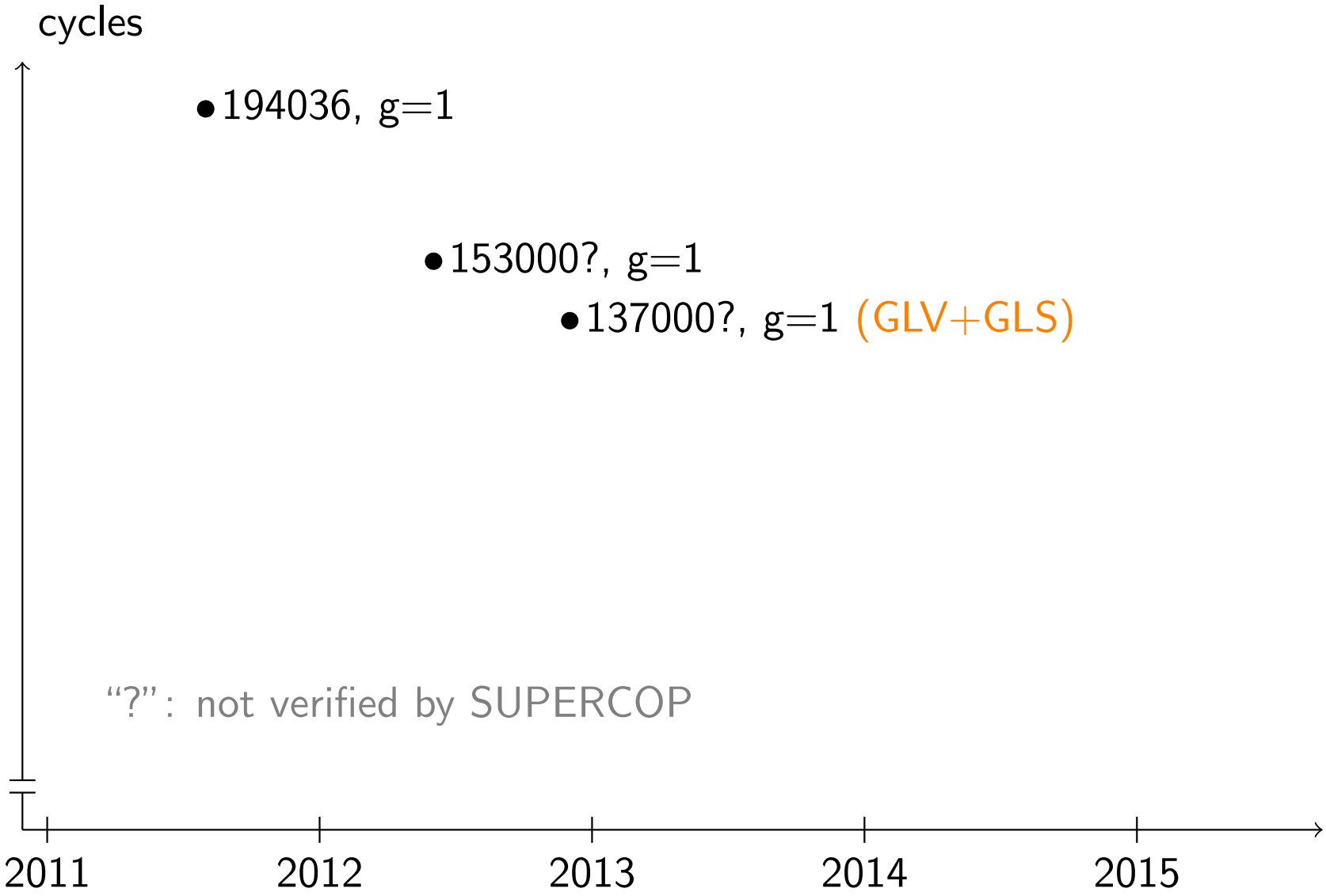
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



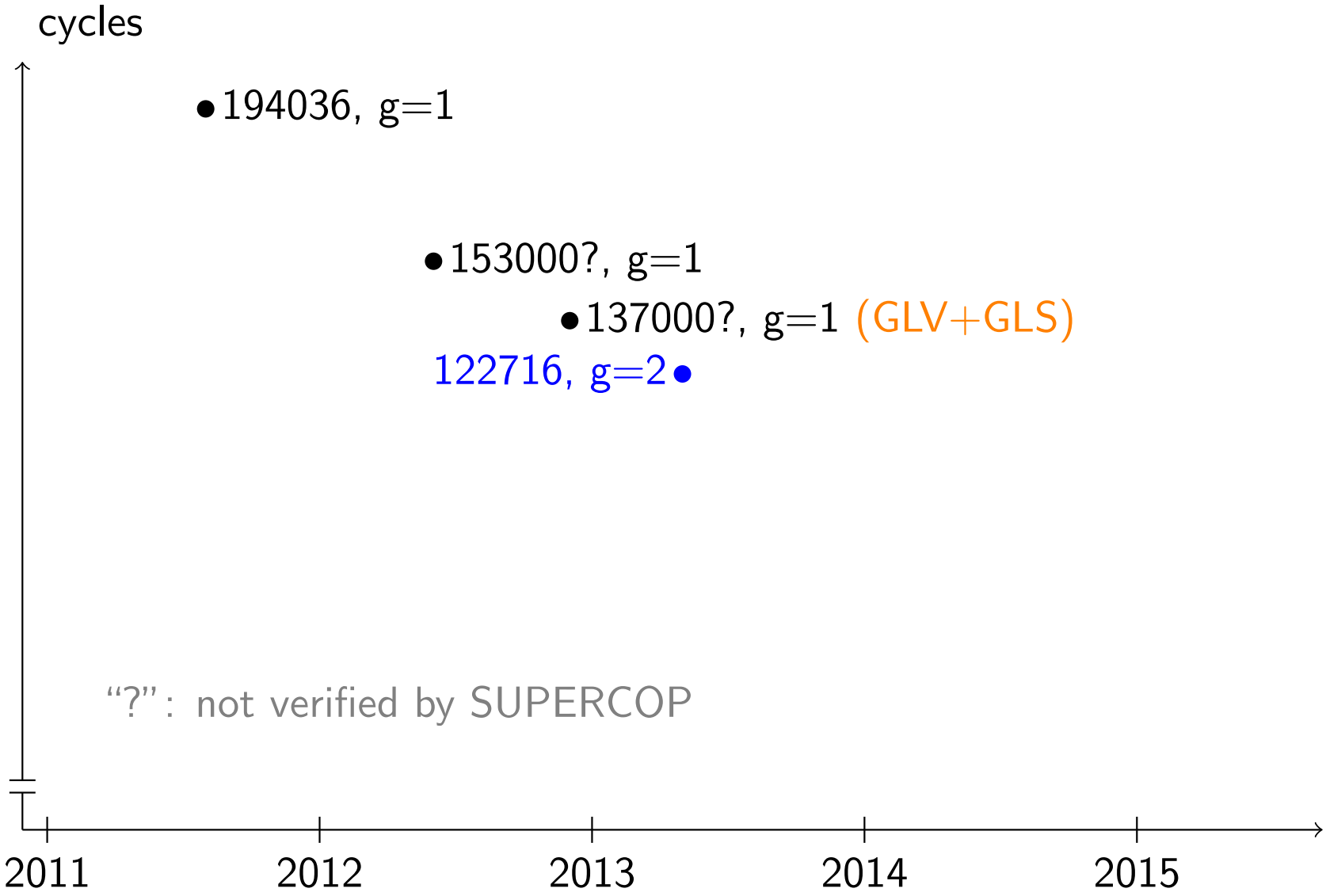
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



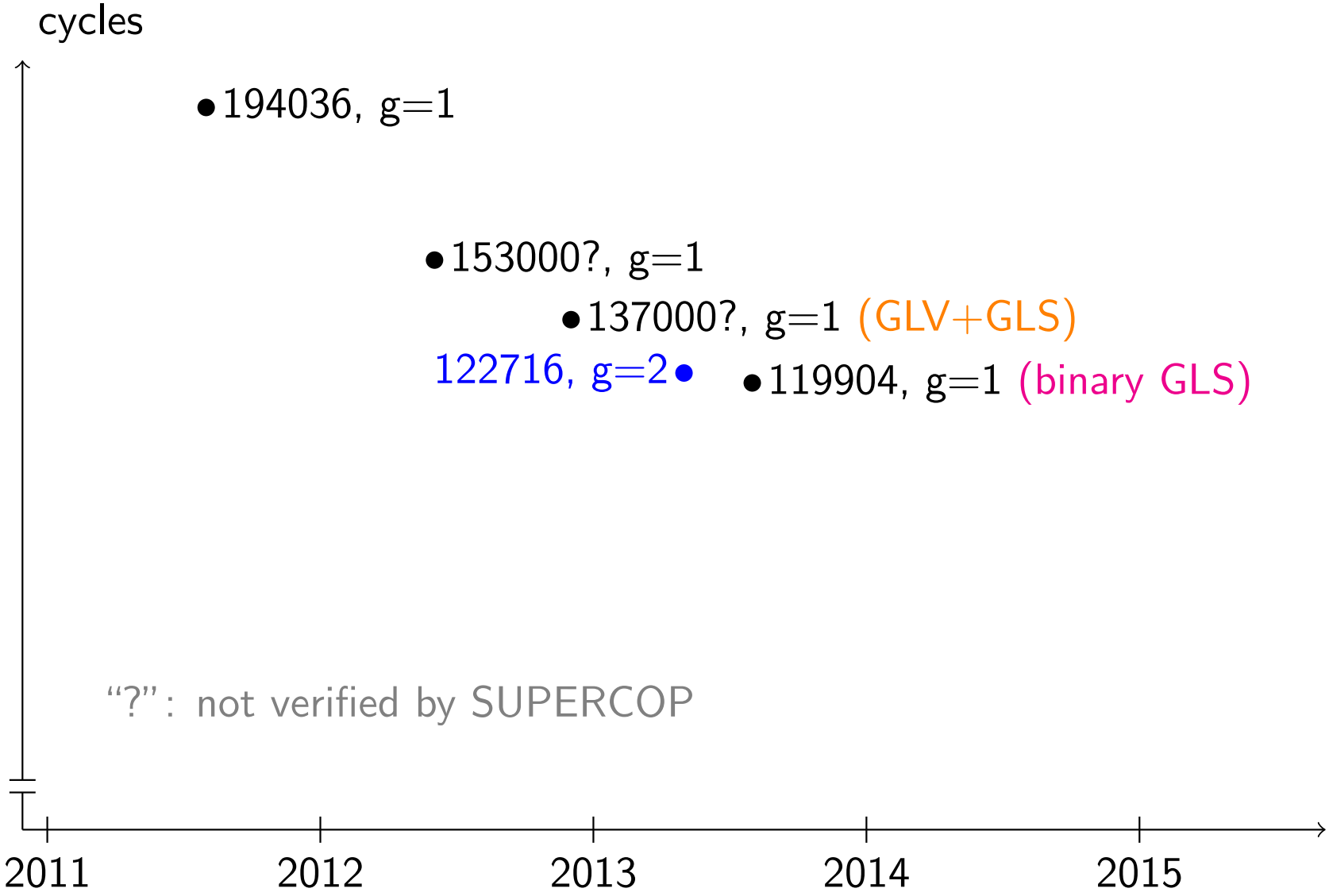
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



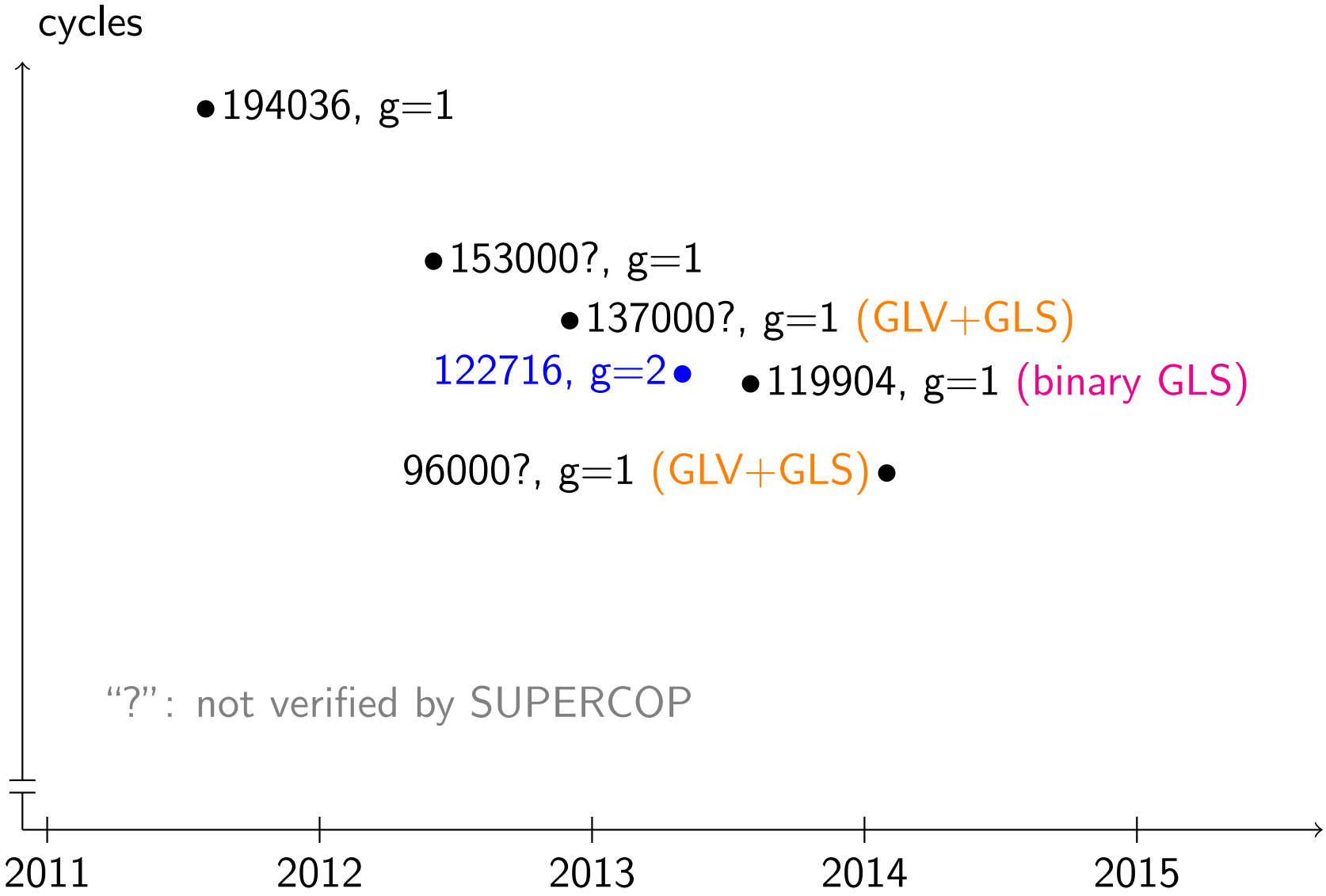
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



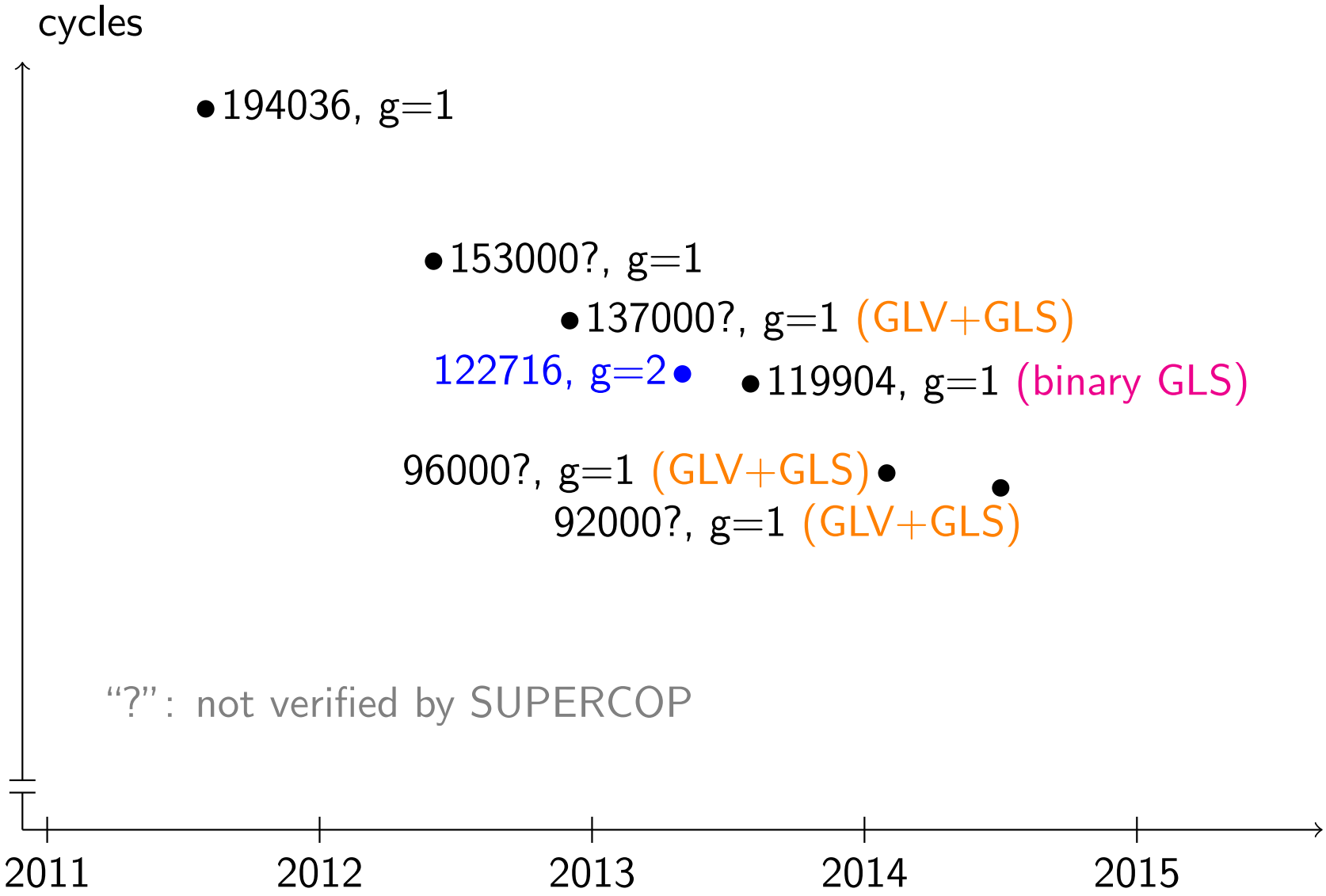
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



