Hardware Implementation of Pairings

Francisco Rodríguez-Henríquez
CINVESTAV-IPN, Mexico City, Mexico

Joint work with:
- Jean-Luc Beuchat: LCIS, University of Tsukuba, Japan
- Nicolas Brisebarre: Arénaire, LIP, ÉNS Lyon, France
- Jérémie Detrey: Caramel, INRIA Nancy Grand-Est, France
- Nicolas Estibals: Caramel, INRIA Nancy Grand-Est, France
- Eiji Okamoto: LCIS, University of Tsukuba, Japan
Outline of the talk

1. Context and motivation
2. Computing the Tate Pairing
3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
6. Wish list on hardware implementation of pairings (Some concrete open problems)
**Agenda**

1. **Context and motivation**
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. **Computing the Tate Pairing**
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. **Case of Study #1: A compact implementation of the $\eta_T$ pairing**
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. **Case of Study #2: Estibals’ composite $\eta_T$ pairing**
   - Attacks

5. **Case of Study #3: A fast implementation of the $\eta_T$ pairing**
   - Implementation Results in Hardware

6. **Wish list on hardware implementation of pairings (Some concrete open problems)**
Elliptic curves

- $E$ defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$
Elliptic curves

- $E$ defined by a Weierstraß equation of the form
  \[ y^2 = x^3 + Ax + B \]

- $E(K)$ set of rational points over a field $K$
Elliptic curves

- $E$ defined by a Weierstraß equation of the form
  \[ y^2 = x^3 + Ax + B \]

- $E(K)$ set of rational points over a field $K$

- Additive group law over $E(K)$

Many applications in cryptography since 1985
- EC-based Diffie-Hellman key exchange
- EC-based Digital Signature Algorithm

Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)

But there's more: bilinear pairings

Francisco Rodríguez-Henríquez

Hardware Implementation of Pairings (4 / 91)
Elliptic curves

- $E$ defined by a Weierstraß equation of the form
  \[ y^2 = x^3 + Ax + B \]

- $E(K)$ set of rational points over a field $K$

- Additive group law over $E(K)$

- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm
  - ...

- Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)
- But there's more: bilinear pairings
Elliptic curves

- $E$ defined by a Weierstraß equation of the form
  \[ y^2 = x^3 + Ax + B \]

- $E(K)$ set of rational points over a field $K$
- Additive group law over $E(K)$
- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm
  - ...
- Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)
Elliptic curves

- $E$ defined by a Weierstraß equation of the form
  \[ y^2 = x^3 + Ax + B \]

- $E(K)$ set of rational points over a field $K$
- Additive group law over $E(K)$
- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm
  - ...

- Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)
- But there’s more: bilinear pairings
Group cryptography

- Let \((G_1, +)\) be an additively-written cyclic group of prime order
  \[#G_1 = \ell\]
Group cryptography

- Let $(\mathbb{G}_1, +)$ be an additively-written [cyclic group](https://en.wikipedia.org/wiki/Cyclic_group) of [prime order](https://en.wikipedia.org/wiki/Prime_number) $\#\mathbb{G}_1 = \ell$
- $P$, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
Group cryptography

- Let $(G_1, +)$ be an additively-written cyclic group of prime order $\#G_1 = \ell$
- $P$, a generator of the group: $G_1 = \langle P \rangle$
- Scalar multiplication: for any integer $k$, we have $kP = P + P + \cdots + P$ (\(k\) times)
Group cryptography

- Let $(\mathbb{G}_1, +)$ be an additively-written cyclic group of prime order $\# \mathbb{G}_1 = \ell$
- $P$, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer $k$, we have $kP = P + P + \cdots + P$ ($k$ times)

Francisco Rodríguez-Henríquez
Group cryptography

- Let \((G_1, +)\) be an additively-written cyclic group of prime order \(\#G_1 = \ell\)
- \(P\), a generator of the group: \(G_1 = \langle P \rangle\)
- Scalar multiplication: for any integer \(k\), we have
  \[kP = P + P + \cdots + P\]
  \(k\) times

Discrete logarithm: given \(Q \in G_1\), compute \(k\) such that
\[Q = kP = P + P + \cdots + P\]
Group cryptography

