New software speed records for cryptographic pairings

Michael Naehrig, Ruben Niederhagen, Peter Schwabe

Eindhoven University of Technology

Technische Universiteit
Eindhoven
University of Technology

July 8, 2010

HGI-Colloquium, Ruhr-Universität Bochum

Pairings

A protocol designer's point of view

- Let G_1, G_2, and G_3 be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$e : G_1 \times G_2 \to G_3$$

Pairings

A protocol designer's point of view

Technische Universiteit Eindhoven University of Technology

- Let G_1, G_2, and G_3 be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$e : G_1 \times G_2 \to G_3$$

- DLP should be hard in G_1, G_2, and G_3

Pairings
A protocol designer's point of view

- Let G_1, G_2, and G_3 be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$e : G_1 \times G_2 \to G_3$$

- DLP should be hard in G_1, G_2, and G_3
- Sometimes required: $G_1 = G_2$ (type-1 pairing)
- Sometimes requires: Efficient isomorphism $G_2 \to G_1$ (type-2)
- Sometimes required: **No** efficient isomorphism $G_2 \to G_1$ (type-3)

The Tate Pairing

A mathematical/algorithmic point of view

- Let E be an elliptic curve over $\mathbb{F}_q$
- Let $r \in \mathbb{N}$ be prime with $r \mid |E(\mathbb{F}_q)|$ and $r^2 \nmid |E(\mathbb{F}_q)|$
- Let $\gcd(r, q) = 1$ and $r \nmid (q-1)$
- Let k be the smallest positive integer such that $r \mid q^k - 1$
- k is called *embedding degree of E with respect to r*

The Tate pairing is a map

$$T_r : E[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r.$$

Technische Universiteit **Eindhoven** University of Technology

Representing elements of $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$

- Let's assume there is no element of order r^2 in $E(\mathbb{F}_{q^k})$
- Then it holds that $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \cong E[r]$

The Tate Pairing
A mathematical/algorithmic point of view

Representing elements of $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$

- Let's assume there is no element of order r^2 in $E(\mathbb{F}_{q^k})$
- Then it holds that $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \cong E[r]$

Consider the Tate pairing as a map

$$T_r : E[r] \times E[r] \to \mathbb{F}_{q^k}^* / (\mathbb{F}_{q^k}^*)^r.$$

Finding unique representatives in $\mathbb{F}_{q^k}^* / (\mathbb{F}_{q^k}^*)^r$.

- Results of the Tate pairing are equivalence classes
- In order to compare: Need unique representative
- $\mathbb{F}_{q^k}^* / (\mathbb{F}_{q^k}^*)^r$ and $\mu_r := \{x \in \mathbb{F}_{q^k} \mid x^r = 1\}$ are isomorphic
- Group isomorphism is given by exponentiation with $\frac{q^k-1}{r}$
- Apply group isomorphism in the end, obtain unique representative

The reduced Tate Pairing
A mathematical/algorithmic point of view

Finding unique representatives in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$.

- Results of the Tate pairing are equivalence classes
- In order to compare: Need unique representative
- $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$ and $\mu_r := \{x \in \mathbb{F}_{q^k} \mid x^r = 1\}$ are isomorphic
- Group isomorphism is given by exponentiation with $\frac{q^k-1}{r}$
- Apply group isomorphism in the end, obtain unique representative

Reduced Tate pairing:

$$e_r : E[r] \times E[r] \to \mu_r$$

$$(P, Q) \mapsto T_r(P, Q)^{\frac{q^k-1}{r}}$$

The reduced Tate Pairing

... on prime-order subgroups of $E[r]$

- The Frobenius endomorphism

$$\pi_q : E[r] \to E[r], (x, y) \mapsto (x^q, y^q)$$

  has eigenvalues 1 and q
- Eigenspace corresponding to eigenvalue 1 is $\ker(\pi_q - [1]) = E(\mathbb{F}_q)[r]$

The reduced Tate Pairing

... on prime-order subgroups of $E[r]$

- The Frobenius endomorphism

$$\pi_q : E[r] \to E[r], (x, y) \mapsto (x^q, y^q)$$

  has eigenvalues 1 and q

- Eigenspace corresponding to eigenvalue 1 is $\ker(\pi_q - [1]) = E(\mathbb{F}_q)[r]$
- Considering pairing on $E(\mathbb{F}_q)[r] \times E(\mathbb{F}_q)[r]$ always yields 1
- But: $\ker(\pi_q - [q])$ also has order r