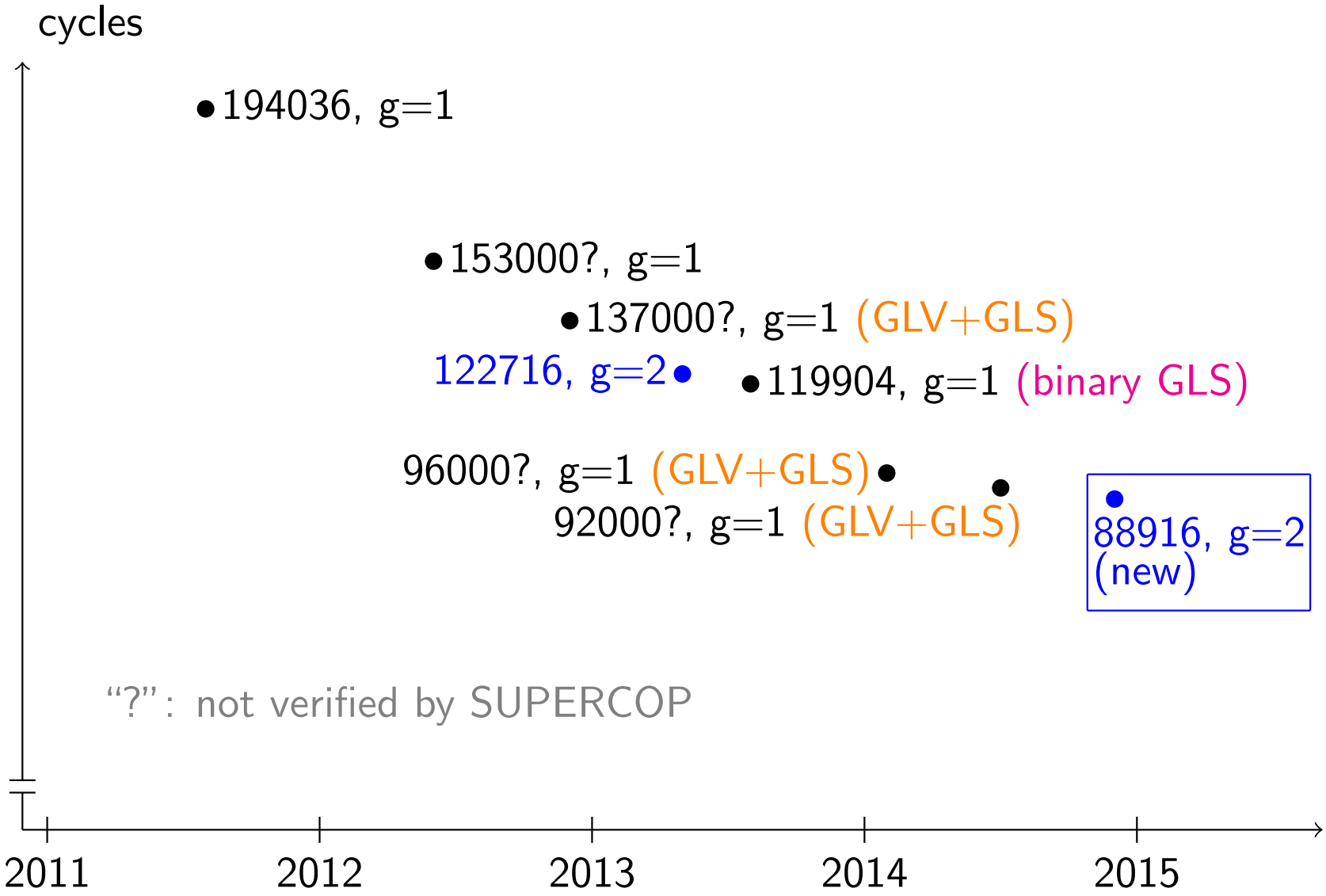
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



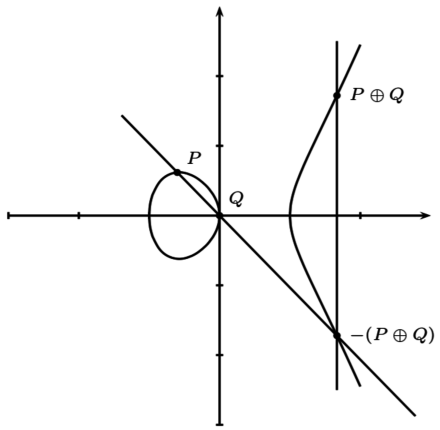
Elliptic VS Hyperelliptic

Sandy Bridge at high security level and (claimed) constant time



Elliptic-Hyperelliptic Analogy

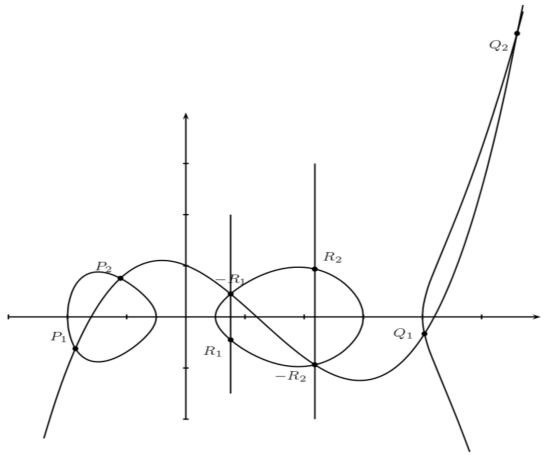
ECC



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

$\longrightarrow$ x-line represented as $(X : Z)$

HECC



$v^2 = u^5 + f_4u^4 + f_3u^3 + f_2u^2 + f_1u^1 + f_0$

$\longrightarrow$ Kummer surface represented as $(X : Y : Z : T)$

Hyperelliptic Advantages

- smaller field size

Hyperelliptic Advantages

- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$ (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster

Hyperelliptic Advantages

- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$ (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster
- improvement to HECC formulas

Hyperelliptic Advantages

- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$
    (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster
- improvement to HECC formulas
  - only 25 multiplications with Gaudry ladder for Kummer surface

Hyperelliptic Advantages

- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$
    (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster
- improvement to HECC formulas
  - only 25 multiplications with Gaudry ladder for Kummer surface
- discovery of *small* constants for Kummer surface

Hyperelliptic Advantages

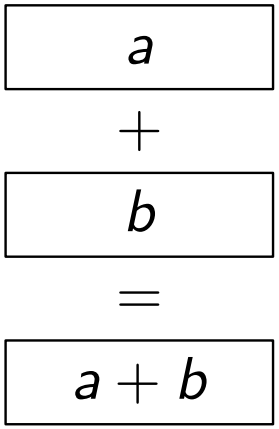
- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$
    (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster
- improvement to HECC formulas
  - only 25 multiplications with Gaudry ladder for Kummer surface
- discovery of *small* constants for Kummer surface
  - 6 of 25 multiplications are by *small* constant

Hyperelliptic Advantages

- smaller field size
  - need field size $\approx 2^{128}$ for group size $\approx 2^{256}$
    (ECC would need field size $\approx 2^{256}$)
  - field arithmetic 2–4 times faster
- improvement to HECC formulas
  - only 25 multiplications with Gaudry ladder for Kummer surface
- discovery of *small* constants for Kummer surface
  - 6 of 25 multiplications are by *small* constant
- new: formulas well suited for vectorization

Vectorization

without vector



Vectorization

without vector

a
$+$
b
$=$
$a + b$

with vector

a_0	a_1	a_2	a_3
$+$	$+$	$+$	$+$
b_0	b_1	b_2	b_3
$=$	$=$	$=$	$=$
$a_0 + b_0$	$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$

Vectorization

without vector

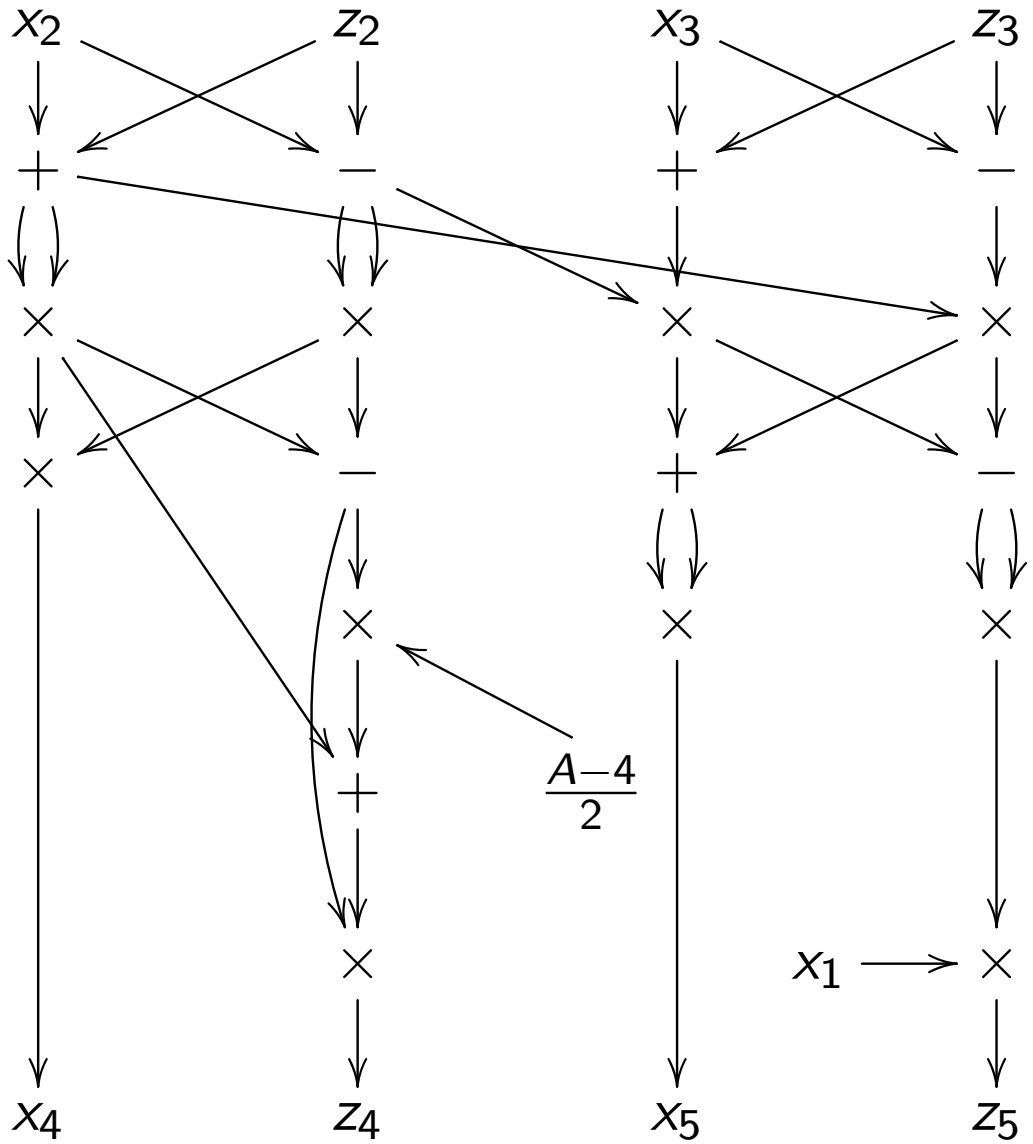
a
$+$
b
$=$
$a + b$

with vector

a_0	a_1	a_2	a_3
$+$	$+$	$+$	$+$
b_0	b_1	b_2	b_3
$=$	$=$	$=$	$=$
$a_0 + b_0$	$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$

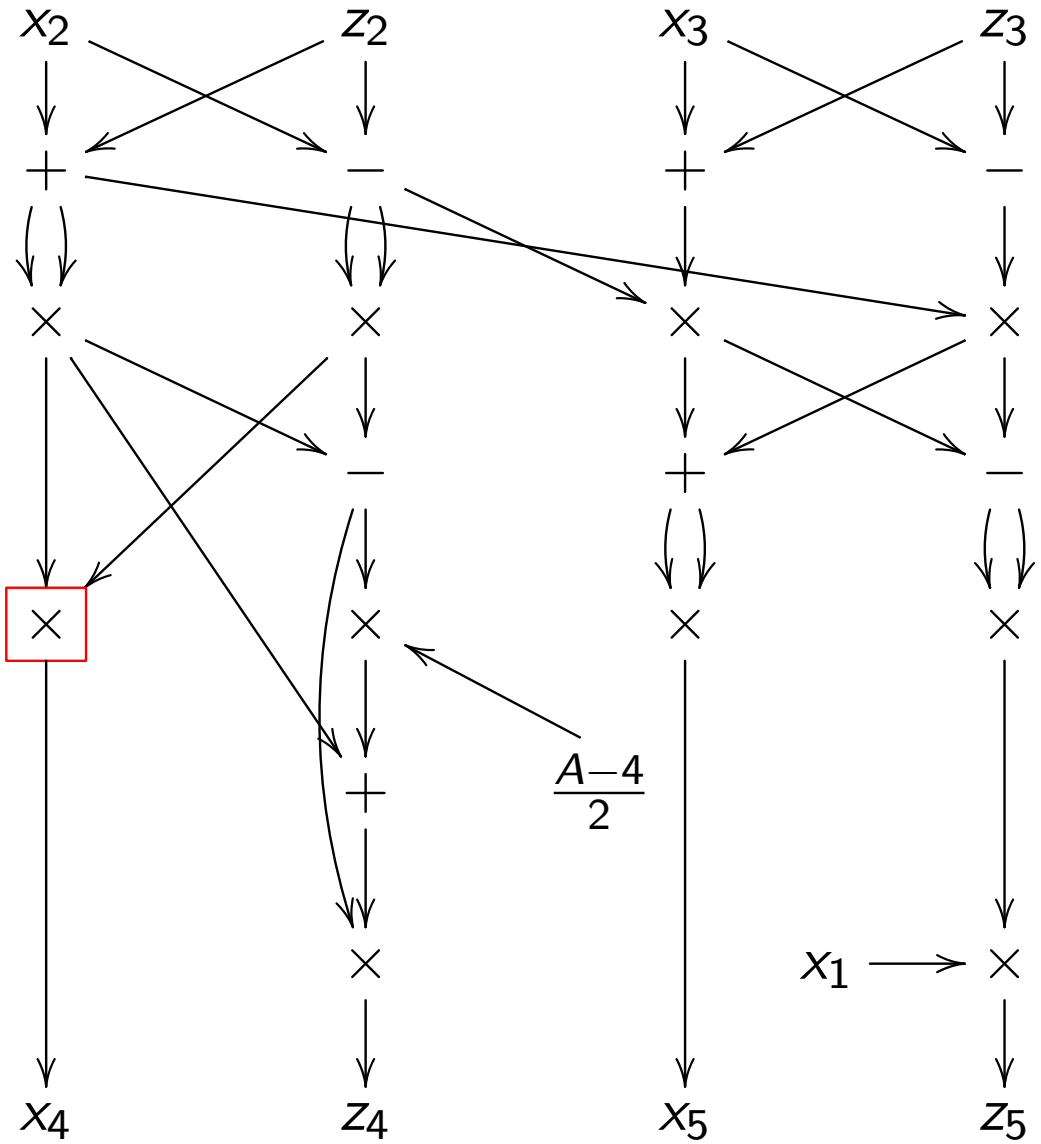
- **single** instruction performing n **independent** operations on **aligned** inputs

ECC Montgomery Ladder (original)



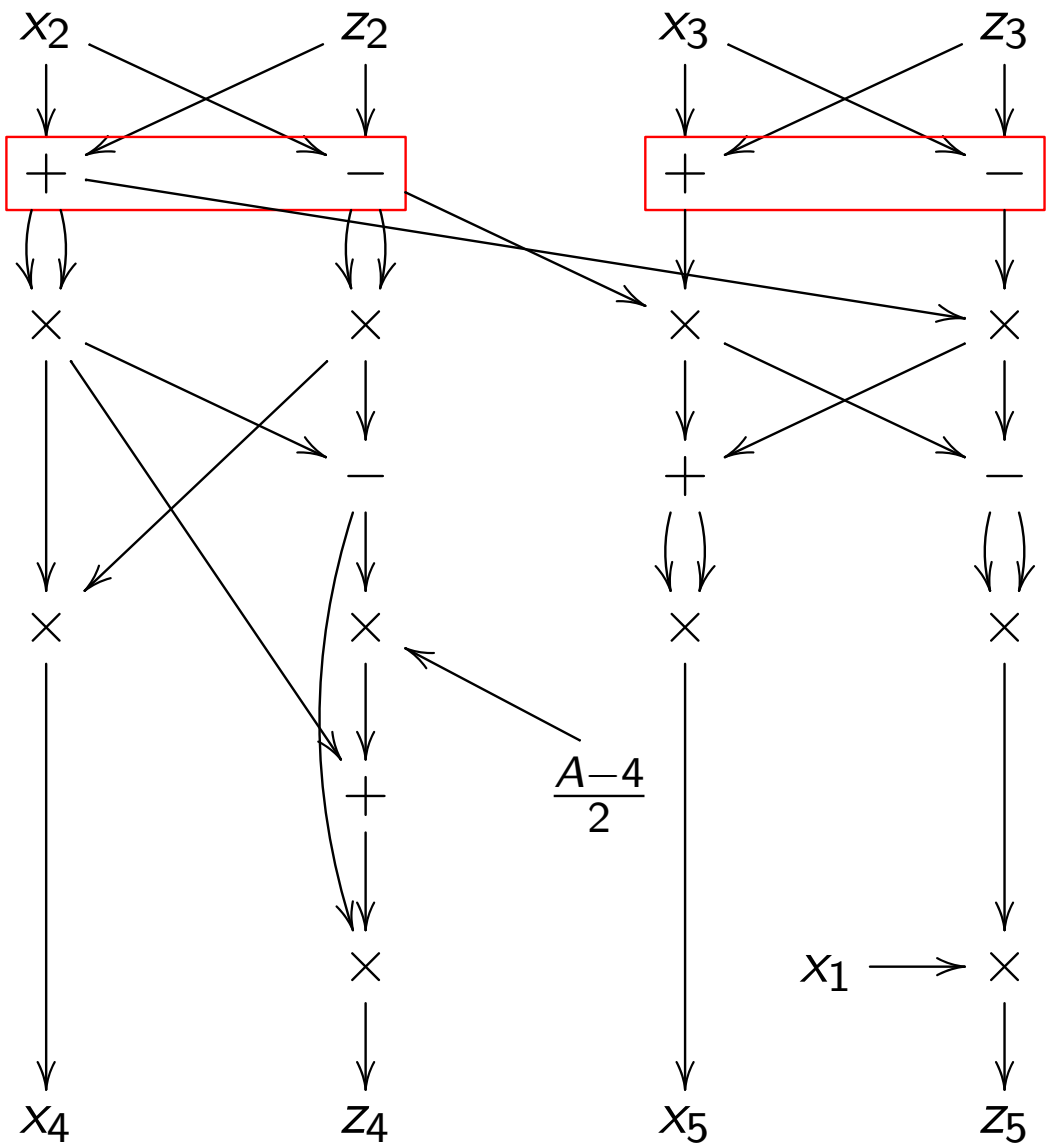
ECC Montgomery Ladder (2-way vectorization)

