# New software speed records for cryptographic pairings 

Michael Naehrig, Ruben Niederhagen, Peter Schwabe

TU/e Technische Universiteit Eindhoven<br>University of Technology

August 9, 2010

Latincrypt 2010, Puebla, México

## Apologies

- Mistake in the paper as appeared in the proceedings
- Wrong choice of curve parameters
- Corrected in current version online
- Software (of course) also corrected

Thanks to Francisco for pointing this out.

## Pairings - the obligatory slide

- Let $G_{1}, G_{2}$, and $G_{3}$ be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$
e: G_{1} \times G_{2} \rightarrow G_{3}
$$

## Pairings - the obligatory slide

- Let $G_{1}, G_{2}$, and $G_{3}$ be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$
e: G_{1} \times G_{2} \rightarrow G_{3}
$$

- Different pairings derived from the Tate pairing


## Pairings - the obligatory slide

- Let $G_{1}, G_{2}$, and $G_{3}$ be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$
e: G_{1} \times G_{2} \rightarrow G_{3}
$$

- Different pairings derived from the Tate pairing
- For practical applications: based on elliptic-curve arithmetic
- Need "special" curves


## Pairings - the obligatory slide

- Let $G_{1}, G_{2}$, and $G_{3}$ be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$
e: G_{1} \times G_{2} \rightarrow G_{3}
$$

- Different pairings derived from the Tate pairing
- For practical applications: based on elliptic-curve arithmetic
- Need "special" curves
- For 128-bit security level: Barreto-Naehrig curves (BN curves)


## Pairings - the obligatory slide

- Let $G_{1}, G_{2}$, and $G_{3}$ be finite abelian groups.
- A pairing is a bilinear, nondegenerate map

$$
e: G_{1} \times G_{2} \rightarrow G_{3}
$$

- Different pairings derived from the Tate pairing
- For practical applications: based on elliptic-curve arithmetic
- Need "special" curves
- For 128-bit security level: Barreto-Naehrig curves (BN curves)
- Currently fastest: optimal ate pairing, $r$-ate pairing


## Optimal ate pairing over BN curves

Technische Universiteit Eindhoven
University of Technology
"Definition" for this talk
The ate pairing over a BN curve is a sequence of operations in a field $\mathbb{F}_{p^{2}}$

## Optimal ate pairing over BN curves

"Definition" for this talk
The ate pairing over a $B N$ curve is a sequence of operations in a field $\mathbb{F}_{p^{2}}$

- BN-curve construction: Find $u$ such that

$$
\begin{aligned}
& p=p(u)=36 u^{4}+36 u^{3}+24 u^{2}+6 u+1 \\
& n=n(u)=36 u^{4}+36 u^{3}+18 u^{2}+6 u+1
\end{aligned}
$$

are prime

- For 128 -bit security level: $n$ should have 256 bits, $p$ will also have 256 bits


## Optimal ate pairing over BN curves

"Definition" for this talk
The ate pairing over a $B N$ curve is a sequence of operations in a field $\mathbb{F}_{p^{2}}$

- BN-curve construction: Find $u$ such that

$$
\begin{aligned}
& p=p(u)=36 u^{4}+36 u^{3}+24 u^{2}+6 u+1 \\
& n=n(u)=36 u^{4}+36 u^{3}+18 u^{2}+6 u+1
\end{aligned}
$$

are prime

- For 128-bit security level: $n$ should have 256 bits, $p$ will also have 256 bits
- The choice of $u$ influences the sequence of operations in $\mathbb{F}_{p^{2}}$
- Details on high-level algorithms in the paper


## Optimal ate pairing over BN curves

"Definition" for this talk
The ate pairing over a BN curve is a sequence of operations in a field $\mathbb{F}_{p^{2}}$

- BN-curve construction: Find $u$ such that

$$
\begin{aligned}
& p=p(u)=36 u^{4}+36 u^{3}+24 u^{2}+6 u+1 \\
& n=n(u)=36 u^{4}+36 u^{3}+18 u^{2}+6 u+1
\end{aligned}
$$

are prime

- For 128-bit security level: $n$ should have 256 bits, $p$ will also have 256 bits
- The choice of $u$ influences the sequence of operations in $\mathbb{F}_{p^{2}}$
- Details on high-level algorithms in the paper

Question: Can we exploit the special shape of $p$ for faster arithmetic in $\mathbb{F}_{p}$ or $\mathbb{F}_{p^{2}}$ ?

## Exploiting the shape of $p$

Technische Universiteit Eindhoven
University of Technology

Answer 1, Scott, May 2009
"exploit the form of $p(x)$ for a faster modular arithmetic [...] Not really demonstrated successfully, (except recently in hardware?)."