The reduced Tate Pairing

... on prime-order subgroups of $E[r]$

- The Frobenius endomorphism

$$\pi_q : E[r] \to E[r], (x, y) \mapsto (x^q, y^q)$$

  has eigenvalues 1 and q
- Eigenspace corresponding to eigenvalue 1 is $\ker(\pi_q - [1]) = E(\mathbb{F}_q)[r]$
- Considering pairing on $E(\mathbb{F}_q)[r] \times E(\mathbb{F}_q)[r]$ always yields 1
- But: $\ker(\pi_q - [q])$ also has order r
- Denote $\ker(\pi_q - [1]) = E(\mathbb{F}_q)[r]$ by G_1
- Denote $\ker(\pi_q - [q]) \subset E(\mathbb{F}_{q^k})$ by G_2

Reduced Tate pairing for cryptography:

$$G_1 \times G_2 \to \mu_r$$

Towards computation of pairings

- I still have not said how the Tate pairing T_r is defined
- General definition requires a lot of background
- Much easier for the special case we will consider
- For the whole story read, e.g., Michael Naehrig's Ph.D. thesis

Towards computation of pairings

- I still have not said how the Tate pairing T_r is defined
- General definition requires a lot of background
- Much easier for the special case we will consider
- For the whole story read, e.g., Michael Naehrig's Ph.D. thesis
- No big surprise: Computation involves arithmetic in $\mathbb{F}_{q^k}^*$ and in $E(\mathbb{F}_q)$
- Only feasible for "small enough" k
- DLP in $\mathbb{F}_{q^k}^*$ only hard for "large enough" q^k

Towards computation of pairings

- I still have not said how the Tate pairing T_r is defined
- General definition requires a lot of background
- Much easier for the special case we will consider
- For the whole story read, e.g., Michael Naehrig's Ph.D. thesis
- No big surprise: Computation involves arithmetic in $\mathbb{F}_{q^k}^*$ and in $E(\mathbb{F}_q)$
- Only feasible for "small enough" k
- DLP in $\mathbb{F}_{q^k}^*$ only hard for "large enough" q^k
- Balance hardness of DLP in $E(\mathbb{F}_q)$ and $\mathbb{F}_{q^k}^*$
- But: Random curves have huge k

Barreto-Naehrig curves

- Let us consider pairings on the 128-bit security level
- r should have 256 bits, ideally $n = |E(\mathbb{F}_q)|$ is prime and has 256 bits, then take $r = n$
- $\mathbb{F}_{q^k}$ should have about 3072 bits (NIST), or about 3248 bits (ECRYPT II)
- Embedding degree should be 12 or 13 $(12 \times 256 = 3072)$

Barreto-Naehrig curves

- Let us consider pairings on the 128-bit security level
- r should have 256 bits, ideally $n = |E(\mathbb{F}_q)|$ is prime and has 256 bits, then take $r = n$
- $\mathbb{F}_{q^k}$ should have about 3072 bits (NIST), or about 3248 bits (ECRYPT II)
- Embedding degree should be 12 or 13 ($12 \times 256 = 3072$)
- Barreto-Naehrig curves (BN curves) are curves over $\mathbb{F}_p$ with prime $n = |E(\mathbb{F}_p)|$ and $k = 12$.
- Polynomial parametrization, $u \in \mathbb{Z}$:

$$p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$

$$n = n(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$

The reduced Tate pairing

Input: $P \in G_1, Q \in G_2, n = (1, n_{m-1}, \ldots, n_0)_2$
Output: $e_r(P, Q)$

```
R ← P
f ← 1
for (i ← m − 1; i ≥ 0; i − −) do
    Compute tangent line l at R
    R ← [2]R
    f ← f^2 l(Q)
    if (n_i = 1) then
        Compute line l through P and R
        R ← R + P
        f ← f l(Q)
    end if
end for
return f^((p^k − 1)/r)
```

Computing pairings over BN curves

Technische Universiteit Eindhoven University of Technology

The reduced Tate pairing

Input: $P \in G_1, Q \in G_2, n = (1, n_{m-1}, \ldots, n_0)_2$
Output: $e_r(P, Q)$

$R \leftarrow P$
$f \leftarrow 1$
for $(i \leftarrow m-1; i \geq 0; i--)$ **do**
  Compute tangent line l at R, compute $l(Q)$, $R \leftarrow [2]R$
  $f \leftarrow f^2 l(Q)$
  if $(n_i = 1)$ **then**
    Compute line l through P and R, compute $l(Q)$, $R \leftarrow R + P$
    $f \leftarrow f l(Q)$
  end if
end for
return $f^{\frac{p^k-1}{r}}$

Loop shortening

- "Miller loop" goes over bits of n
- n has about 256 bits, can we use shorter loop?