- move $\times$ down



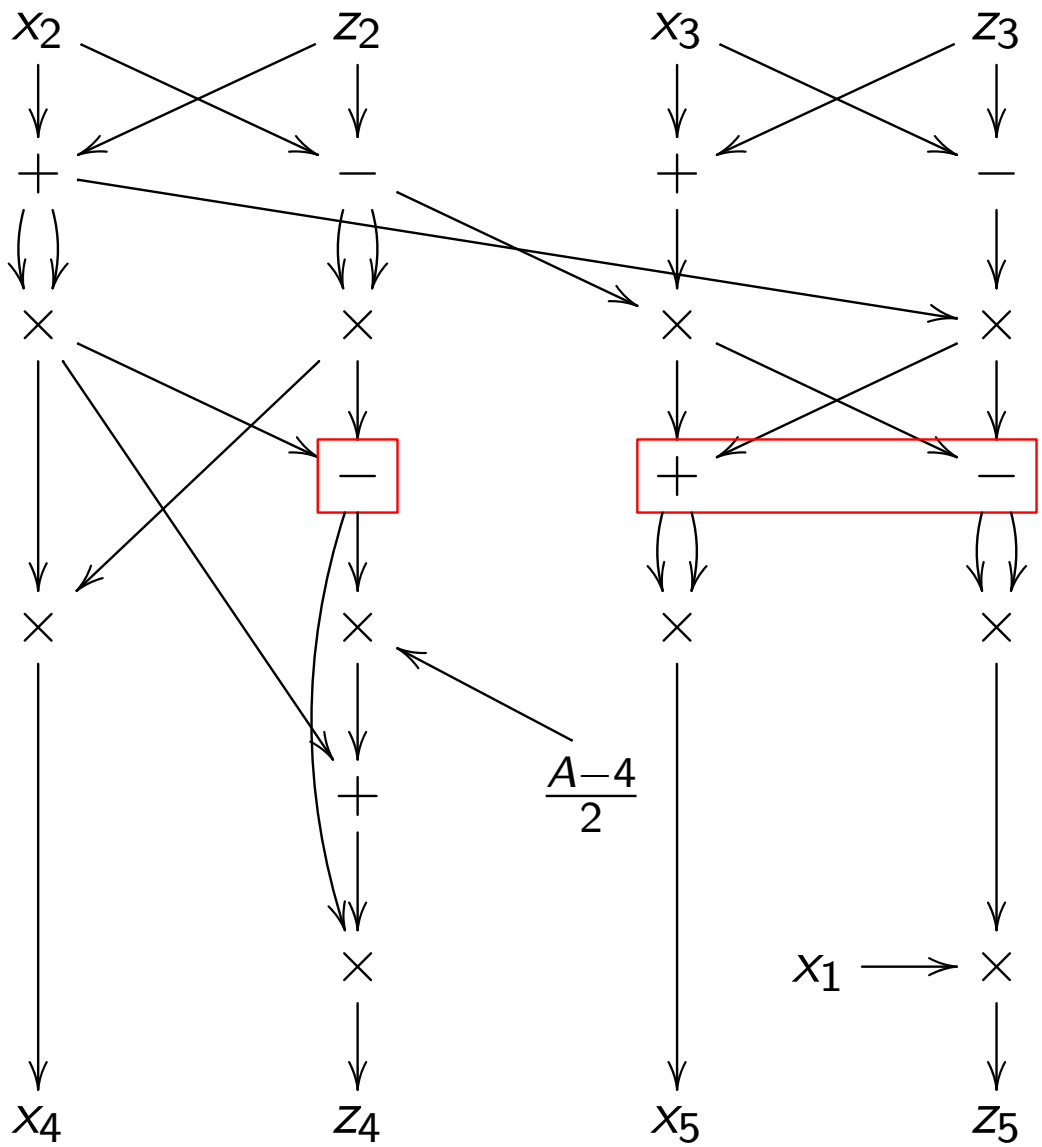
ECC Montgomery Ladder (2-way vectorization)

- require permutation
- move $\times$ down



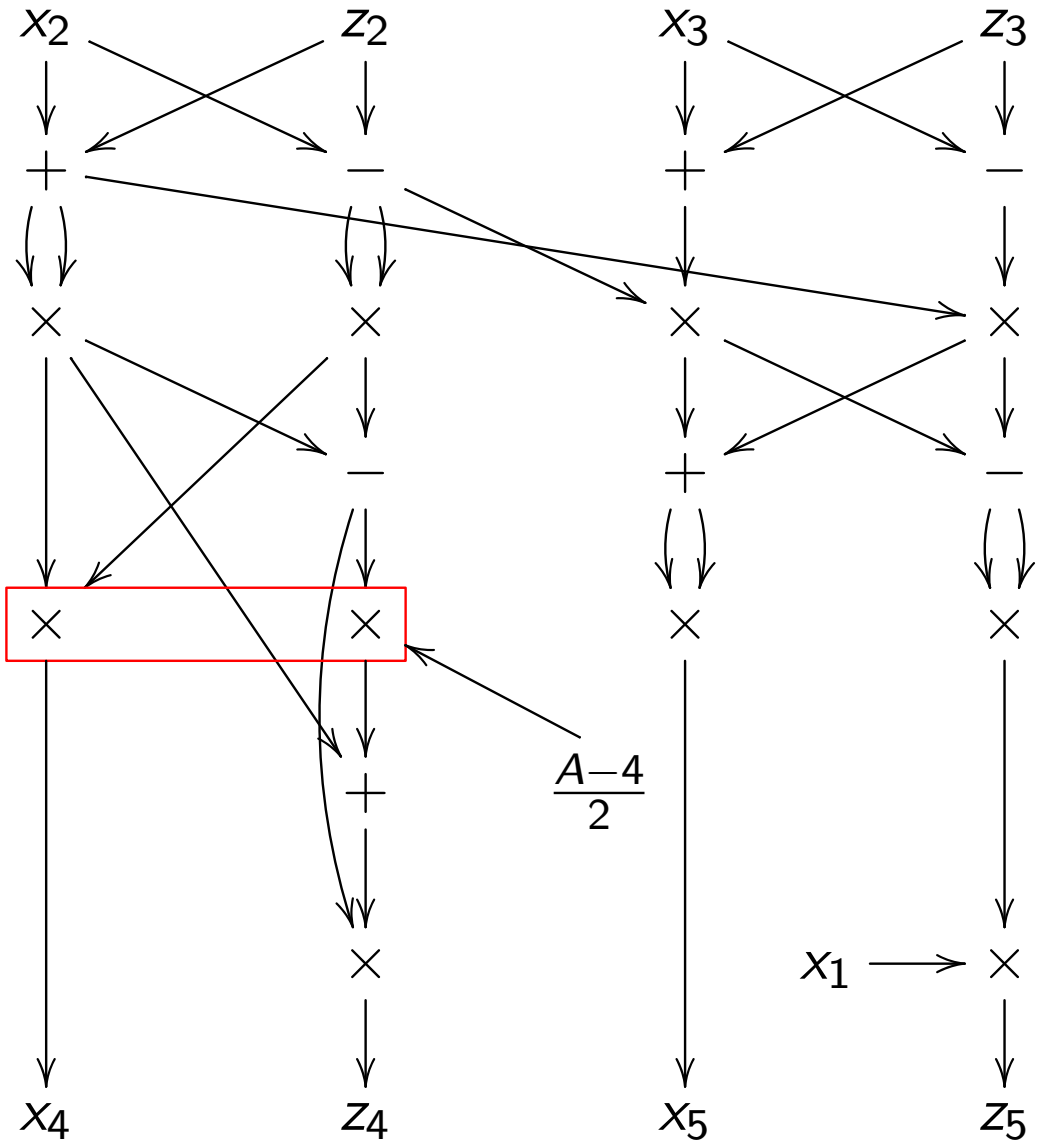
ECC Montgomery Ladder (2-way vectorization)

- require permutation
- move $\times$ down
- require permutation and leave $+$ idle



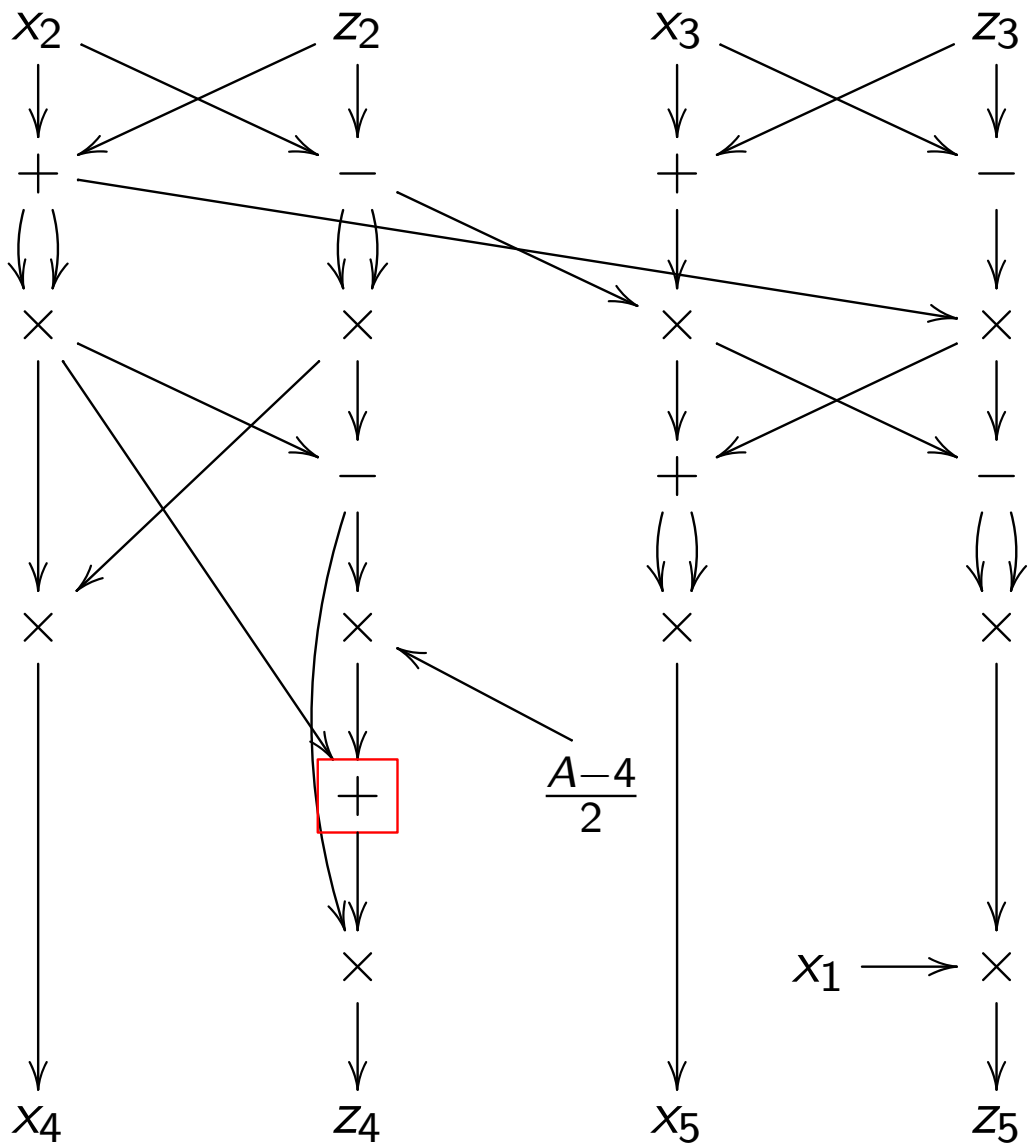
ECC Montgomery Ladder (2-way vectorization)

- require permutation
- move $\times$ down
- require permutation and leave $+$ idle
- slow down mult. by constant



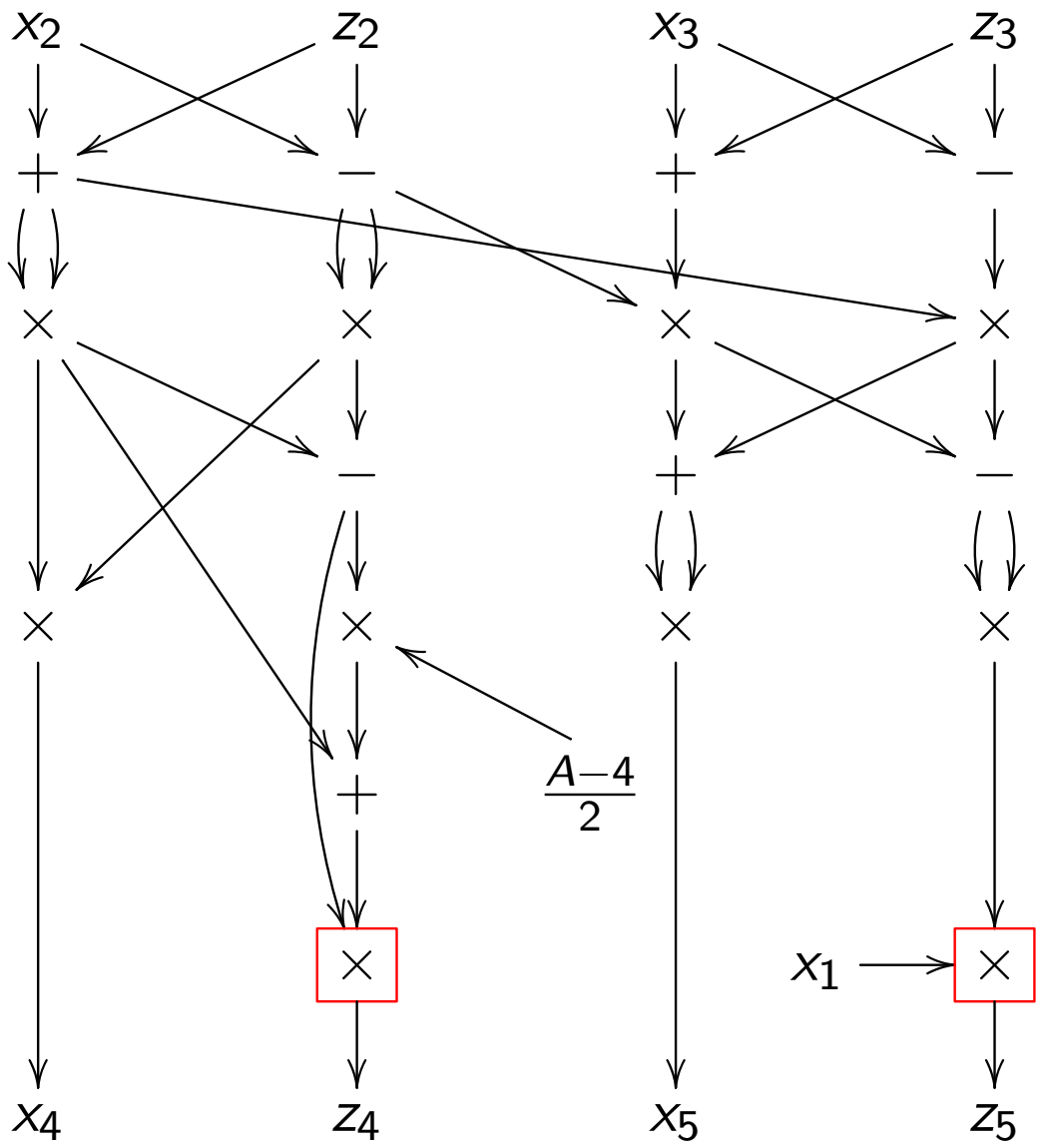
ECC Montgomery Ladder (2-way vectorization)

- require permutation
- move $\times$ down
- require permutation and leave $+$ idle
- slow down mult. by constant
- nothing to match $+$



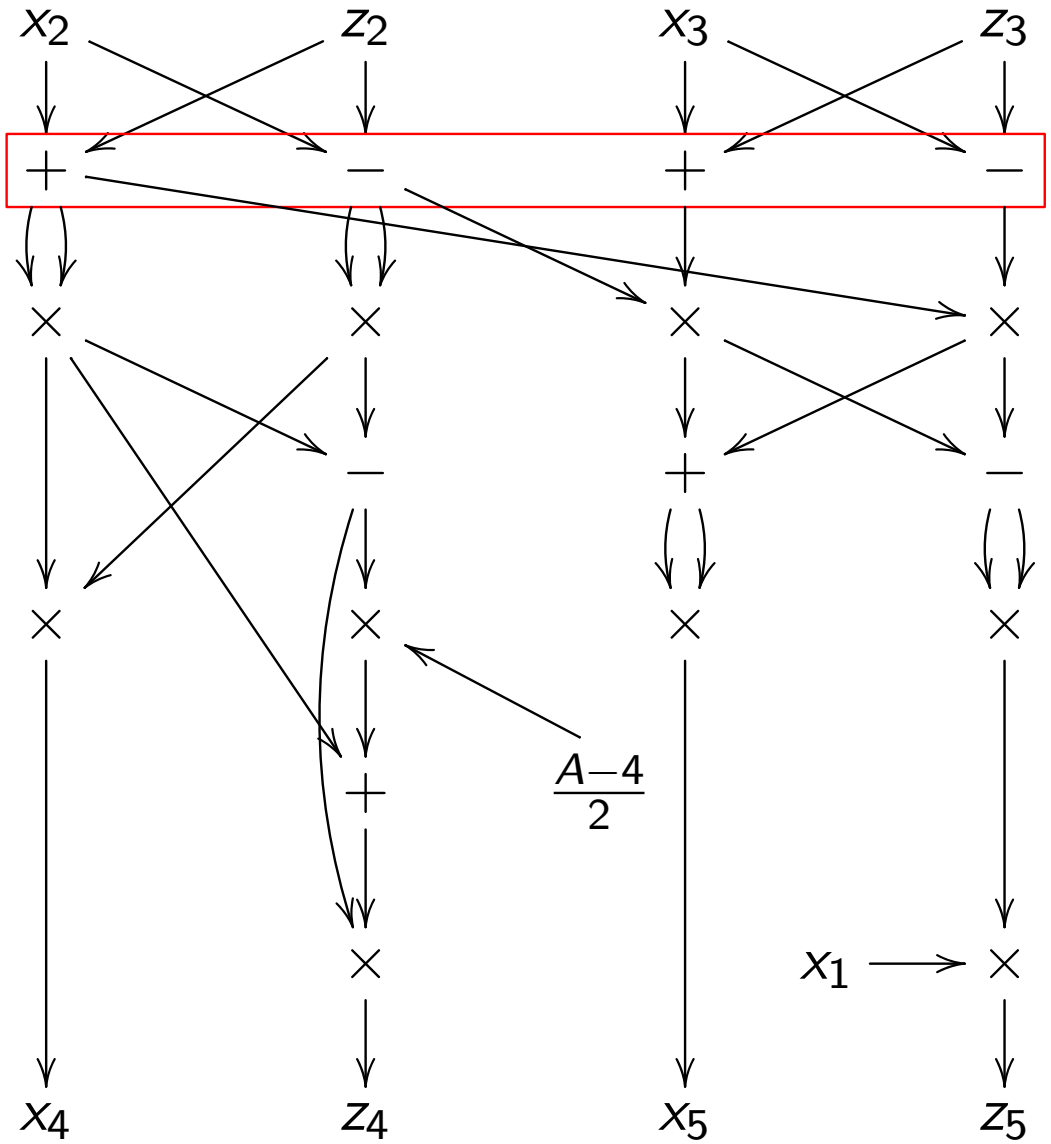
ECC Montgomery Ladder (2-way vectorization)