## Exploiting the shape of $p$

Answer 1, Scott, May 2009
"exploit the form of $p(x)$ for a faster modular arithmetic [...] Not really demonstrated successfully, (except recently in hardware?).'"

Answer 2, Fan, Vercauteren, Verbauwhede, Sep. 2009
Faster $\mathbb{F}_{p}$-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves, presented at CHES 2009, exploiting the special structure of $p$ in hardware

## Exploiting the shape of $p$

Answer 1, Scott, May 2009
"exploit the form of $p(x)$ for a faster modular arithmetic [...] Not really demonstrated successfully, (except recently in hardware?).'"

Answer 2, Fan, Vercauteren, Verbauwhede, Sep. 2009
Faster $\mathbb{F}_{p}$-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves, presented at CHES 2009, exploiting the special structure of $p$ in hardware

How about software?

## Polynomial representation

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

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

Then $P(1)=p$.

## Polynomial representation

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

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

Then $P(1)=p$.

- Represent $f \in \mathbb{F}_{p}$ as polynomial in $R$ :

$$
\begin{aligned}
F & =f_{0}+f_{1} \cdot \sqrt{6}(\sqrt{6} u x)+f_{2} \cdot(\sqrt{6} u x)^{2}+f_{3} \cdot \sqrt{6}(\sqrt{6} u x)^{3} \\
& =f_{0}+f_{1} \cdot(6 u) x+f_{2} \cdot\left(6 u^{2}\right) x^{2}+f_{3} \cdot\left(36 u^{3}\right) x^{3}
\end{aligned}
$$

- Then: $f=F(1)$
- For implementation needs to store 4 coefficients $f_{0}, f_{1}, f_{2}, f_{3}$.


## Multiplication and degree reduction

Polynomial multiplication of $f$ and $g$ yields 7 coefficients $t_{0}, \ldots, t_{6}$ Reduction $\bmod p$ to $r_{0}, \ldots, r_{3}$ :

$$
\begin{aligned}
& r_{0} \leftarrow t_{0}-t_{4}+6 t_{5}-2 t_{6} \\
& r_{1} \leftarrow t_{1}-t_{4}+5 t_{5}-t_{6} \\
& r_{2} \leftarrow t_{2}-4 t_{4}+18 t_{5}-3 t_{6} \\
& r_{3} \leftarrow t_{3}-t_{4}+2 t_{5}+3 t_{6}
\end{aligned}
$$

## 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 $f_{0}, \ldots, f_{11}$

$$
\begin{aligned}
f= & f_{0}+6 v f_{1}+6 v^{2} f_{2}+6 v^{3} f_{3}+6 v^{4} f_{4}+6 v^{5} f_{5}+6 v^{6} f_{6}+ \\
& 36 v^{7} f_{7}+36 v^{8} f_{8}+36 v^{9} f_{9}+36 v_{10} f_{10}+36 v^{11} f_{11}
\end{aligned}
$$

## 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 $f_{0}, \ldots, f_{11}$

$$
\begin{aligned}
f= & f_{0}+6 v f_{1}+6 v^{2} f_{2}+6 v^{3} f_{3}+6 v^{4} f_{4}+6 v^{5} f_{5}+6 v^{6} f_{6}+ \\
& 36 v^{7} f_{7}+36 v^{8} f_{8}+36 v^{9} f_{9}+36 v_{10} f_{10}+36 v^{11} f_{11}
\end{aligned}
$$

- $v$ 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)


## 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 $f_{1}$
$c \leftarrow \operatorname{round}\left(f_{0} / 6 v\right)$
$f_{0} \leftarrow f_{0}-c \cdot 6 v$
$f_{1} \leftarrow f_{1}+c$
- Carry from $f_{1}$ to $f_{2}$
$c \leftarrow \operatorname{round}\left(f_{1} / v\right)$
$f_{1} \leftarrow f_{1}-c \cdot v$
$f_{2} \leftarrow f_{2}+c$


## 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 $f_{1}$
$c \leftarrow \operatorname{round}\left(f_{0} / 6 v\right)$
$f_{0} \leftarrow f_{0}-c \cdot 6 v$
$f_{1} \leftarrow f_{1}+c$
- Carry from $f_{1}$ to $f_{2}$
$c \leftarrow \operatorname{round}\left(f_{1} / v\right)$
$f_{1} \leftarrow f_{1}-c \cdot v$
$f_{2} \leftarrow f_{2}+c$
- $f_{0} \in[-3 v, 3 v], f_{1} \in[-v / 2, v / 2]$
- Carry from $f_{11}$ goes to $f_{0}, f_{3}, f_{6}$, and $f_{9}$