Loop shortening

- "Miller loop" goes over bits of n
- n has about 256 bits, can we use shorter loop?
- Many ideas, leading to eta, ate, r-ate, optimal ate pairing
- Shortest loop: optimal ate and r-ate pairing
- Looplength for BN-curves: $6u + 2$, about 66 bits
- In the following: consider optimal ate a_{opt}

Loop shortening

- "Miller loop" goes over bits of n
- n has about 256 bits, can we use shorter loop?
- Many ideas, leading to eta, ate, r-ate, optimal ate pairing
- Shortest loop: optimal ate and r-ate pairing
- Looplength for BN-curves: $6u + 2$, about 66 bits
- In the following: consider optimal ate a_{opt}
- Downside: Requires swapping arguments, curve arithmetic in $E(\mathbb{F}_{q^k})$
- Reason: Shortening based on Frobenius endomorphism, no effect in $E(\mathbb{F}_p)$
- Two additional line-function computations after the loop

Using twists

- Arithmetic in $E(\mathbb{F}_{q^k})$ is very much effort (recall: $k = 12!$)
- BN curve E has twist E' defined over $\mathbb{F}_{p^2}$
- $E'(\mathbb{F}_{p^2})$ has a subgroup of order n, call it G_2'
- There is an efficient isomorphism from G_2' to G_2
- Idea: Perform curve arithmetic on G_2'
- Compute line-function coefficients from points on G_2'
- Requires arithmetic only on $\mathbb{F}_{p^2}$

Resulting algorithm

Input: $Q' \in G_2', P \in G_1, l = 6u + 2 = (1, l_{m-1}, \ldots, l_0)_2$
Output: $a_{opt}(Q, P)$

$R' \leftarrow Q'$
$f \leftarrow 1$
for $(i \leftarrow m-1; i \geq 0; i--)$ **do**
  Compute tangent line l at R, compute $l(P)$, $R' \leftarrow [2]R'$
  $f \leftarrow f^2 l(P)$
  if $(l_i = 1)$ **then**
    Compute line l through Q and R, compute $l(P)$, $R' \leftarrow R' + Q'$
    $f \leftarrow f l(P)$
  end if
end for
Two final linefunction additions modifying f
return $f^{\frac{p^k - 1}{r}}$

Computing the final exponentiation

The easy part

- Decompose exponent $\frac{p^{12}-1}{n}$ in $(p^6-1)(p^2+1)((p^4-p^2+1)/n)$
- Exponentiation with p^6-1 is p^6 Frobenius and one inversion
- Exponentiation with p^2+1 is p^2 Frobenius and one multiplication
- $(p^6-1)(p^2+1)$ is called the "easy part"
- After the easy part: Inversion is conjugation, squaring also faster

Computing the final exponentiation

The hard part

- Remaining part: $(p^4 - p^2 + 1)/n$
- Algorithm by Scott, Benger, Charlemagne, Perez and Kachisa
- Idea: Exploit polynomial parametrization of p
- Requires 3 exponentiations with u
- Some more work: 13 multiplications, 4 squarings in $\mathbb{F}_{p^k}$

The Hamming-weight of u

- In the Miller loop, number of additions depends on Hamming-weight of $6u + 2$
- We can use NAF representation for the exponent

The Hamming-weight of u

Technische Universiteit **Eindhoven** University of Technology

- In the Miller loop, number of additions depends on Hamming-weight of $6u + 2$
- We can use NAF representation for the exponent
- Hard part of final exponentiation: 3 exponentiations with u
- Can use addition-subtraction chain

The Hamming-weight of u

- In the Miller loop, number of additions depends on Hamming-weight of $6u + 2$
- We can use NAF representation for the exponent
- Hard part of final exponentiation: 3 exponentiations with u
- Can use addition-subtraction chain