- require permutation
- move $\times$ down
- require permutation and leave $+$ idle
- slow down mult. by constant
- nothing to match $+$
- require permutation



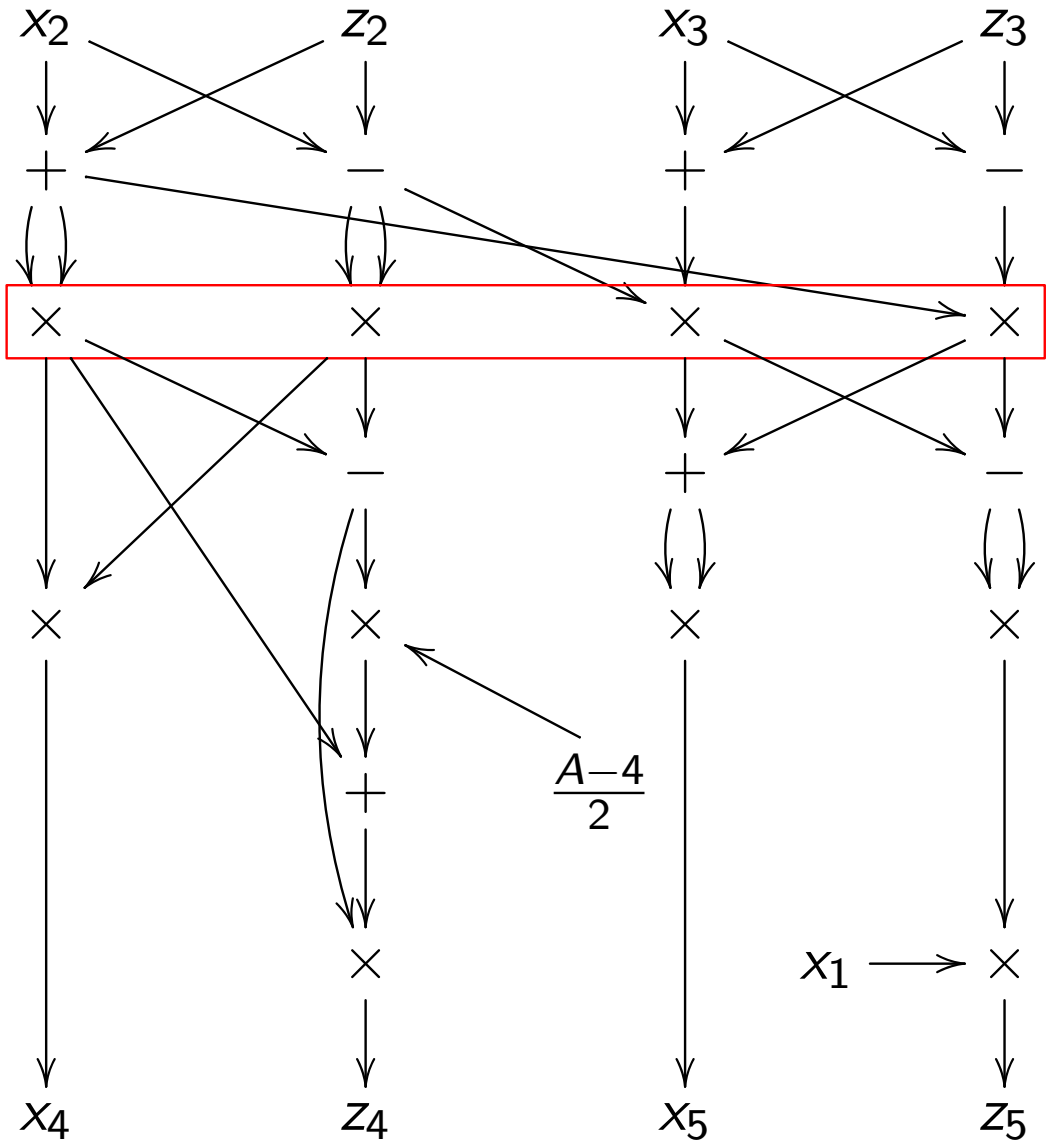
ECC Montgomery Ladder (4-way vectorization)

- match $+$ with $-$



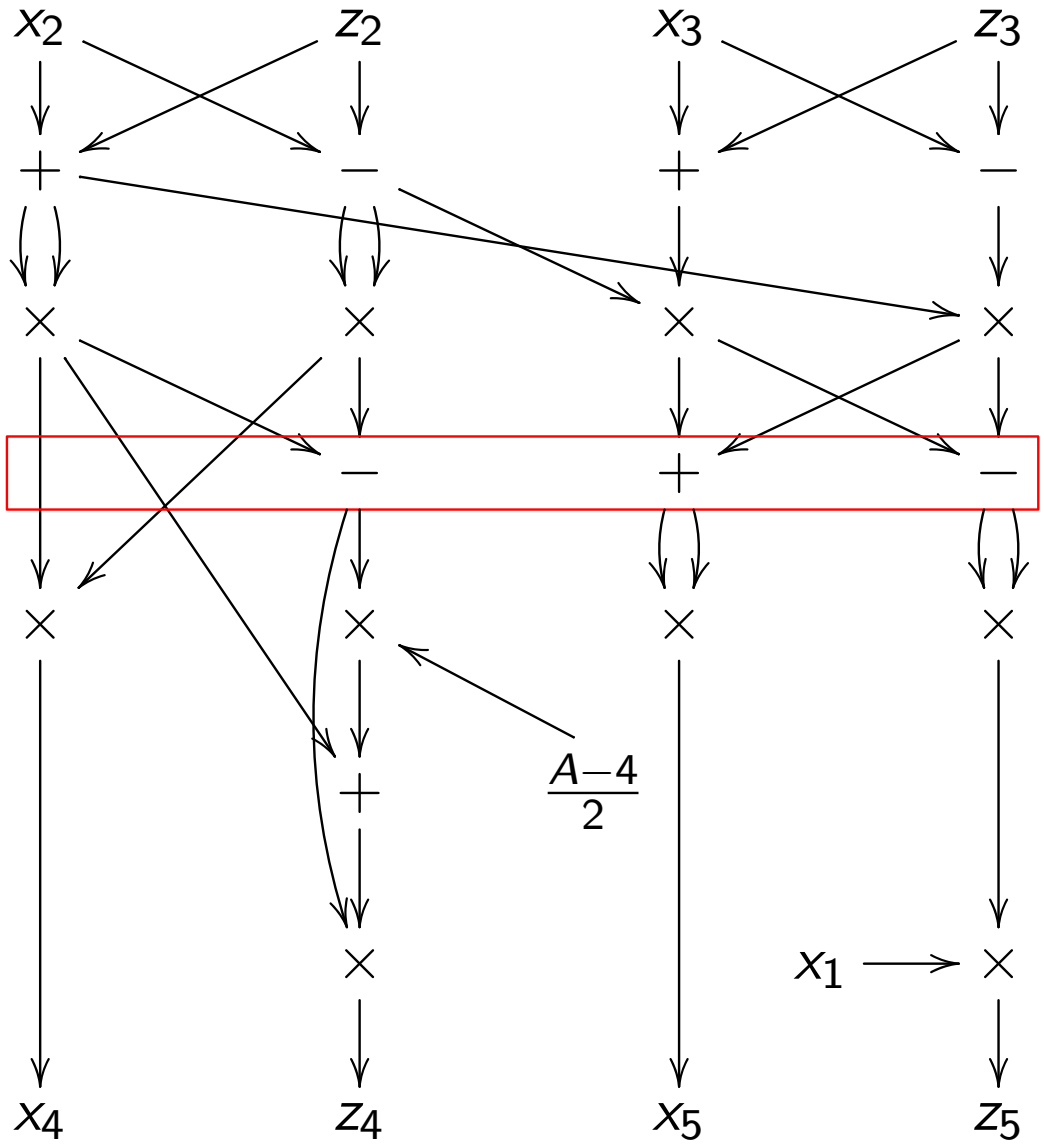
ECC Montgomery Ladder (4-way vectorization)

- match $+$ with $-$
- slow down squaring



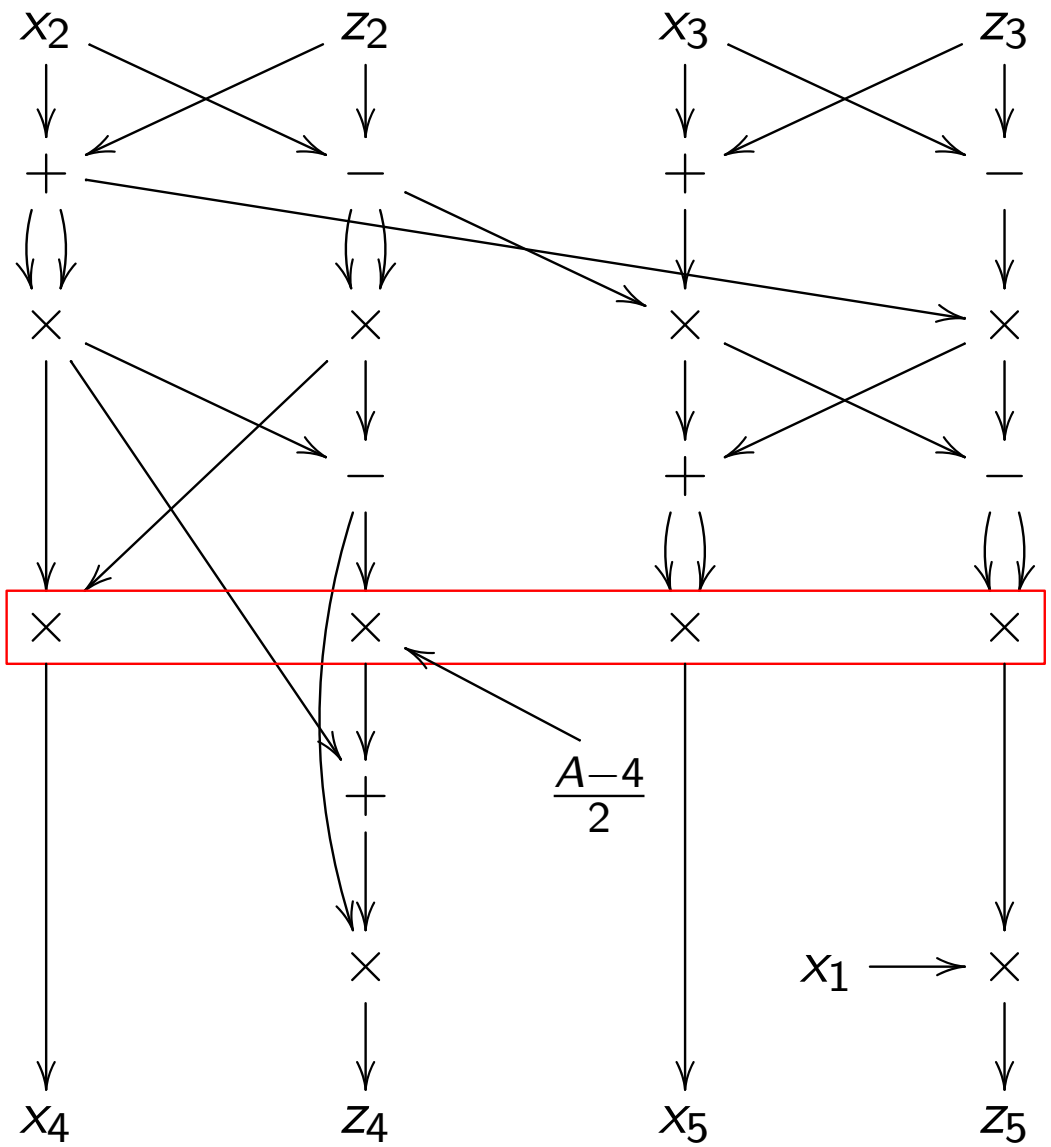
ECC Montgomery Ladder (4-way vectorization)

- match $+$ with $-$
- slow down squaring
- nothing to match and match $+$ and $-$



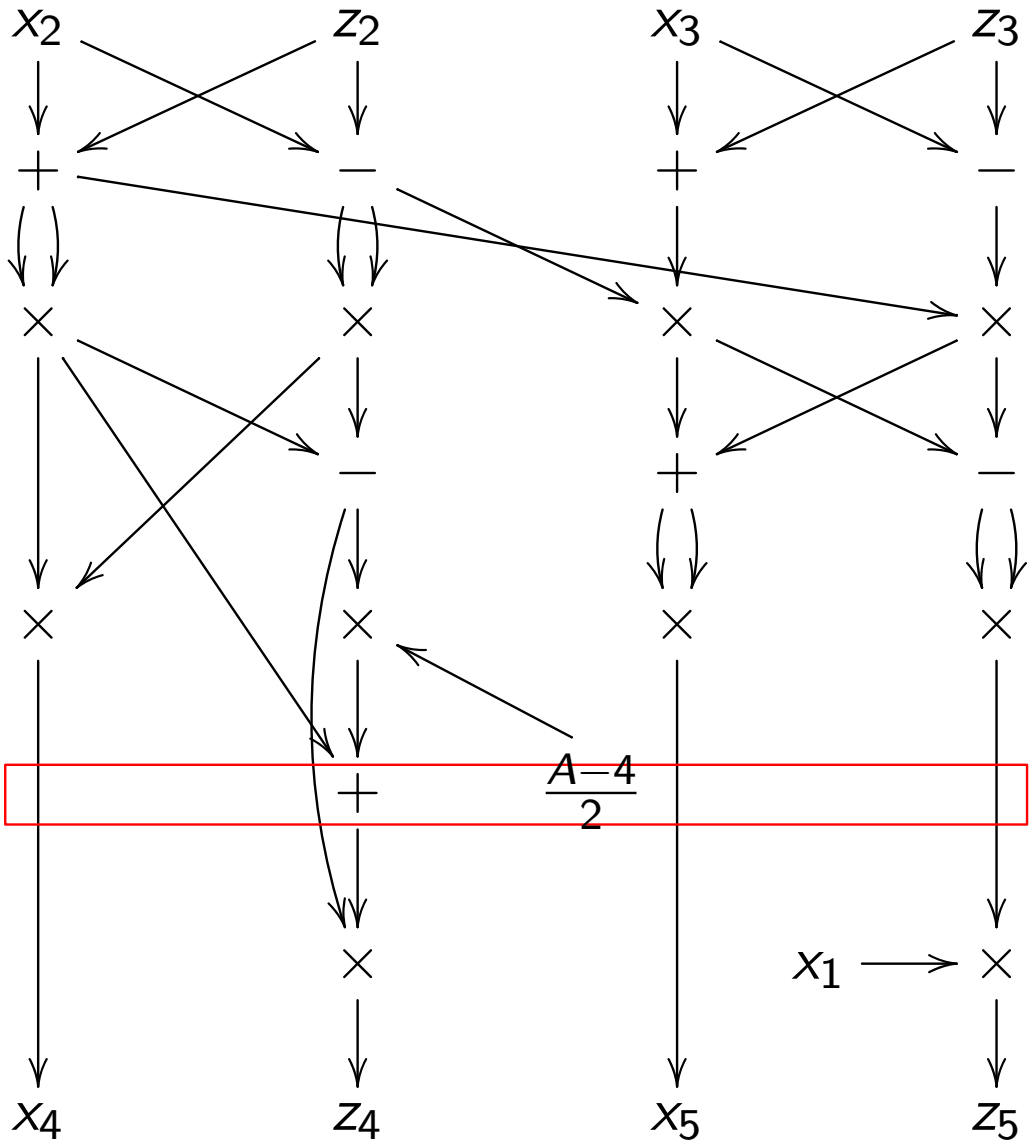
ECC Montgomery Ladder (4-way vectorization)

- match $+$ with $-$
- slow down squaring
- nothing to match and match $+$ and $-$
- slow down mult. by constant and squaring



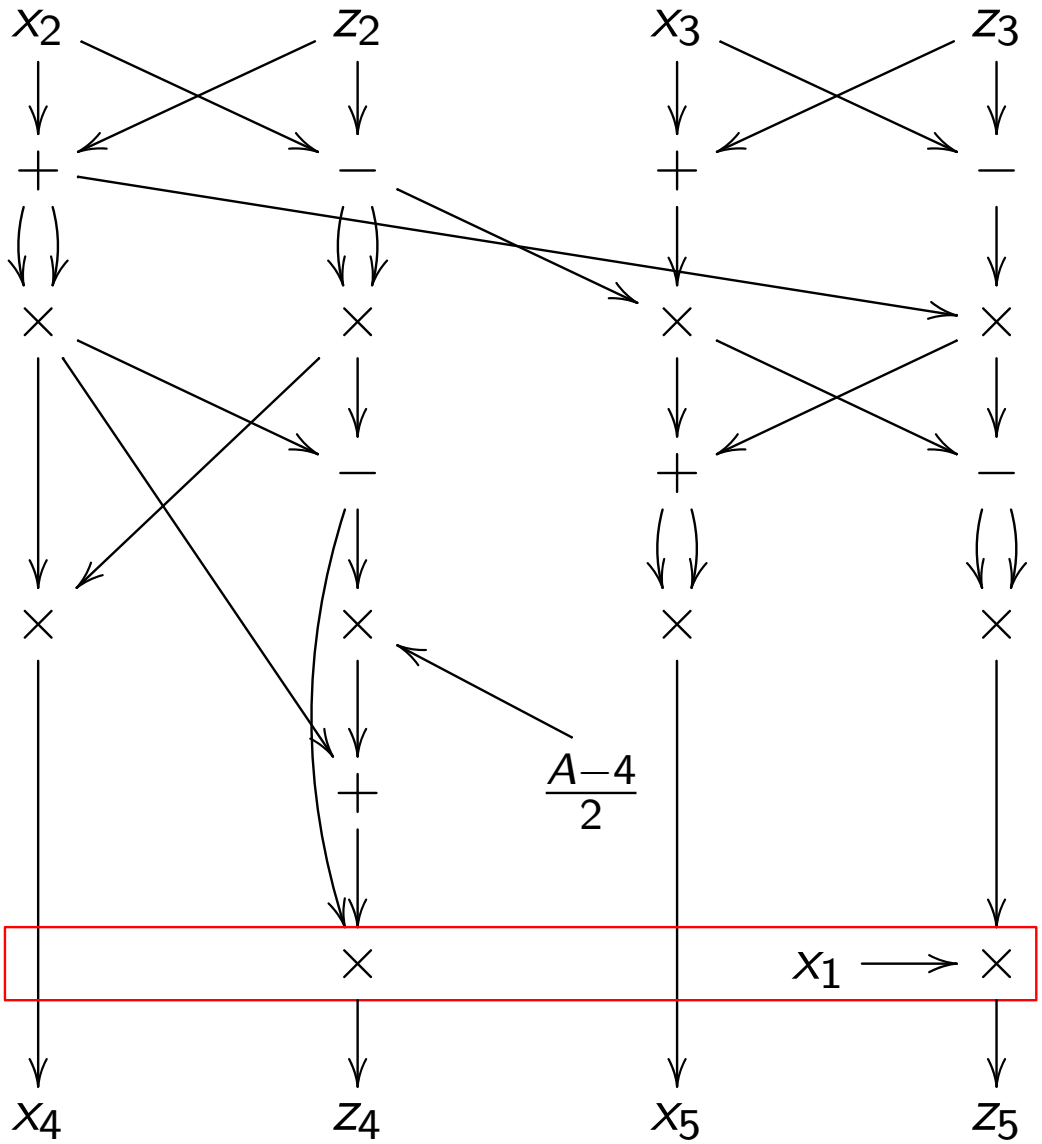
ECC Montgomery Ladder (4-way vectorization)