## Implementation on a Core 2 processor

- Use fast vector 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 vector 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 vector 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 $\mathbb{F}_{p}$
- For squaring in $\mathbb{F}_{p^{2}}$ use complex method: 2 multiplications in $\mathbb{F}_{p}$
- Perform $2 \mathbb{F}_{p}$ multiplications in parallel using vector instructions


## Implementation on a Core 2 processor

- Use fast vector 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 $\mathbb{F}_{p}$
- For squaring in $\mathbb{F}_{p^{2}}$ use complex method: 2 multiplications in $\mathbb{F}_{p}$
- Perform $2 \mathbb{F}_{p}$ multiplications in parallel using vector instructions
- $\mathbb{F}_{p}$ polynomial reduction after $\mathbb{F}_{p^{2}}$ polynomial reduction
- Only two $\mathbb{F}_{p}$ polynomial reductions and two coefficient reductions 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


## Performance of dclxvi software

- Cycles on an Intel Core 2 Quad Q6600 ( 65 nm ): 4,134,643 cycles
- Similar for other 64-bit Intel processors


## Performance of dclxvi software

- Cycles on an Intel Core 2 Quad Q6600 (65 nm): 4,134,643 cycles
- Similar for other 64-bit Intel processors
- Comparison: Fastest published pairing benchmark (on one core) 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 Beuchat, González Díaz, Mitsunari, Okamoto, Rodríguez-Henríquez, and Teruya:
"High-Speed Software Implementation of the Optimal Ate Pairing over
Barreto-Naehrig Curves"
Claims: $2,490,000$ cycles on a Core i7, 3,140,000 cycles on a Core 2 with Visual Studio 2008

## Even faster pairings

New paper by Beuchat, González Díaz, Mitsunari, Okamoto, Rodríguez-Henríquez, and Teruya:
"High-Speed Software Implementation of the Optimal Ate Pairing over
Barreto-Naehrig Curves"
Claims: $2,490,000$ cycles on a Core i7, $3,140,000$ cycles on a Core 2 with Visual Studio 2008
Cycle counts on a Core 2 Q6600 with gcc-4.3.3

|  | dclxvi | [BGM+10] |
| :--- | ---: | ---: |
| multiplication in $\mathbb{F}_{p^{2}}$ | $\sim 585$ | $\sim 588$ |
| squaring in $\mathbb{F}_{p^{2}}$ | $\sim 359$ | $\sim 487$ |
| optimal ate pairing | $\sim 4,135,000$ | $\sim 3,269,000$ |

## Why is our software slower?

Technische Universiteit
[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_{p}$ and fast $64 \times 64$-bit integer multiplier.
Three reasons why we are slower

## Why is our software slower?

Technische Universiteit
[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_{p}$ and fast $64 \times 64$-bit integer multiplier.
Three reasons why we are slower

1. Restricted choice of $u$ : Need more operations in $\mathbb{F}_{p^{2}}$

## Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_{p}$ and fast $64 \times 64$-bit integer multiplier.
Three reasons why we are slower

1. Restricted choice of $u$ : Need more operations in $\mathbb{F}_{p^{2}}$
2. Additional coefficient reductions take quite a bit of time ( $>400,000$ cycles)

## Why is our software slower?

[BGM+10] uses Montgomery arithmetic in $\mathbb{F}_{p}$ and fast $64 \times 64$-bit integer multiplier.
Three reasons why we are slower

1. Restricted choice of $u$ : Need more operations in $\mathbb{F}_{p^{2}}$
2. Additional coefficient reductions take quite a bit of time ( $>400,000$ cycles)
3. Multiplication is not (much) faster

## Why is our multiplication not faster?

- Fast multiplication (and squaring) was the target of our implementation
- Always need to perform even number of $\mathbb{F}_{p}$ multiplications
- Have to use schoolbook instead of Karatsuba in $\mathbb{F}_{p^{2}}$
- 4 instead of 3 multiplications in $\mathbb{F}_{p}$


## Why is our multiplication not faster?

- Fast multiplication (and squaring) was the target of our implementation
- Always need to perform even number of $\mathbb{F}_{p}$ multiplications
- Have to use schoolbook instead of Karatsuba in $\mathbb{F}_{p^{2}}$
- 4 instead of 3 multiplications in $\mathbb{F}_{p}$
- Using vector instructions still requires quite some shuffeling
- Overhead: 60 cycles per $\mathbb{F}_{p^{2}}$ multiplication


## Conclusion

- Fastest (current) implementation based on double-precision floating-point arithmetic exploits special $p$
- On Intel (and AMD) processors: integer-based approach (with Montgomery arithmetic) is faster
- But: several architectures have much faster double-precision floating-point than integer arithmetic


## References

TU/e

Paper: http://cryptojedi.org/users/peter/\#dclxvi Software: http://cryptojedi.org/crypto/\#dclxvi (public domain)