$\Longrightarrow$ Choice of u has huge impact on performance

An implementor's view

Technische Universiteit Eindhoven University of Technology

- All elliptic-curve arithmetic is on $E'(\mathbb{F}_{p^2})$
- Evaluating line functions at P yields elements of $\mathbb{F}_{p^{12}}$
- Evaluation means multiplication $\mathbb{F}_{p^2} \times \mathbb{F}_p$
- $\mathbb{F}_{p^{12}}$ is extension of $\mathbb{F}_{p^2}$

An implementor's view

- All elliptic-curve arithmetic is on $E'(\mathbb{F}_{p^2})$
- Evaluating line functions at P yields elements of $\mathbb{F}_{p^{12}}$
- Evaluation means multiplication $\mathbb{F}_{p^2} \times \mathbb{F}_p$
- $\mathbb{F}_{p^{12}}$ is extension of $\mathbb{F}_{p^2}$

$\Longrightarrow$ We can see the whole computation as sequence of operations in $\mathbb{F}_{p^2}$
Let's make $\mathbb{F}_{p^2}$ arithmetic as fast as possible

Modular arithmetic in $\mathbb{F}_p$

- Recall that p has a special shape

$$p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$

- Can we exploit this special shape for efficient modular arithmetic?
- Fan, Vercauteren, Verbauwhede (2009) demonstrate that the answer is "yes" for hardware implementations
- More efficient because it uses specially sized multipliers
- How about software implementations?

Polynomial representation
(Inspired by Bernstein's curve25519 paper)

Consider the ring $R = \mathbb{Z}[x] \cap \overline{\mathbb{Z}}[\sqrt{6}ux]$ and the element

$$\begin{aligned} P &= 36u^4x^4 + 36u^3x^3 + 24u^2x^2 + 6ux + 1 \\ &= (\sqrt{6}ux)^4 + \sqrt{6}(\sqrt{6}ux)^3 + 4(\sqrt{6}ux)^2 + \sqrt{6}(\sqrt{6}ux) + 1. \end{aligned}$$

Then $P(1) = p$.

Polynomial representation
(Inspired by Bernstein's curve25519 paper)

Consider the ring $R = \mathbb{Z}[x] \cap \overline{\mathbb{Z}}[\sqrt{6}ux]$ and the element

$$\begin{aligned} P &= 36u^4x^4 + 36u^3x^3 + 24u^2x^2 + 6ux + 1 \\ &= (\sqrt{6}ux)^4 + \sqrt{6}(\sqrt{6}ux)^3 + 4(\sqrt{6}ux)^2 + \sqrt{6}(\sqrt{6}ux) + 1. \end{aligned}$$

Then $P(1) = p$. Represent $f \in \mathbb{F}_p$ by a polynomial $F \in R$ as

$$\begin{aligned} F &= f_0 + f_1 \cdot \sqrt{6}(\sqrt{6}ux) + f_2 \cdot (\sqrt{6}ux)^2 + f_3 \cdot \sqrt{6}(\sqrt{6}ux)^3 \\ &= f_0 + f_1 \cdot (6u)x + f_2 \cdot (6u^2)x^2 + f_3 \cdot (36u^3)x^3 \end{aligned}$$

such that $F(1) = f$, or

$$f = f_0 + 6uf_1 + 6u^2 f_2 + 36u^3 f_3, f_i \in \mathbb{Z}$$

Multiplication and degree reduction

Polynomial multiplication of f and g yields 7 coefficients $t_0, \ldots, t_6$
Reduction mod p to $r_0, \ldots, r_3$:

$$r_0 \leftarrow t_0 - t_4 + 6t_5 - 2t_6$$
$$r_1 \leftarrow t_1 - t_4 + 5t_5 - t_6$$
$$r_2 \leftarrow t_2 - 4t_4 + 18t_5 - 3t_6$$
$$r_3 \leftarrow t_2 - t_4 + 2t_5 + 3t_6$$

Four coefficients are not enough

- 256-bit numbers in 4 coefficients: Each coefficient 64 bits
- Coefficients do not have exactly the same size
- Small multiples in the reduction are larger than 128 bits
- Easy to realize in hardware, not in software
- For software we need more coefficients