- match $+$ with $-$
- slow down squaring
- nothing to match and match $+$ and $-$
- slow down mult. by constant and squaring
- nothing to match $+$

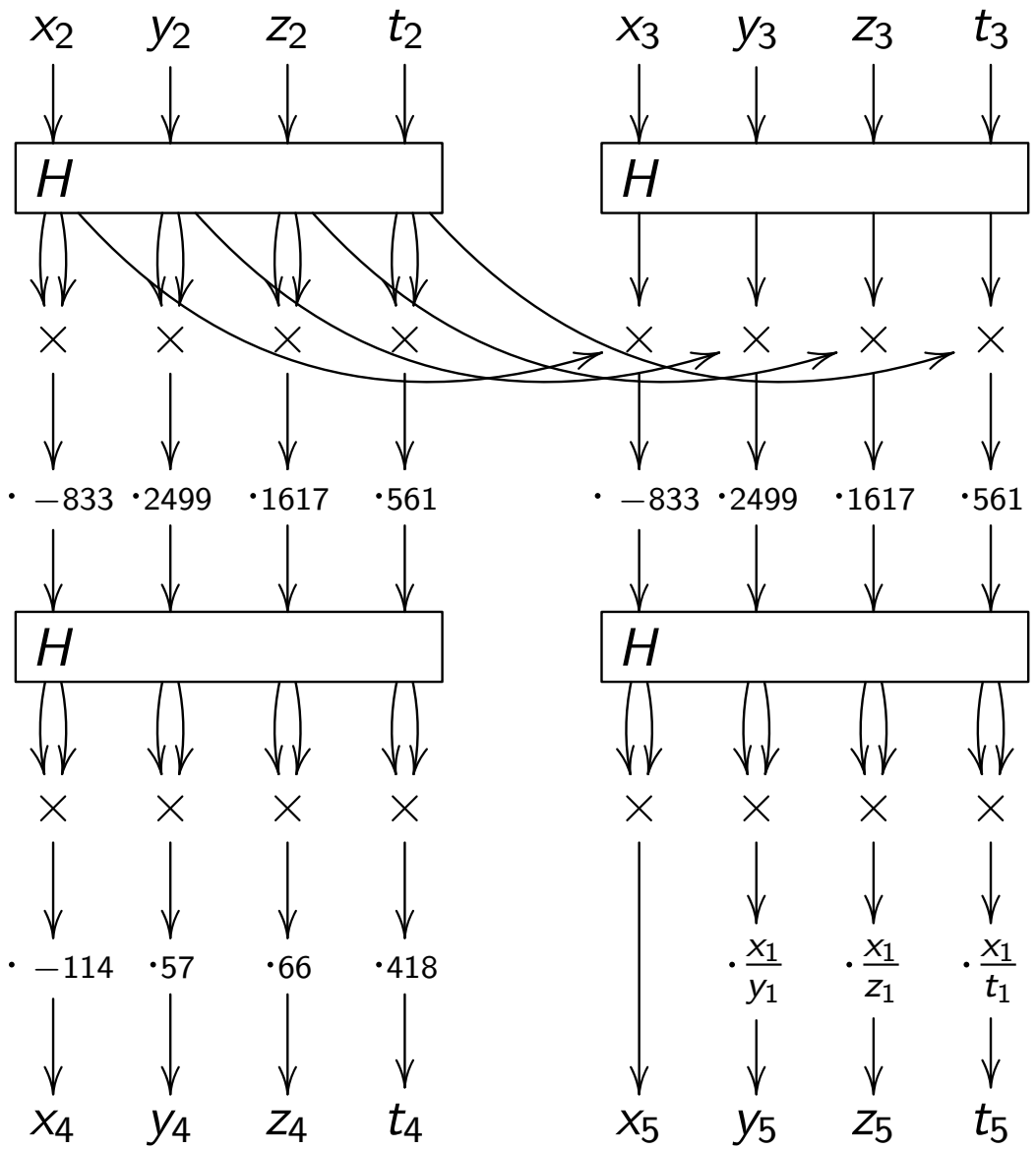


ECC Montgomery Ladder (4-way vectorization)

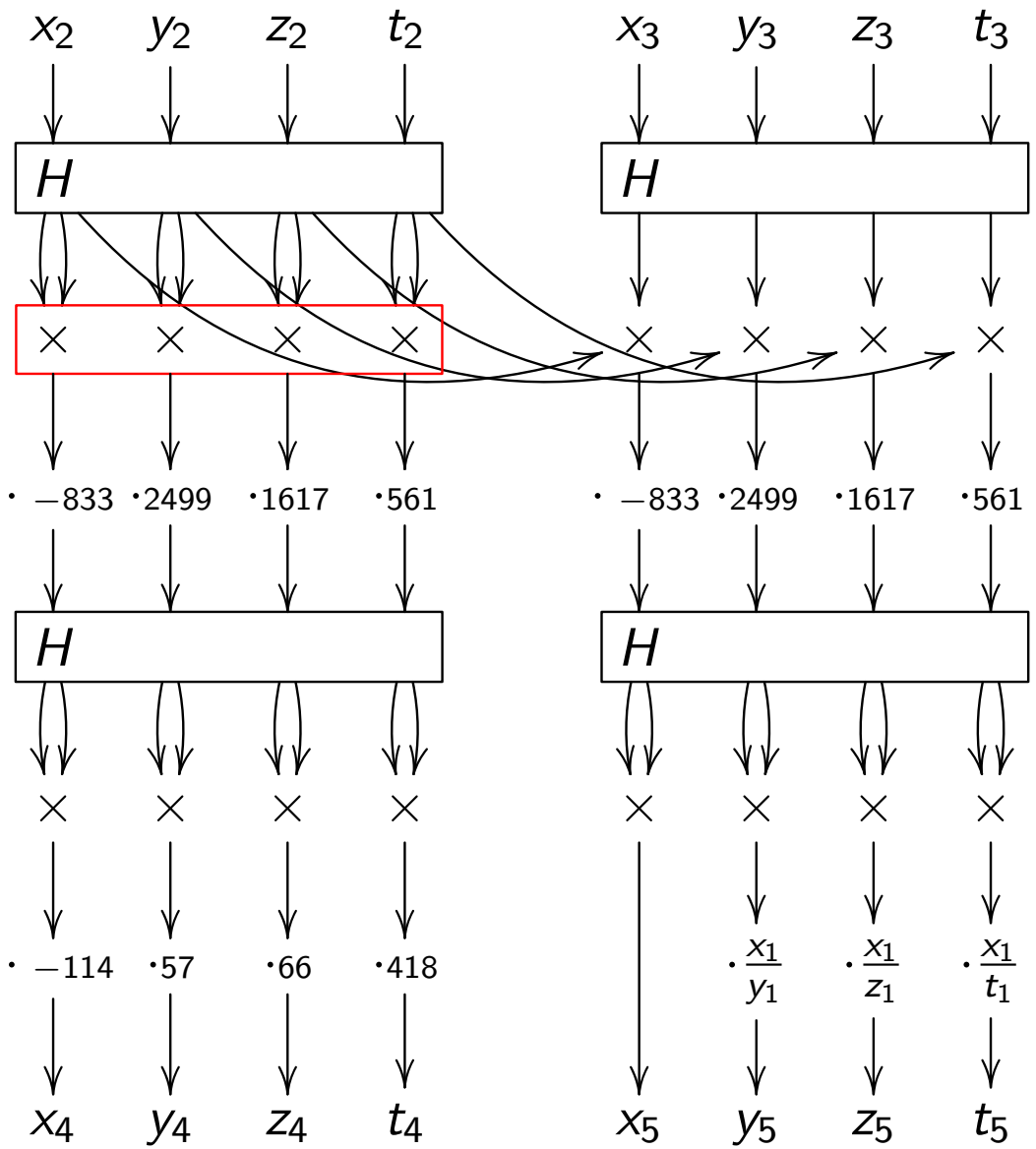
- match $+$ with $-$
- slow down squaring
- nothing to match and match $+$ and $-$
- slow down mult. by constant and squaring
- nothing to match $+$
- nothing to match $\times$



