Hardware Acceleration of the Tate Pairing in Characteristic Three

Philipp Grabher (Graz) and Dan Page (Bristol)

CHES 2005

Introduction

- Pairing based cryptography is a (fairly) new area:
  - Has provided new instantiations of Identity Based Encryption.
  - Has provided a wealth of new "hard problems" and proof techniques.
  - Has opened a new area for those interested in implementation.
- So far, most implementations have been done in software; our main aims before we started were:
  - Compare hardware polynomial and normal basis arithmetic in the finite fields $\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{3^{89}}$ respectivley.
    - Ideally we wanted same field size but curve selection and FPGA size bit us.
  - Evaluate the size and performance of a flexible pairing accelerator for use in constrained environments.
  - Ignore the fact that η-pairings, MNT curves and $\mathbb{F}_p$ arithmetic might be a more modern and better way to go :-)

Pairing Based Cryptography (1)

- For our purposes, the pairing is just a map between groups:

  $$e : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$$

  where we usually set $\mathbb{G}_1 = E(\mathbb{F}_q)$ and $\mathbb{G}_2 = \mathbb{F}_{q^k}$.
- The main interesting property of the map is termed bilinearity:

  $$e(a \cdot P, b \cdot Q) = e(P, Q)^{a \cdot b}$$

  which means we can play about with the exponents at will.
- To work in a useful way, the map also needs to be:
  - Non-degenerate, i.e. not all $e(P, Q) = 1$.
  - Computable, i.e. we can evaluate $e(P, Q)$ easily.
- In real applications we generally use the Tate or Weil pairing.

Pairing Based Cryptography (2)

- Such pairings were originally thought to only be useful in a destructive setting.
- Boneh-Franklin identity based encryption is perhaps the most interesting constructive use:
  - The trust authority or TA has a public key $P_{TA} = s \cdot P$ for a public value P and secret value s.
  - A users public key is calculated from the string ID using a hash function as $P_{ID} = H_1(ID)$.
  - A users secret key is calculated by the TA as $S_{ID} = s \cdot P_{ID}$.
- To encrypt M, select a random r and compute the tuple:

$$C = (r \cdot P, M \oplus H_2(e(P_{ID}, P_{TA})^r)).$$

- To decrypt $C = (U, V)$, we compute the result:

$$M = V \oplus H_2(e(S_{ID}, U)).$$

Pairing Based Cryptography (3)

- We are interested in the case where $q = 3^m$ and $k = 6$ since this is attractive from a parameterisation perspective.
- Along with the standard Miller-style BKLS algorithm, there are two closed-form algorithms in this case.
- Both compute $e(P, Q)$ with $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

The Duursma-Lee Algorithm

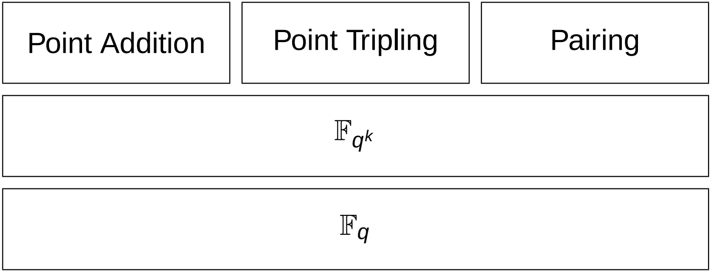
```
$f \leftarrow 1$
**for** $i = 1$ **upto** $m$ **do**
    $x_1 \leftarrow x_1^3$
    $y_1 \leftarrow y_1^3$
    $\mu \leftarrow x_1 + x_2 + b$
    $\lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
    $g \leftarrow \lambda - \mu\rho - \rho^2$
    $f \leftarrow f \cdot g$
    $x_2 \leftarrow x_2^{1/3}$
    $y_2 \leftarrow y_2^{1/3}$
**return** $f^{q^3 - 1}$
```