Four coefficients are not enough

- 256-bit numbers in 4 coefficients: Each coefficient 64 bits
- Coefficients do not have exactly the same size
- Small multiples in the reduction are larger than 128 bits
- Easy to realize in hardware, not in software
- For software we need more coefficients
- Idea: Consider $u = v^3$, use 12 coefficients $f_0, \ldots, f_{11}$

$$f = f_0 + 6vf_1 + 6v^2f_2 + 6v^3f_3 + 6v^4f_4 + 6v^5f_5 + 6v^6f_6 + \\ 36v^7f_7 + 36v^8f_8 + 36v^9f_9 + 36v_{10}f_{10} + 36v^{11}f_{11}$$

- v has about 21 bits, products have about 42 bits
- Double-precision floats have 53-bit mantissa
- Use double-precision floats, still some space to add up coefficients and compute small multiples

Reducing coefficients

- At some point the coefficients will *overflow* (become larger than 53 bits)
- Need to do coefficient reduction (carry)
- Carry from f_0 to f_1

  $c \leftarrow \text{round}(f_0/6v)$
  $f_0 \leftarrow f_0 - c \cdot 6v$
  $f_1 \leftarrow f_1 + c$
- Carry from f_1 to f_2

  $c \leftarrow \text{round}(f_1/v)$
  $f_1 \leftarrow f_1 - c \cdot v$
  $f_2 \leftarrow f_2 + c$
- $f_0 \in [-3v, 3v], f_1 \in [-v/2, v/2]$
- Carry from f_{11} goes to f_0, f_3, f_6, and f_9

Implementation on a Core 2 processor

- Use fast SIMD instructions `mulpd` and `addpd`
- 2 multiplications/ 2 additions in one instruction
- 1 `mulpd` and 1 `addpd` (and one `mov`) per cycle

Implementation on a Core 2 processor

- Use fast SIMD instructions `mulpd` and `addpd`
- 2 multiplications/ 2 additions in one instruction
- 1 `mulpd` and 1 `addpd` (and one `mov`) per cycle
- Problem: $\mathbb{F}_p$ arithmetic requires a lot of shuffeling, combining etc.

Implementation on a Core 2 processor

- Use fast SIMD instructions `mulpd` and `addpd`
- 2 multiplications/ 2 additions in one instruction
- 1 `mulpd` and 1 `addpd` (and one `mov`) per cycle
- Problem: $\mathbb{F}_p$ arithmetic requires a lot of shuffeling, combining etc.
- Solution: Implement arithmetic in $\mathbb{F}_{p^2}$
- Use schoolbook multiplication in $\mathbb{F}_{p^2}$ yielding 4 multiplications in $\mathbb{F}_p$
- Perform 2 multiplications in parallel using SIMD instructions

Implementation on a Core 2 processor

- Use fast SIMD instructions `mulpd` and `addpd`
- 2 multiplications/ 2 additions in one instruction
- 1 `mulpd` and 1 `addpd` (and one `mov`) per cycle
- Problem: $\mathbb{F}_p$ arithmetic requires a lot of shuffeling, combining etc.
- Solution: Implement arithmetic in $\mathbb{F}_{p^2}$
- Use schoolbook multiplication in $\mathbb{F}_{p^2}$ yielding 4 multiplications in $\mathbb{F}_p$
- Perform 2 multiplications in parallel using SIMD instructions
- $\mathbb{F}_p$ polynomial reduction after $\mathbb{F}_{p^2}$ polynomial reduction
- Only two $\mathbb{F}_p$ polynomial reduction and two coefficient reduction per multiplication in $\mathbb{F}_{p^2}$
- Those reductions also done in SIMD way

Detecting and avoiding overflows

- After each multiplication we need to reduce coefficients
- Sometimes also before a multiplication after several additions
- Problem: How to detect where?
- Need to detect overflow in the worst case

Detecting and avoiding overflows

- After each multiplication we need to reduce coefficients
- Sometimes also before a multiplication after several additions
- Problem: How to detect where?
- Need to detect overflow in the worst case
- Implement software in C
- Replace `double` with C++ class `CheckDouble`
- Perform arithmetic on values and in parallel on worst-case values
- Abort at overflow (allows backtrace in debugger)

Detecting and avoiding overflows

- After each multiplication we need to reduce coefficients
- Sometimes also before a multiplication after several additions
- Problem: How to detect where?
- Need to detect overflow in the worst case
- Implement software in C
- Replace `double` with C++ class `CheckDouble`
- Perform arithmetic on values and in parallel on worst-case values
- Abort at overflow (allows backtrace in debugger)
- Re-implement algorithms in assembly (qhasm)
- Would be good to have overflow checks in assembly