- Let $(G_1, +)$ be an additively-written cyclic group of prime order $\#G_1 = \ell$
- $P$, a generator of the group: $G_1 = \langle P \rangle$
- Scalar multiplication: for any integer $k$, we have $kP = P + P + \cdots + P$
  $\underbrace{P + P + \cdots + P}_{k \text{ times}}$

- Discrete logarithm: given $Q \in G_1$, compute $k$ such that $Q = kP$
Group cryptography

- Let \((G_1, +)\) be an additively-written cyclic group of prime order \(\#G_1 = \ell\)
- \(P\), a generator of the group: \(G_1 = \langle P \rangle\)
- Scalar multiplication: for any integer \(k\), we have \(kP = P + P + \cdots + P\) \(k\) times

Discrete logarithm: given \(Q \in G_1\), compute \(k\) such that \(Q = kP\)
Group cryptography

- Let \((G_1, +)\) be an additively-written cyclic group of prime order 
  \(\#G_1 = \ell\)
- \(P\), a generator of the group: \(G_1 = \langle P \rangle\)
- Scalar multiplication: for any integer \(k\), we have 
  \[kP = P + P + \cdots + P\quad \text{\(k\) times}\]
- Discrete logarithm: given \(Q \in G_1\), compute \(k\) such that \(Q = kP\)
Group cryptography

- Let $(\mathbb{G}_1, +)$ be an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- $P$, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer $k$, we have $kP = P + P + \cdots + P$ (k times)

\[ Q = kP \]

- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute $k$ such that $Q = kP$
Group cryptography

- Let \((G_1, +)\) be an additively-written cyclic group of prime order \(\#G_1 = \ell\)
- \(P\), a generator of the group: \(G_1 = \langle P \rangle\)
- Scalar multiplication: for any integer \(k\), we have 
  \[kP = P + P + \cdots + P\] 
  \(k\) times
- Discrete logarithm: given \(Q \in G_1\), compute \(k\) such that \(Q = kP\)
- We assume that the discrete logarithm problem (DLP) in \(G_1\) is hard
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(#G_1 = #G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[ \hat{e} : G_1 \times G_2 \to G_\tau \]

that satisfies the following conditions:
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map
  \[\hat{e} : G_1 \times G_2 \rightarrow G_\tau\]
  that satisfies the following conditions:
  - non-degeneracy: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[
\hat{e} : G_1 \times G_2 \rightarrow G_\tau
\]

that satisfies the following conditions:

- **non-degeneracy**: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- **bilinearity**: 
  \[
  \hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)
  \]

Immediate property: for any two integers \(k_1, k_2\)

\[
\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}
\]

When \(G_1 = G_2\) we say that the pairing is symmetric, otherwise if \(G_1 \neq G_2\), the pairing is asymmetric.
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(\#G_1 = \#G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(\#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[
\hat{e} : G_1 \times G_2 \rightarrow G_\tau
\]

that satisfies the following conditions:

- **non-degeneracy**: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- **bilinearity**:
  \[
  \hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)
  \]
- **computability**: \(\hat{e}\) can be efficiently computed
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(#G_1 = #G_2 = \ell\)

- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(#G_\tau = \ell\)

- A non-degenerate bilinear pairing is a map

\[ \hat{e} : G_1 \times G_2 \rightarrow G_\tau \]

that satisfies the following conditions:

- **non-degeneracy**: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))

- **bilinearity**:

\[
\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) = \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)
\]

- **computability**: \(\hat{e}\) can be efficiently computed

- **Immediate property**: for any two integers \(k_1\) and \(k_2\)

\[
\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}
\]
Bilinear pairings

- Let \((G_1, +), (G_2, +)\) be two additively-written cyclic groups of prime order \(#G_1 = #G_2 = \ell\)
- \((G_\tau, \times)\), a multiplicatively-written cyclic group of order \(#G_\tau = \ell\)
- A non-degenerate bilinear pairing is a map

\[
\hat{e} : G_1 \times G_2 \to G_\tau
\]

that satisfies the following conditions:

- **non-degeneracy**: \(\hat{e}(P, P) \neq 1_{G_\tau}\) (equivalently \(\hat{e}(P, P)\) generates \(G_\tau\))
- **bilinearity**:
  \[
  \hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)
  \]
- **computability**: \(\hat{e}\) can be efficiently computed