The Kwon-BGOS Algorithm

```
$f \leftarrow 1$
$x_2 \leftarrow x_2^3$
$y_2 \leftarrow y_2^3$
$d \leftarrow mb$
**for** $i = 1$ **upto** $m$ **do**
    $x_1 \leftarrow x_1^9$
    $y_1 \leftarrow y_1^9$
    $\mu \leftarrow x_1 + x_2 + d$
    $\lambda \leftarrow y_1 y_2 \sigma - \mu^2$
    $g \leftarrow \lambda - \mu\rho - \rho^2$
    $f \leftarrow f^3 \cdot g$
    $y_2 \leftarrow -y_2$
    $d \leftarrow d - b$
**return** $f^{q^3 - 1}$
```

Hardware Implementation (1)

- We need quite a few different operations:
  - $E(\mathbb{F}_q)$: Addition, Tripling, Scalar Multiplication.
  - $\mathbb{F}_q$: Addition, Multiplication, Inversion, Cube, Cube Root.
  - $\mathbb{F}_{q^k}$: Addition, Multiplication, Inversion, Cube.
- Everything depends on high-performance $\mathbb{F}_q$ arithmetic.
  - We approach is to implement only $\mathbb{F}_q$ arithmetic in hardware.
  - One can obviously get some different results by exploiting the parallelism in $\mathbb{F}_{q^k}$ or by building a dedicated pairing circuit.

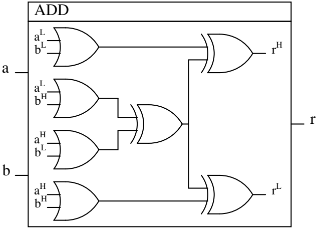


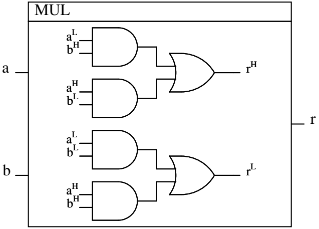
Hardware Implementation (2)

- In either basis, our field elements are polynomials with coefficients in $\mathbb{F}_3$.
- We take the now conventional approach of representing the i-th coefficient a_i as two bits:

$$\begin{array}{rcccc} a_i^L & = & a_i & \text{mod} & 2 \\ a_i^H & = & a_i & \text{div} & 2 \end{array}$$

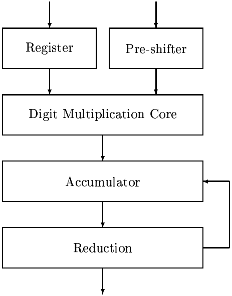
and then constructing basic arithmetic cells using a fairly low-cost arrangement of logic gates:





Hardware Implementation (4)

- In a polynomial basis, the multiplication $c = a \cdot b$ is performed by normal polynomial multiplication and reduction.
- We use a digit-wise rather than bit-wise multiplier design:



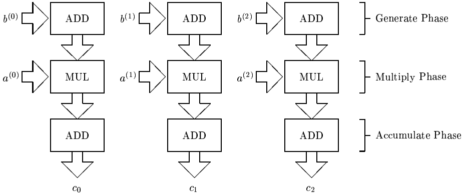
- We were able to fit a digit-size of 4 onto our experimental platform.

Hardware Implementation (4)

- In a normal basis, the multiplication $c = a \cdot b$ is performed according to the formula:

$$c_k = \sum_{i=0}^{m-1} a_{k+i} \cdot \sum_{j=0}^{m-1} M_{i,j} \cdot b_{k+j}$$

- The matrix M essentially determines how reduction works, it is very sparse so the whole operation is fairly efficient.
- The structure of the multiplier allows a similar digit-wise approach, we used a digit-size of 2:



Hardware Implementation (4)

- In a polynomial basis, cubing can be calculated in a similar way to squaring in characteristic two:
- That is, we expand the element using the identity:

$$(a_i x^i)^3 = a_i^3 x^{3i} = a_i x^{3i}$$

- Because of reduction, the cube operation dominates critial path of design since unreduced element is large.
- Cube root can be calculated using the method of Barreto, for our field $u = 32$ and $v = 5$:

$$\begin{aligned}
t_0 &= \textstyle\sum_{i=0}^{u} a_{3i} x^i \\
t_1 &= \textstyle\sum_{i=0}^{u-1} a_{3i+1} x^i \\
t_2 &= \textstyle\sum_{i=0}^{u-1} a_{3i+2} x^i \\
\sqrt[3]{a} &= t_0 + t_1^{\ll 2u+1} - t_1^{\ll u+v+1} + t_1^{\ll 2v+2} - 2t_2^{\ll u+1} - 2t_2^{\ll v+1}
\end{aligned}$$

which turns out to be quite efficient.