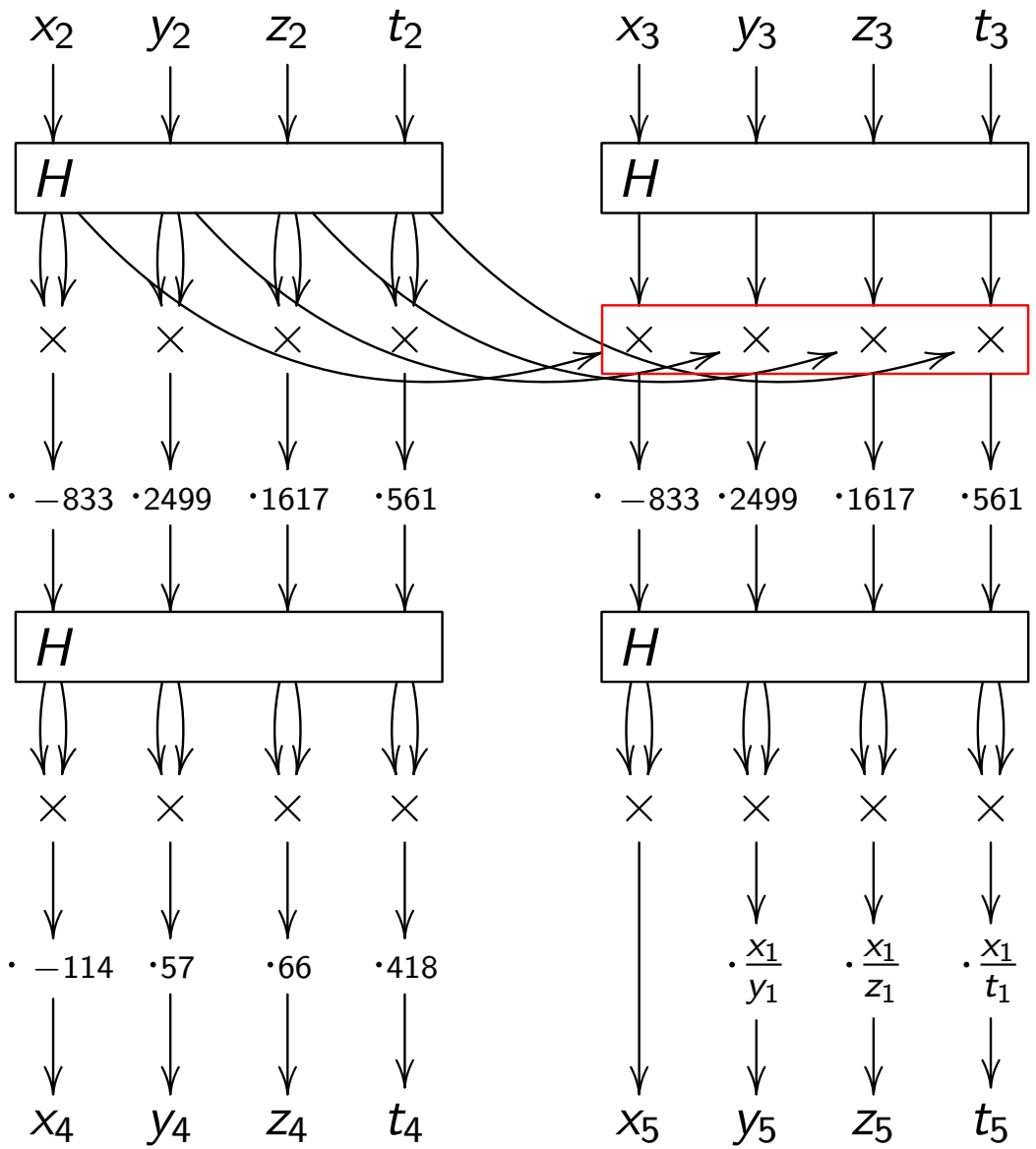
Squared Kummer Surface Ladder



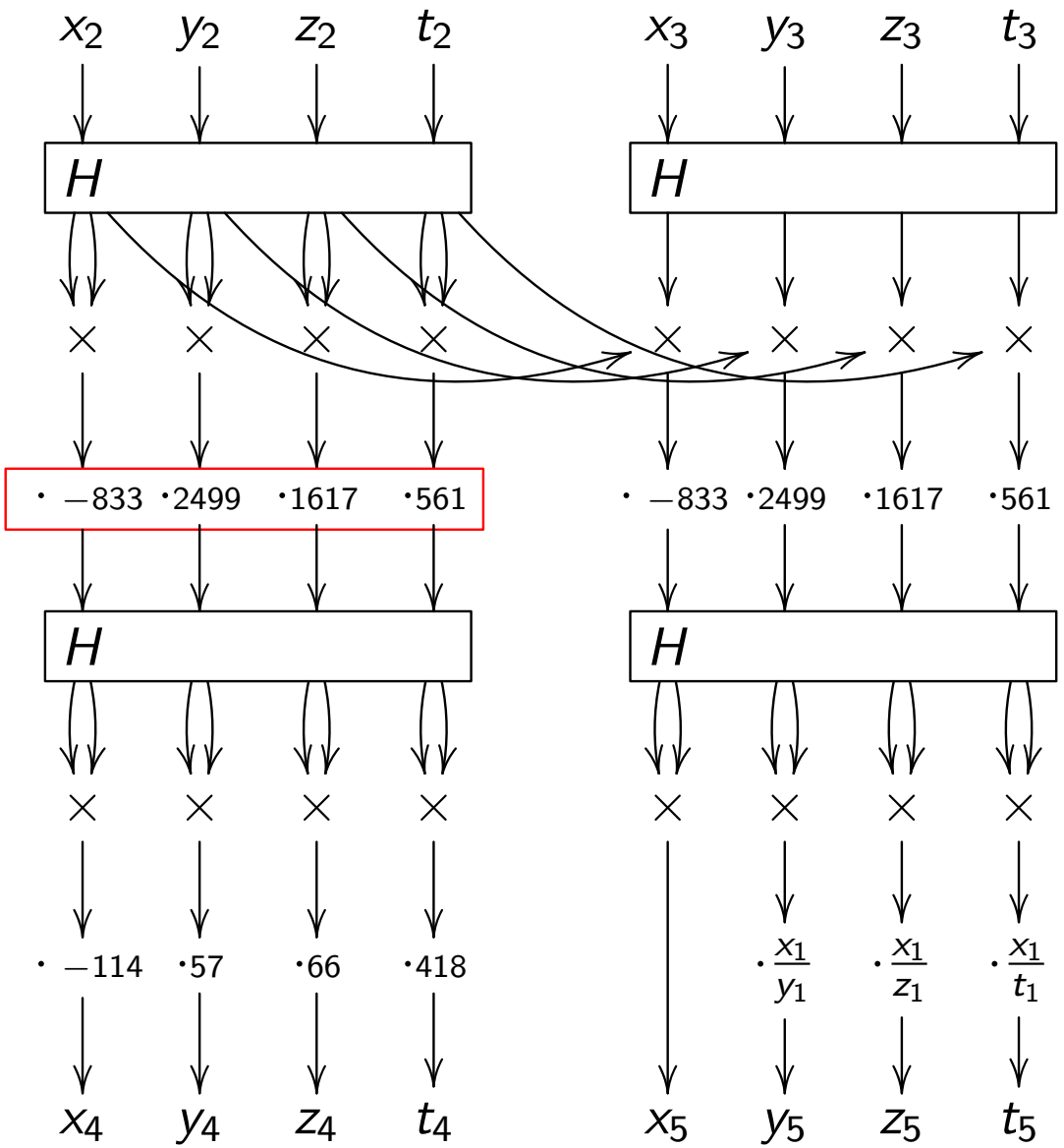
Squared Kummer Surface Ladder



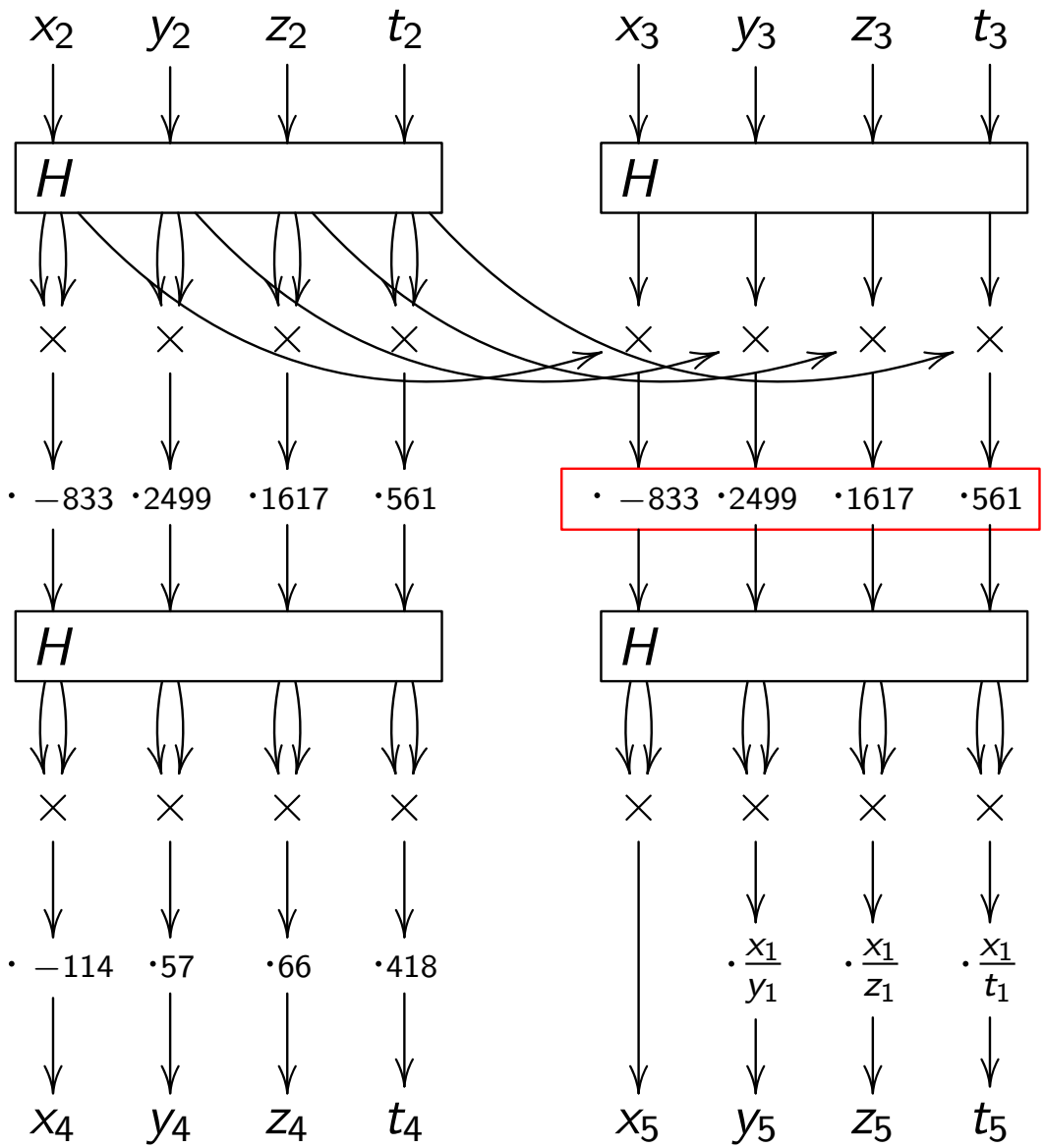
Squared Kummer Surface Ladder



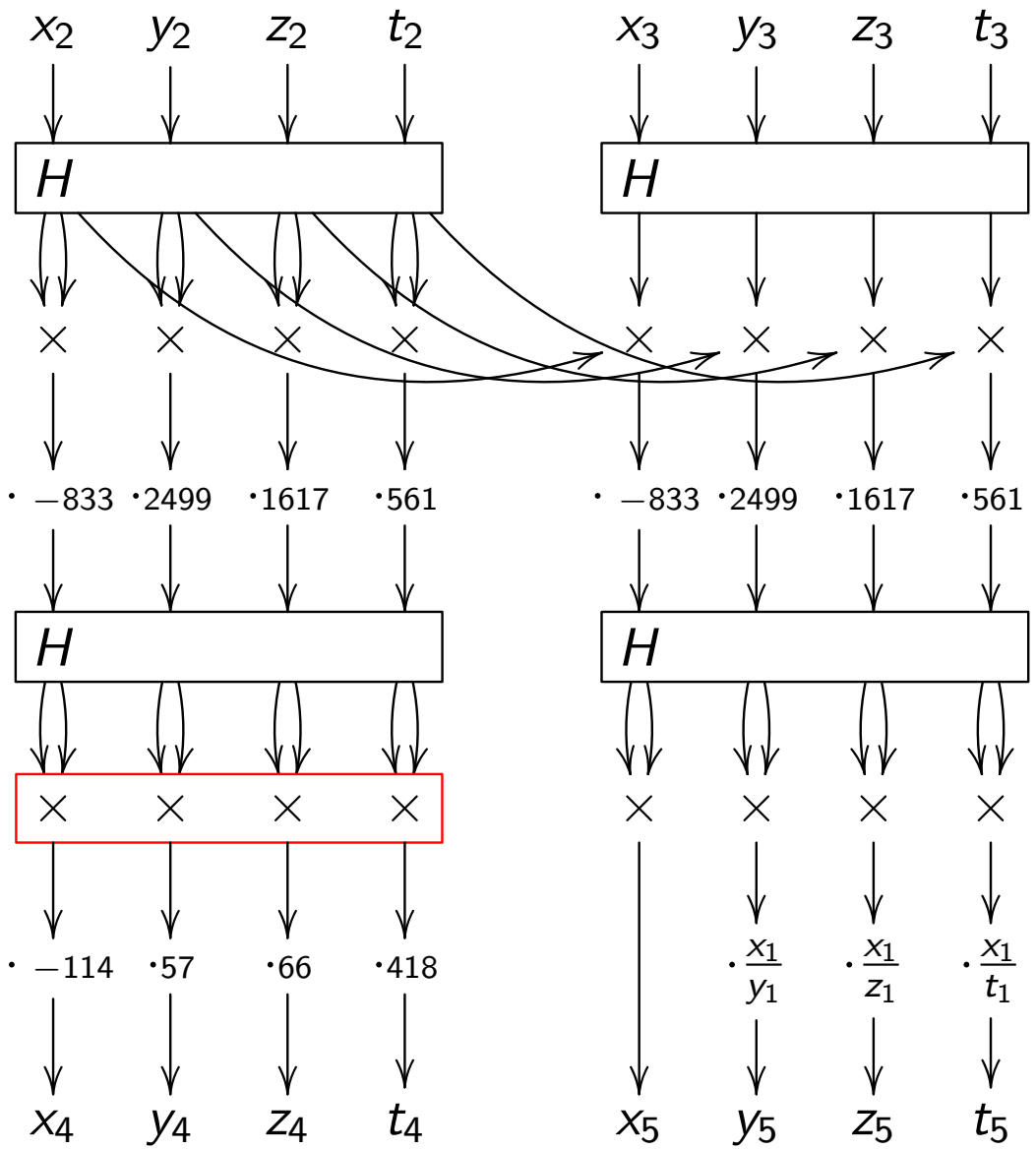
Squared Kummer Surface Ladder



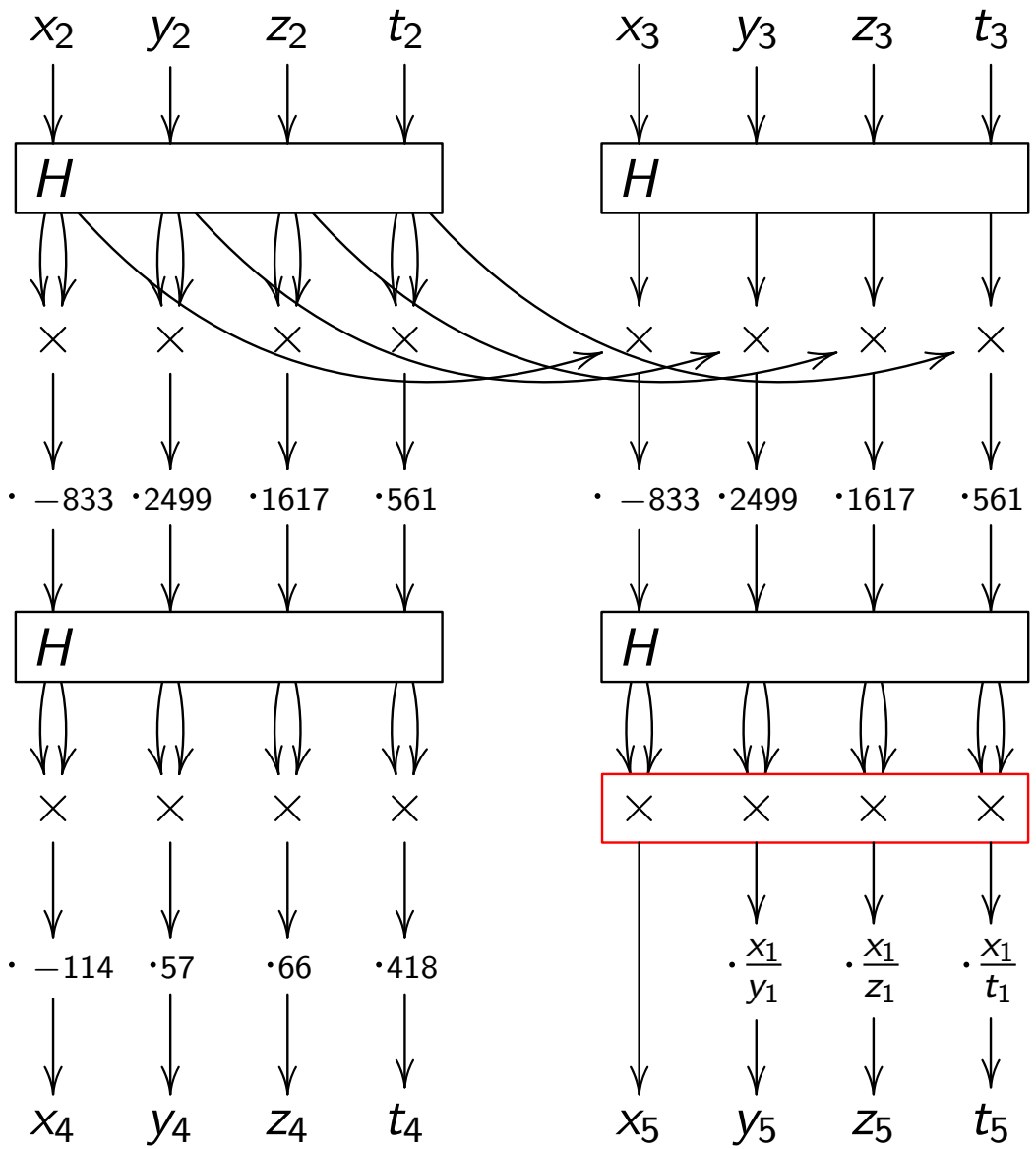
Squared Kummer Surface Ladder



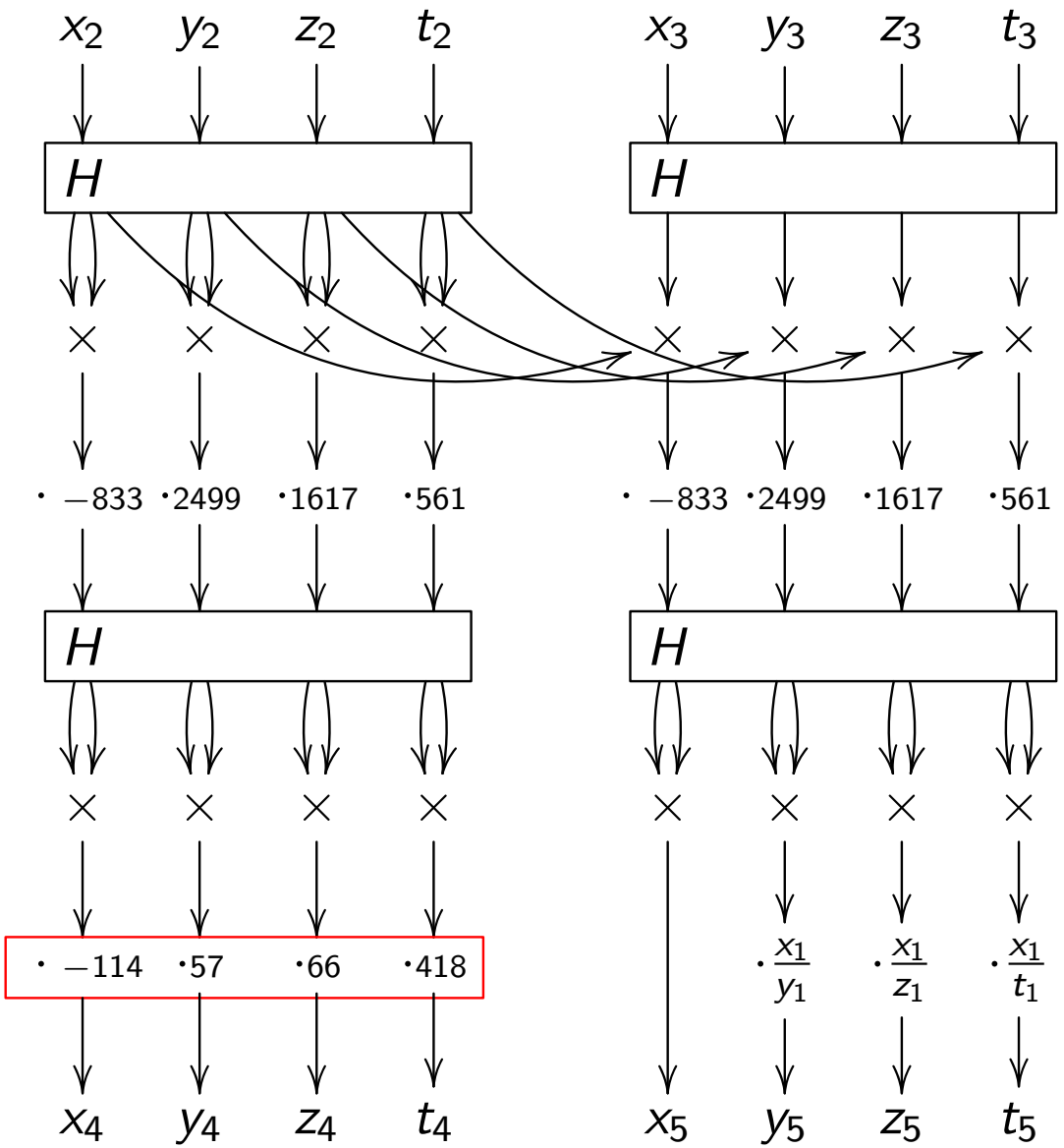
Squared Kummer Surface Ladder



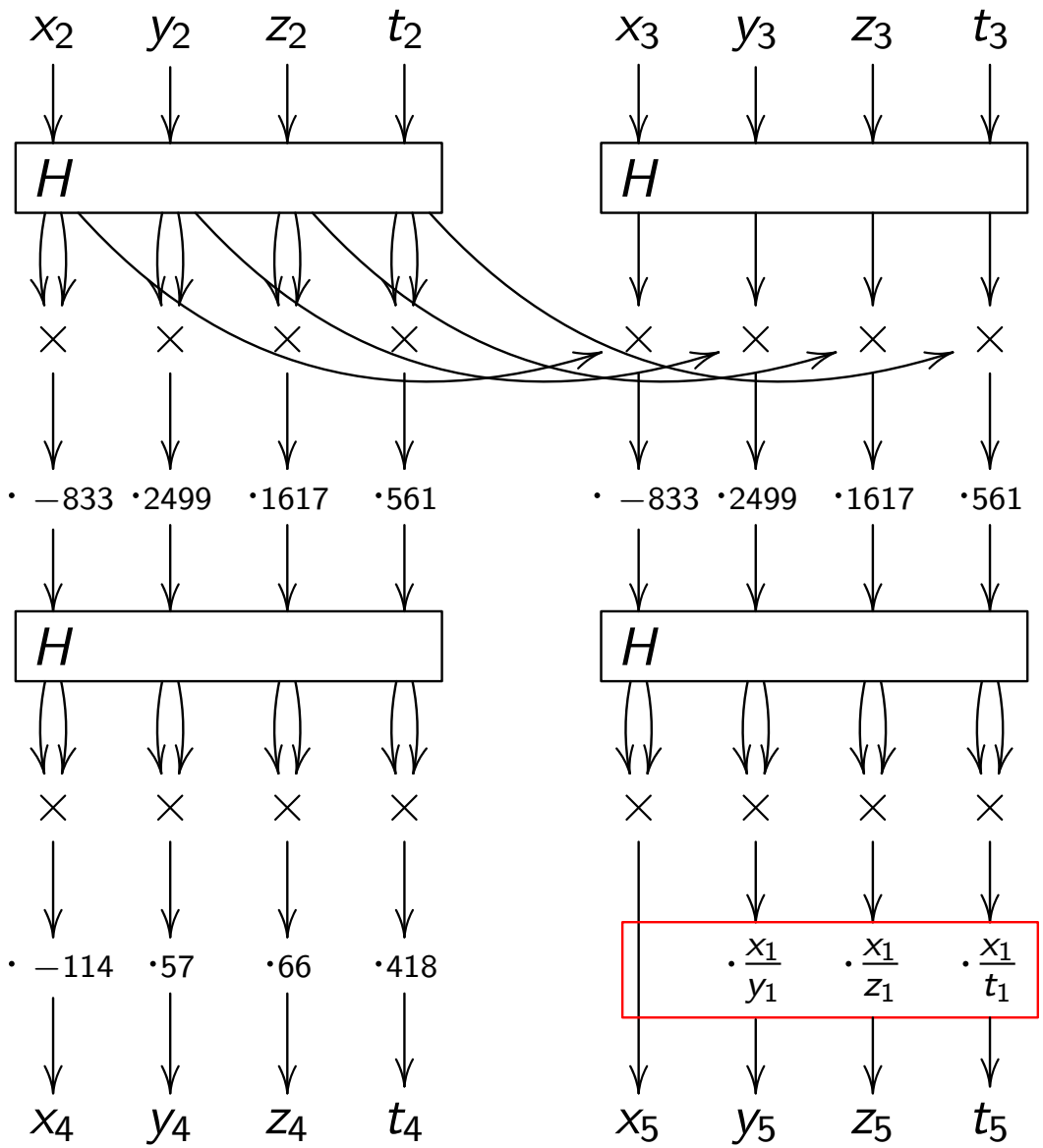
Squared Kummer Surface Ladder



Squared Kummer Surface Ladder



Squared Kummer Surface Ladder



Making It Run Really Fast

- maximize usage of available vector multipliers
- minimize cost from carries
  - use redundant representation
  - use non-integer radix, e.g., $2^{25.4}$ on Cortex-A8
  - do not perform full carry
  - do parallel carry chain
- eliminate redundancy inside field operations
  - precompute to reuse values $2f_1, 2f_2, 2f_3, 2f_4$
- minimize overhead from permutations
  - organize data to fit instruction format
- minimize permutations in H
  - use different order of input/output and change sign
  - e.g. we reduced 144 Sandy Bridge permutations to 36
- schedule instructions to keep CPU as busy as possible
- see paper for details

Performance Comparison

Arch	Cycles	g	Field	Source of software
A8-slow	497389	1	$2^{255}-19$	BeSc CHES 2012
A8-slow	305395	2	$2^{127}-1$	**BeChLaSc** Asiacrypt 2014
A8-fast	460200	1	$2^{255}-19$	BeSc CHES 2012
A8-fast	273349	2	$2^{127}-1$	**BeChLaSc** Asiacrypt 2014
Sandy	194036	1	$2^{255}-19$	BeDuLaScYa CHES 2011
Sandy	153000?	1	$2^{252}-2^{232}-1$	Hamburg
Sandy	137000?	1	$(2^{127}-5997)^2$	LoSi Asiacrypt 2012
Sandy	122716	2	$2^{127}-1$	BoCoHiLa Eurocrypt 2013
Sandy	119904	1	2^{254}	OlLóArRo CHES 2013
Sandy	96000?	1	$(2^{127}-5997)^2$	FaLoSá CT-RSA 2014
Sandy	92000?	1	$(2^{127}-5997)^2$	FaLoSá July 2014
Sandy	88916	2	$2^{127}-1$	**BeChLaSc** Asiacrypt 2014
Haswell	161648	1	$2^{255}-19$	BeDuLaScYa CHES 2011
Haswell	110740	2	$2^{127}-1$	BoCoHiLa Eurocrypt 2013
Haswell	61712	1	2^{254}	OlLóArRo CHES 2013
Haswell	60556	2	$2^{127}-1$	**BeChLaSc** Asiacrypt 2014

Part II

Curve41417: Karatsuba revisited
CHES 2014

Joint work with Daniel J. Bernstein and Tanja Lange

Performance Budget

- This part focuses on ARM Cortex-A8

Performance Budget

- This part focuses on ARM Cortex-A8
- OpenSSL `secp160r1` (security level only 2^{80})
  - least secure option supported by OpenSSL
  - $\approx$ 2.1 million cycles on A8-fast and A8-slow

Performance Budget

- This part focuses on ARM Cortex-A8
- OpenSSL `secp160r1` (security level only 2^{80})
  - least secure option supported by OpenSSL
  - $\approx$ 2.1 million cycles on A8-fast and A8-slow
- Curve25519 (security level 2^{125})
  - $\approx$ 0.5 million cycles on A8-fast and A8-slow