- **Immediate property**: for any two integers \(k_1\) and \(k_2\)

\[
\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1k_2}
\]

- When \(G_1 = G_2\) we say that the pairing is symmetric, otherwise if \(G_1 \neq G_2\), the pairing is asymmetric.
Pairings in cryptography

- At first, used to attack supersingular elliptic curves

\[
\text{DLP}_{\mathbb{G}_1} <_P \Rightarrow \text{DLP}_{\mathbb{G}_\tau}
\]

\[
kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k
\]

- for cryptographic applications, we will also require the DLP in \( \mathbb{G}_\tau \) to be hard
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
  \[
  \text{DLP}_{\mathbb{G}_1} \overset{<_P}{\longrightarrow} \text{DLP}_{\mathbb{G}_\tau}
  \]
  \[
  kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k
  \]
  - for cryptographic applications, we will also require the DLP in \( \mathbb{G}_\tau \) to be hard

- One-round three-party key agreement (Joux, 2000)
Pairings in cryptography

- At first, used to attack **supersingular elliptic curves**
    
    \[
    \text{DLP}_{G_1} <_P \text{DLP}_{G_T} \\
    kP \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k 
    \]

- for **cryptographic applications**, we will also require the **DLP** in \(G_T\) to be hard

- **One-round three-party** key agreement (**Joux**, 2000)

- **Identity-based encryption**
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
    \[
    \begin{align*}
    \text{DLP}_{G_1} & <_P \text{ DLP}_{\hat{G}_\tau} \\
    kP & \rightarrow \hat{e}(kP, P) = \hat{e}(P, P)^k
    \end{align*}
    \]
  - for cryptographic applications, we will also require the DLP in $\hat{G}_\tau$ to be hard
- One-round three-party key agreement (Joux, 2000)
- Identity-based encryption
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001
- Short digital signatures
  - Boneh–Lynn–Shacham, 2001
  - Zang–Safavi-Naini–Susilo, 2004
- ...
1. Context and motivation
   - Bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
But.... Why should one bother implementing pairings in Hardware?
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz

- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]

- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]


- **2003**: Simplifications on the Miller loop and final exponentiation computation [Duursma and Lee Asiacrypt'03]

- **2006**: eta pairing [Hess-Smart-Vercauteren IEEE TIT]

- **2007**: $\eta_T$ pairing [Barreto-Galbraith-\'{E}igeartaigh-Scott DCC'07]

- **2010**: optimal pairings [Vercauteren IEEE TIT]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
- **2003**: Simplifications on the Miller loop and final exponentiation computation [Duursma and Lee Asiacrypt’03]
- **2006**: η pairing [Hess-Smart-Vercauteren IEEE TIT]
- **2007**: η pairing [Barreto-Ballbraith-Eigeartaigh-Scott DCC’07]
- **2010**: Optimal pairings [Vercauteren IEEE TIT]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
- **2003**: Simplifications on the Miller loop and final exponentiation computation [Duursma and Lee Asiacrypt’03]
- **2006**: eta pairing [Hess-Smart-Vercauteren IEEE TIT]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
- **2003**: Simplifications on the Miller loop and final exponentiation computation [Duursma and Lee Asiacrypt’03]
- **2006**: eta pairing [Hess-Smart-Vercauteren IEEE TIT]
- **2007**: $\eta_T$ pairing [Barreto-Galbraith-hÉigeartaigh-Scott DCC’07]
Some important breakthroughs on pairing computations