Hardware Implementation (5)

- In a normal basis, cube and cube root are just cyclic shifts of the coefficients:

$$\begin{aligned} a^3 &= (a_{m-1}, a_0, \ldots, a_{m-3}, a_{m-2}), \\ \sqrt[3]{a} &= (a_1, a_2, \ldots, a_{m-1}, a_0). \end{aligned}$$

- This was the whole point of investigating their use:
  - Cube is used extensively throughout point and pairing arithmetic.
  - Efficient cube root it vital for Duursma-Lee algorithm.

Hardware Implementation (6)

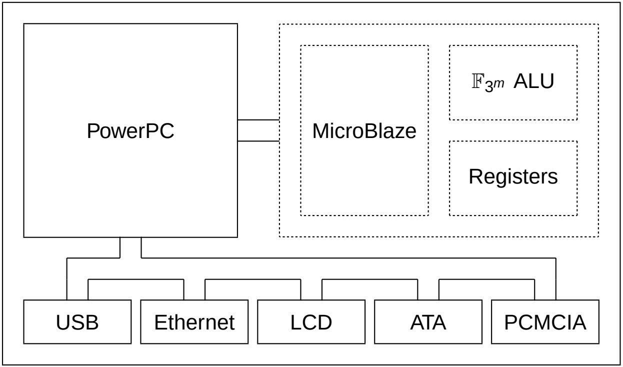
- Inversion was always going to be unpleasant:
  - Fortunately we only need it once to perform the final powering which computes the result f^{q^3-1}.
  - This computation is decomposed into $f^{3^{3m}} \cdot f^{-1}$.
  - The field representation means we only need one inversion in $\mathbb{F}_q$ (and some extra operations) to invert in $\mathbb{F}_{q^k}$.
- Since it is only used once, we didn't feel extra hardware was worthwhile.
- Could have used a variant of the binary EEA, but instead resorted to powering:

$$a^{-1} = a^{3^m-2}$$

- This turns out to be not too bad but can obviously be improved on depending on the constraints imposed.

Results (1)

- Used a Xilinx ML300 prototyping device for implementation.
- Essentially, we put an embedded processor and $\mathbb{F}_{3^m}$ ALU on the Virtex-II PRO FPGA.
- The hope was to mimic the kind of architecture in a real processor design.



Results (2)

$\mathbb{F}_{3^{97}}$ in Polynomial Basis

	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 150 MHz
Add	112	3	1	-	-
Subtract	112	3	1	-	-
Multiply	946	28	1	-	-
Cube	128	3	1	-	-
Cube Root	115	3	1	-	-
Pairing					
Duursma-Lee	-	59946	7857	3746.6μs	399.4μs
Kwon	-	64602	9409	4037.6μs	430.7μs
Powering	-	4941	397	308.8μs	32.9μs
Total	4481	-	-	-	-

$\mathbb{F}_{3^{89}}$ in Normal Basis

	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 85 MHz
Add	102	3	1	-	-
Subtract	102	3	1	-	-
Multiply	1505	48	1	-	-
Cube	0	3	1	-	-
Cube Root	0	3	1	-	-
Pairing					
Duursma-Lee	-	89046	7857	5563.3μs	1047.6μs
Kwon	-	93702	9409	5856.3μs	1102.4μs
Powering	-	7941	397	496.3μs	93.4μs
Total	4233	-	-	-	-

Conclusions

- We can comfortably compute the pairing in under a second even at low clock speeds.
- There wasn’t a lot of advantage from the normal basis arithmetic:
  - Cube and cube root are cheap but multiplier is expensive.
  - The polynomial basis cube root method of Baretto is single-cycle.
  - Finding suitable curves and so on is a nightmare ...
  - Using the Kwon-BGOS method seems a better choice.
- There is plenty of scope for miniaturisation given performance margin:
  - Using Kwon-BGOS removes need for cube-root hardware.
  - Can adjust multiplier digit size, maybe even use a bit-wise design.
  - Share addition logic between adder and multiplier.
  - Reduce storage size by improving register allocation or introduce spillage into main memory.