Parameters of our implementation

- We use $v = 1868033$, $u = v^3 = 6518589491078791937$
- 18 addition/subtraction steps in the Miller loop
- 12 multiplications for exponentiation with u
- p is congruent 3 mod 4, construct $\mathbb{F}_{p^2}$ as $\mathbb{F}_p[X]/(X^2+1)$

Results

Performance of dclxvi software

- Cycles on an Intel Core 2 Quad Q6600 (65 nm): 4,387,491 cycles
- Cycles on an Intel Core 2 Quad Q9550 (45 nm): 4,390,004 cycles

Performance of dclxvi software

- Cycles on an Intel Core 2 Quad Q6600 (65 nm): 4,387,491 cycles
- Cycles on an Intel Core 2 Quad Q9550 (45 nm): 4,390,004 cycles
- Cycles on an Intel Xeon E5504: 4,448,504 cycles
- Cycles on an AMD Phenom II X4 955: 4,774,059 cycles

Results

Technische Universiteit **Eindhoven** University of Technology

Performance of dclxvi software

- Cycles on an Intel Core 2 Quad Q6600 (65 nm): 4,387,491 cycles
- Cycles on an Intel Core 2 Quad Q9550 (45 nm): 4,390,004 cycles
- Cycles on an Intel Xeon E5504: 4,448,504 cycles
- Cycles on an AMD Phenom II X4 955: 4,774,059 cycles
- Comparison: Fastest published pairing benchmark before: 10,000,000 cycles on a Core 2 by Hankerson, Menezes, Scott, 2008
- Unpublished: 7,850,000 cycles on a Core 2 T5500 (Scott 2010)

Even faster pairings

New paper by Jean-Luc Beuchat, Jorge Enrique González Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya:
"High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves"
Claims: 2,630,000 cycles on a Core i7, 3,320,000 cycles on a Core 2

Even faster pairings

New paper by Jean-Luc Beuchat, Jorge Enrique González Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya:
"High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves"
Claims: 2,630,000 cycles on a Core i7, 3,320,000 cycles on a Core 2

Cycle counts on a Core 2 Q6600

	dclxvi	[BGM+10]
multiplication in $\mathbb{F}_{p^2}$	~ 656	~ 590
squaring in $\mathbb{F}_{p^2}$	~ 386	~ 481
optimal ate pairing	$\sim 4,390,000$	~ 3512000

Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_p$ and fast 64×64-bit integer multiplier.

Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_p$ and fast 64×64-bit integer multiplier.

Three reasons why we are slower

1. Restricted choice of u: More addition steps in Miller loop and exponentiation with u more expensive

Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_p$ and fast 64×64-bit integer multiplier.

Three reasons why we are slower

1. Restricted choice of u: More addition steps in Miller loop and exponentiation with u more expensive
2. Coefficient reductions take quite a bit of time ($\sim 450,000$ cycles)

Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_p$ and fast 64×64-bit integer multiplier.

Three reasons why we are slower

1. Restricted choice of u: More addition steps in Miller loop and exponentiation with u more expensive
2. Coefficient reductions take quite a bit of time ($\sim 450,000$ cycles)
3. Multiplication in $\mathbb{F}_{2^2}$ is slower (squaring is faster)

Which approach is better?

Highly depends on the architecture

- On the Core i7: Very clearly Montgomery arithmetic [BGM+10]
- On the AMD K11: again [BGM+10]
- On the Core 2: currently [BGM+10], but ... let's see

Which approach is better?

Technische Universiteit **Eindhoven** University of Technology

Highly depends on the architecture

- On the Core i7: Very clearly Montgomery arithmetic [BGM+10]
- On the AMD K11: again [BGM+10]
- On the Core 2: currently [BGM+10], but ... let's see
- Other microarchitectures or architectures?
  Mainly depends on performance of double-precision floating-point multiplication/addition vs. integer multiplication/addition
- Our approach is the fastest approach using double-precision floating-point arithmetic

References

Paper: http://cryptojedi.org/users/peter/#dclxvi
(has an error, will be updated soon)

Software: http://cryptojedi.org/crypto/#dclxvi
(public domain)