Performance Budget

- This part focuses on ARM Cortex-A8
- OpenSSL `secp160-r1` (security level only 2^{80})
  - least secure option supported by OpenSSL
  - $\approx$ 2.1 million cycles on A8-fast and A8-slow
- Curve25519 (security level 2^{125})
  - $\approx$ 0.5 million cycles on A8-fast and A8-slow
- Kummer (security level 2^{128})
  - $\approx$ 0.3 million cycles on A8-fast and A8-slow

Performance Budget

- This part focuses on ARM Cortex-A8
- OpenSSL `secp160-r1` (security level only 2^{80})
  - least secure option supported by OpenSSL
  - $\approx$ 2.1 million cycles on A8-fast and A8-slow
- Curve25519 (security level 2^{125})
  - $\approx$ 0.5 million cycles on A8-fast and A8-slow
- Kummer (security level 2^{128})
  - $\approx$ 0.3 million cycles on A8-fast and A8-slow
- Curve41417 (security level above 2^{200})
  - $\approx$ 1.6 million cycles on A8-fast
  - $\approx$ 1.8 million cycles on A8-slow

Design of Curve41417

- High-security elliptic curve (security level above 2^{200})
- Defined over prime field $\mathbf{F}_p$ where $p = 2^{414} - 17$
- In Edwards curve form

$$x^2 + y^2 = 1 + 3617x^2y^2$$

Design of Curve41417

- High-security elliptic curve (security level above 2^{200})
- Defined over prime field $\mathbf{F}_p$ where $p = 2^{414} - 17$
- In Edwards curve form

$$x^2 + y^2 = 1 + 3617x^2y^2$$

- Large prime-order subgroup (cofactor 8)
- IEEE P1363 criteria (large embedding degree, etc.)
- Twist secure, i.e., twist of Curve41417 also secure

ECC Arithmetic

- Mix coordinate systems:
  - doubling: projective X, Y, Z
  - addition: extended X, Y, Z, T

  (See https://hyperelliptic.org/EFD/)
- Scalar multiplication:
  - signed fixed windows of width $\omega = 5$
  - precompute $0P, 1P, 2P, \ldots, 16P$
    also multiply $d = 3617$ to T coordinate
  - special first doubling
  - compute T only before addition

Double-and-add

Input: $Q, n = (n_{i-1}, \ldots, n_0)_2$
Output: $R = nQ$

$R \leftarrow 1Q$

for $c = n_{i-2}$ **to** n_0
  $R \leftarrow 2R$
  if $c = 1$ **then**
    $R \leftarrow R + Q$

Return R

Example: Compute $9Q$

$9 = 1001_2$

$R = 1Q$

$n_2 = 0: \quad 2Q$

$n_1 = 0: \quad 4Q$

$n_0 = 1: \quad 8Q + Q = 9Q$

Windowing Method

Input: $Q, n = (n_{i-1}, \ldots, n_0)_2$
Output: $R = nQ$

$R \leftarrow (n_{i-1}, \ldots, n_{i-k})_2$

for $c = \lfloor (i-1)/\omega \rfloor$ **down to** 0
  $R \leftarrow 2^\omega R + S_c$

Return R

Note: ω = window width

Example: Compute $2345Q$

$2345 = \underline{10}\ \underline{01001}\ \underline{01001}_2$

$R = 10_2 = 2Q$

$01001_2 = 9$:

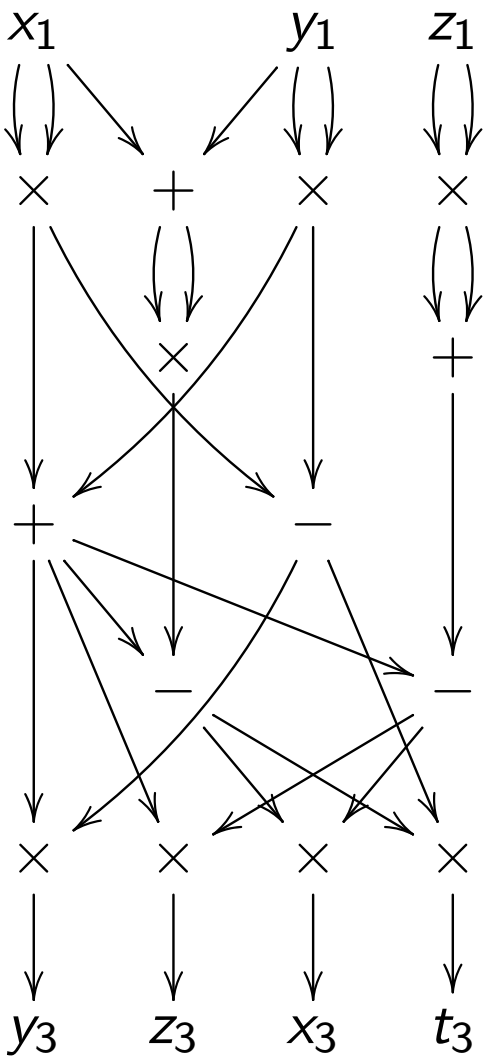
$2^5(2Q) + 9Q = 73Q$

$01001_2 = 9$:

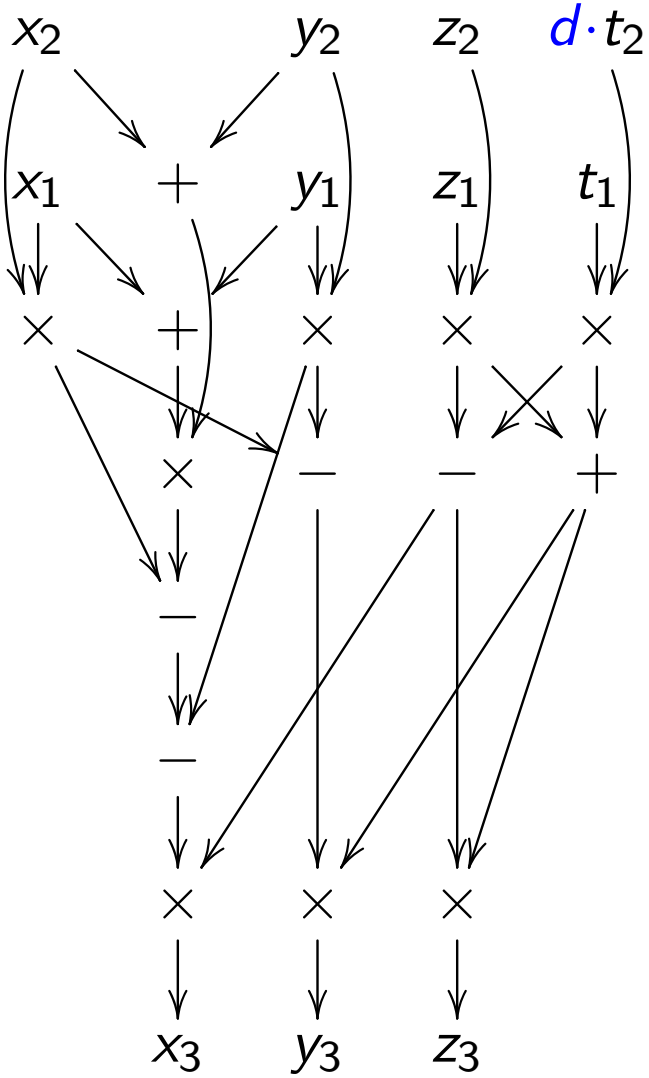
$2^5(72Q) + 9Q = 2345Q$

Point Operations

Point Doubling

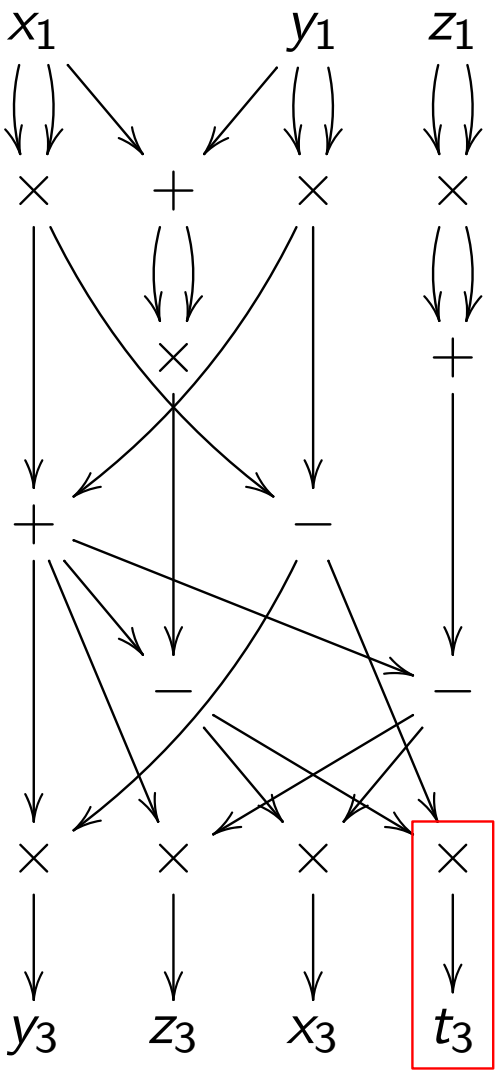


Point Addition

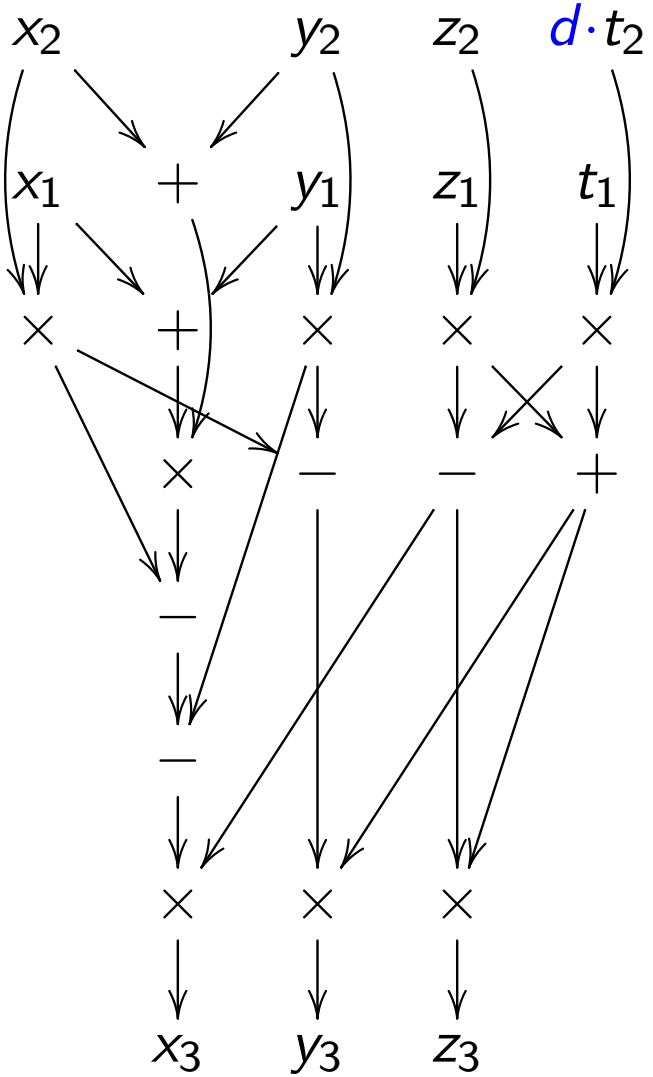


Point Operations

Point Doubling



Point Addition



Level of Karatsuba

- Karatsuba splits 1 $(2n \times 2n)$ into 3 $(n \times n)$

Level of Karatsuba

- Karatsuba splits 1 $(2n \times 2n)$ into 3 $(n \times n)$
- Zero-level Karatsuba (Schoolbook)
  e.g. for 16 limbs: $16 \times 16 = 256$

Level of Karatsuba

- Karatsuba splits 1 $(2n \times 2n)$ into 3 $(n \times n)$
- Zero-level Karatsuba (Schoolbook)
  e.g. for 16 limbs: $16 \times 16 = 256$
- One-level Karatsuba
  e.g.: $16 \times 16 \rightarrow 3 \cdot (8 \times 8) +$ some additions
  $= 192 +$ some additions

Level of Karatsuba

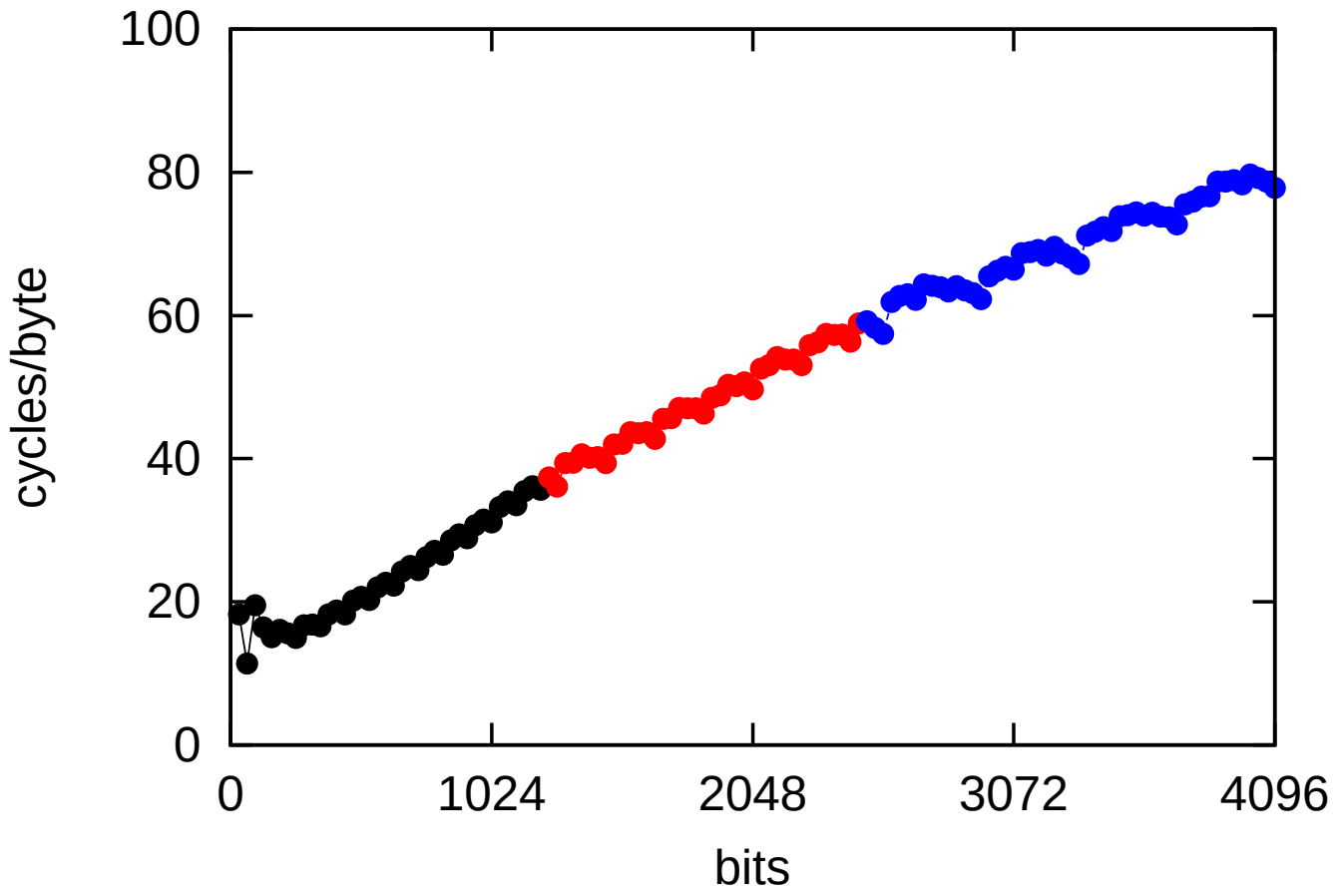
- Karatsuba splits 1 $(2n \times 2n)$ into 3 $(n \times n)$
- Zero-level Karatsuba (Schoolbook)
  e.g. for 16 limbs: $16 \times 16 = 256$
- One-level Karatsuba
  e.g.: $16 \times 16 \rightarrow 3 \cdot (8 \times 8) +$ some additions
  $= 192 +$ some additions
- Two-level Karatsuba
  e.g.: $3 \cdot (8 \times 8) \rightarrow 3 \cdot (3 \cdot (4 \times 4)) +$ even more additions
  $= 144 +$ even more additions

Level of Karatsuba

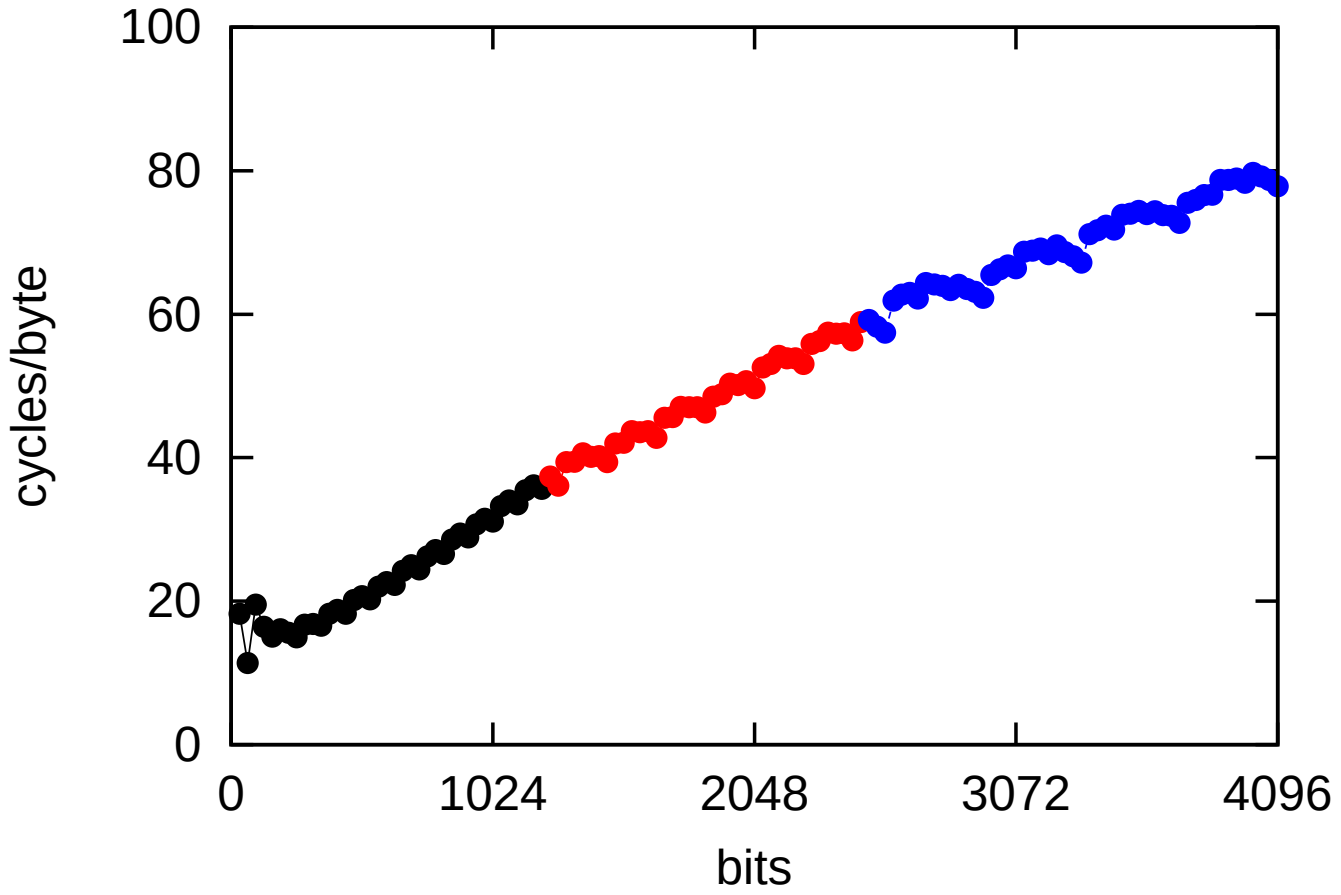
- Karatsuba splits 1 $(2n \times 2n)$ into 3 $(n \times n)$
- Zero-level Karatsuba (Schoolbook)
  e.g. for 16 limbs: $16 \times 16 = 256$
- One-level Karatsuba
  e.g.: $16 \times 16 \rightarrow 3 \cdot (8 \times 8) +$ some additions
  $= 192 +$ some additions
- Two-level Karatsuba
  e.g.: $3 \cdot (8 \times 8) \rightarrow 3 \cdot (3 \cdot (4 \times 4)) +$ even more additions
  $= 144 +$ even more additions

- What is the zero-level/one-level cutoff for number of limbs?

GMP's Cutoffs for Karatsuba

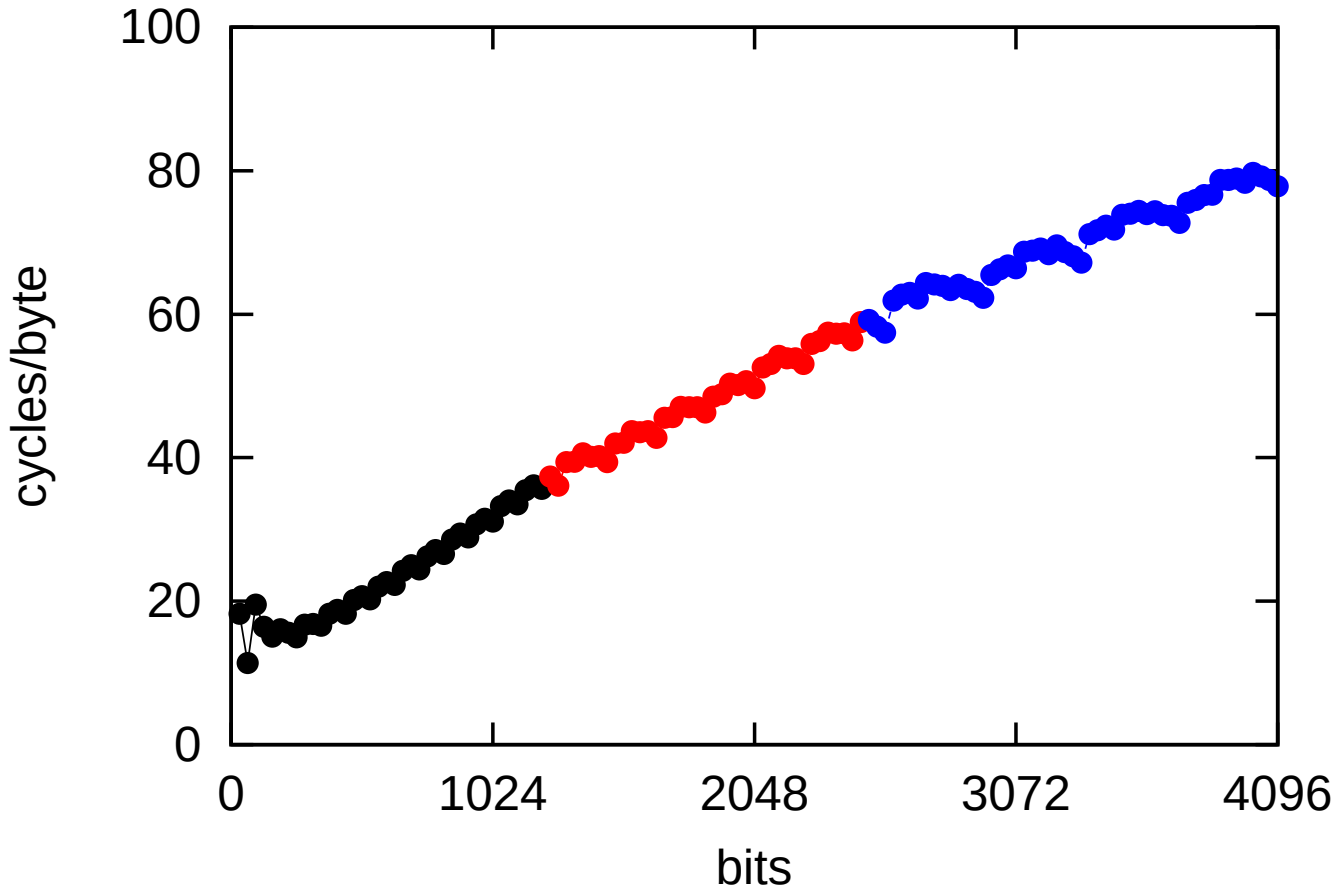


GMP's Cutoffs for Karatsuba



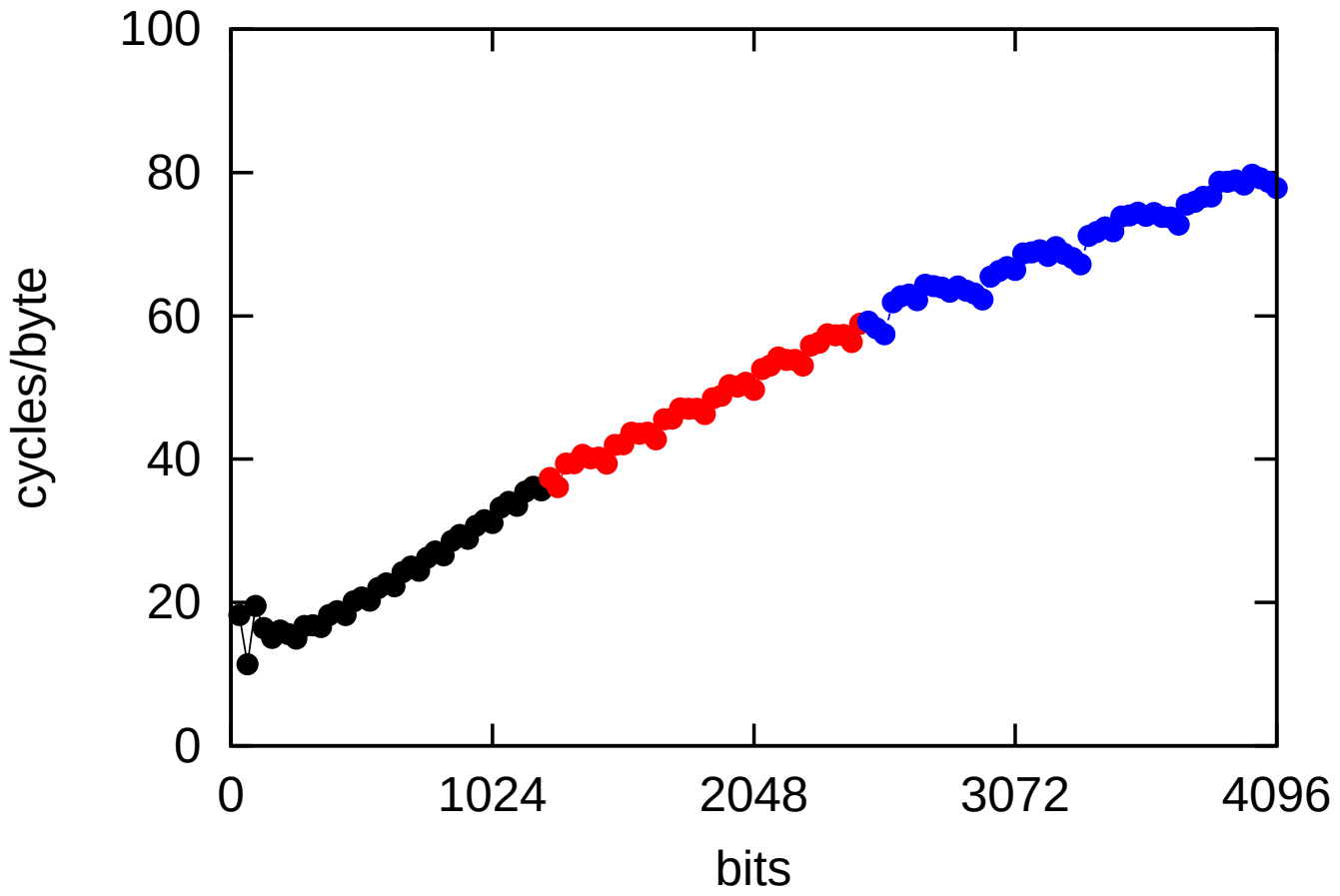
- GMP 6.0.0a library chooses 1248 bits on ARM Cortex-A8

GMP's Cutoffs for Karatsuba



- GMP 6.0.0a library chooses 1248 bits on ARM Cortex-A8
- We reduce cutoff via improvements to Karatsuba

GMP's Cutoffs for Karatsuba



- GMP 6.0.0a library chooses 1248 bits on ARM Cortex-A8
- We reduce cutoff via improvements to Karatsuba
- We reduce cutoff via redundant representation

Polynomial Multiplication

- Goal: Compute $P = AB$
  given $A = a_0 + a_1 t^n$ and $B = b_0 + b_1 t^n$
- Method 1: schoolbook
  $P = a_0 b_0 + (a_0 b_1 + a_1 b_0) t^n + a_1 b_1 t^{2n}$
- Method 2: Karatsuba ($8n-4$ additions)
  $P = a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) t^n + a_1 b_1 t^{2n}$
- Method 3: refined Karatsuba ($7n-3$ additions)
  $P = (a_0 b_0 - a_1 b_1 t^n)(1 - t^n) + (a_0 + a_1)(b_0 + b_1) t^n$

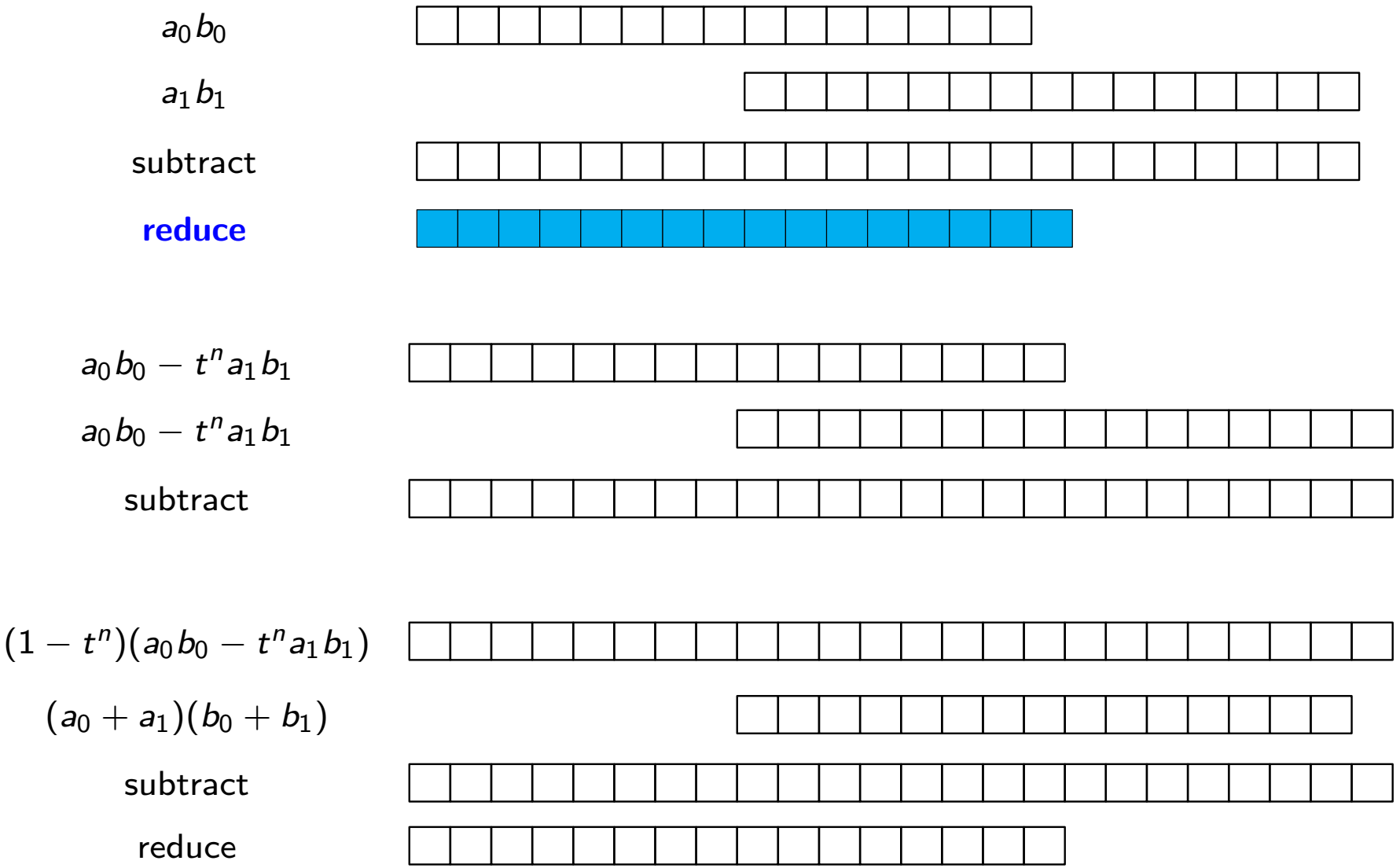
Polynomial Multiplication $\bmod Q$

- Goal: Compute $P = AB \bmod Q$
  given $A = a_0 + a_1 t^n$ and $B = b_0 + b_1 t^n$
- Method 1: schoolbook
  $P = a_0 b_0 + (a_0 b_1 + a_1 b_0) t^n + a_1 b_1 t^{2n} \bmod Q$
- Method 2: Karatsuba ($8n-4$ additions)
  $P = a_0 b_0 + ((a_0+a_1)(b_0+b_1) - a_0 b_0 - a_1 b_1) t^n + a_1 b_1 t^{2n} \bmod Q$
- Method 3: refined Karatsuba ($7n-3$ additions)
  $P = (a_0 b_0 - a_1 b_1 t^n)(1 - t^n) + (a_0 + a_1)(b_0 + b_1) t^n \bmod Q$

Polynomial Multiplication $\bmod Q$

- Goal: Compute $P = AB \bmod Q$
  given $A = a_0 + a_1 t^n$ and $B = b_0 + b_1 t^n$
- Method 1: schoolbook
  $P = a_0 b_0 + (a_0 b_1 + a_1 b_0) t^n + a_1 b_1 t^{2n} \bmod Q$
- Method 2: Karatsuba ($8n-4$ additions)
  $P = a_0 b_0 + ((a_0+a_1)(b_0+b_1) - a_0 b_0 - a_1 b_1) t^n + a_1 b_1 t^{2n} \bmod Q$
- Method 3: refined Karatsuba ($7n-3$ additions)
  $P = (a_0 b_0 - a_1 b_1 t^n)(1 - t^n) + (a_0 + a_1)(b_0 + b_1) t^n \bmod Q$
- Method 4: reduced refined Karatsuba ($6n-2$ additions) (new)
  $P = (a_0 b_0 - a_1 b_1 t^n \bmod Q)(1 - t^n) + (a_0 + a_1)(b_0 + b_1) t^n \bmod Q$

Reduced Refined Karatsuba



Cost Comparison (Karatsuba)

Level	Mult.	Add		Cost
		64-bit	32-bit	
0-level	256	15	0	$256+\ 8+\ 0 = 264$
1-level	192	59	16	$192+30+\ 4 = 226$
2-level	144	119	40	$144+60+10 = 214$
3-level	108	191	76	$108+96+19 = 223$

Note: use multiply-add instructions

Recall:

- 1 cycle per multiplication
- 0.5 cycle per 64-bit addition
- 0.25 cycle per 32-bit addition

Cost Comparison (refined Karatsuba)

Level	Mult.	Add		Cost
		64-bit	32-bit	
0-level	256	15	0	$256+\ 8+\ 0 = 264$
1-level	192	52	16	$192+26+\ 4 = 222$
2-level	144	103	40	$144+52+10 = 206$
3-level	108	166	76	$108+83+19 = 210$

Note: use multiply-add instructions

Recall:

1 cycle per multiplication

0.5 cycle per 64-bit addition

0.25 cycle per 32-bit addition

Cost Comparison (reduced refined Karatsuba)

Level	Mult.	Add		Cost
		64-bit	32-bit	
0-level	256	15	0	$256+\ 8+\ 0 = 264$
1-level	192	45	16	$192+23+\ 4 = 219$
2-level	144	96	40	$144+48+10 = 202$
3-level	108	159	76	$108+80+19 = 207$

Note: use multiply-add instructions

Recall:

1 cycle per multiplication

0.5 cycle per 64-bit addition

0.25 cycle per 32-bit addition

Performance Comparison

- OpenSSL

curve	# cycle on i.MX515	# cycle on Sitara
`secp160r1`	$\approx$ 2.1 million	$\approx$ 2.1 million
`nistp192`	$\approx$ 2.9 million	$\approx$ 2.8 million
`nistp224`	$\approx$ 4.0 million	$\approx$ 3.9 million
`nistp256`	$\approx$ 4.0 million	$\approx$ 3.9 million
`nistp384`	$\approx$ 13.3 million	$\approx$ 13.2 million
`nistp521`	$\approx$ 29.7 million	$\approx$ 29.7 million

- Curve41417 (security level above 2^{200})
  - $\approx$ 1.6 million cycles on A8-fast (FreeScale i.MX515)
  - $\approx$ 1.8 million cycles on A8-slow (TI Sitara)

Performance Comparison

- OpenSSL

curve	# cycle on i.MX515	# cycle on Sitara
`secp160r1`	$\approx$ 2.1 million	$\approx$ 2.1 million
`nistp192`	$\approx$ 2.9 million	$\approx$ 2.8 million
`nistp224`	$\approx$ 4.0 million	$\approx$ 3.9 million
`nistp256`	$\approx$ 4.0 million	$\approx$ 3.9 million
`nistp384`	$\approx$ 13.3 million	$\approx$ 13.2 million
`nistp521`	$\approx$ 29.7 million	$\approx$ 29.7 million

- Curve41417 (security level above 2^{200})
  - $\approx$ 1.6 million cycles on A8-fast (FreeScale i.MX515)
  - $\approx$ 1.8 million cycles on A8-slow (TI Sitara)

- Coming soon: Intel Haswell implementation with Sebastian Verschoor