- **1985**: Elliptic curve cryptography is independently invented by V. Miller and N. Koblitz
- **1986**: V. Miller devises an iterative algorithm for computing pairings [unpublished report until it appeared at JoC 2004]
- **1993**: MOV attack [Menezes-Okamoto-Vanstone IEEE TIT]
- **2003**: Simplifications on the Miller loop and final exponentiation computation [Duursma and Lee Asiacrypt’03]
- **2006**: eta pairing [Hess-Smart-Vercauteren IEEE TIT]
- **2007**: $\eta_T$ pairing [Barreto-Galbraith-hÉigeartaigh-Scott DCC’07]
- **2010**: optimal pairings [Vercauteren IEEE TIT]
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- A general method for constructing curves of prescribed embedding degree: [Cocks and Pinch- unpublished 2001]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- A general method for constructing curves of prescribed embedding degree: [Cocks and Pinch- unpublished 2001]
- **BLS curves**: [Barreto-Lynn-Scott 2002]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- A general method for constructing curves of prescribed embedding degree: [Cocks and Pinch- unpublished 2001]
- **BLS curves**: [Barreto-Lynn-Scott 2002]
- **BN Curves**: [Barreto and Naehrig SAC’05]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- A general method for constructing curves of prescribed embedding degree: [Cocks and Pinch- unpublished 2001]
- **BLS curves**: [Barreto-Lynn-Scott 2002]
- **BN Curves**: [Barreto and Naehrig SAC’05]
- **BW Curves**: [Brezing and Weng DCC’05]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- A general method for constructing curves of prescribed embedding degree: [Cocks and Pinch- unpublished 2001]
- **BLS curves**: [Barreto-Lynn-Scott 2002]
- **BN Curves**: [Barreto and Naehrig SAC’05]
- **BW Curves**: [Brezing and Weng DCC’05]
- **Freeman Curves**: [Freeman ANTS’06]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
A brief recount of pairing-friendly ordinary curves

- **MNT curves**: [Miyaji-Nakabayashi-Takano IEICE 2001]
- **A general method for constructing curves of prescribed embedding degree**: [Cocks and Pinch- unpublished 2001]
- **BLS curves**: [Barreto-Lynn-Scott 2002]
- **BN Curves**: [Barreto and Naehrig SAC’05]
- **BW Curves**: [Brezing and Weng DCC’05]
- **Freeman Curves**: [Freeman ANTS’06]
- **KSS curves**: [Kachisa-Schaefer-Scott Pairing’08]

Unfortunately, we don’t know how to find binary pairing friendly ordinary curves
But.... Why should one bother implement pairings in Hardware?

- computation not very well suited for *general purpose* processor
But.... Why should one bother implement pairings in Hardware?

- computation not very well suited for **general purpose** processor
- There exist specific targets, one of the most prominent ones being smart cards
But…. Why should one bother implement pairings in Hardware?

- computation not very well suited for general purpose processor
- There exist specific targets, one of the most prominent ones being smart cards
- Hardware may be the fastest/most efficient way to implement pairings.
But.... Why should one bother implement pairings in Hardware?

- computation not very well suited for general purpose processor
- There exist specific targets, one of the most prominent ones being smart cards
- Hardware may be the fastest/most efficient way to implement pairings.
- However if a pairing hardware accelerator is going to be attractive at all, a significant performance improvement should be observed with respect to software implementations.
A brief recount and notes on the first pairing hardware accelerators

- First designs appeared circa 2005: [Grabher & Page CHES’05], [Kerins et al. CHES’05]
A brief recount and notes on the first pairing hardware accelerators

- First designs appeared circa 2005: [Grabher & Page CHES’05], [Kerins et al. CHES’05]

- Historically, the first designs targeted low-security Tate Pairing over Supersingular elliptic curves defined on ternary fields [a typical selection was \( \mathbb{F}_{3^{97}}^\times \)]
A brief recount and notes on the first pairing hardware accelerators

- First designs appeared circa 2005: [Grabher & Page CHES’05], [Kerins et al. CHES’05]

- Historically, the first designs targeted low-security Tate Pairing over Supersingular elliptic curves defined on ternary fields [a typical selection was $\mathbb{F}_3^{97}$]

- First designs did not take advantage of the inverse Frobenius map [extracting cube roots was considered expensive]
A brief recount and notes on the first pairing hardware accelerators

- First designs appeared circa 2005: [Grabher & Page CHES’05], [Kerins et al. CHES’05]

- Historically, the first designs targeted low-security Tate Pairing over Supersingular elliptic curves defined on ternary fields [a typical selection was $\mathbb{F}_3^{97}$]

- First designs did not take advantage of the inverse Frobenius map [extracting cube roots was considered expensive]

- Curiously, the first hardware implementations of the Tate pairing defined on binary field extensions appeared one year later: [Keller et al. ARC’06], [Shu et al. FPT’06].
A brief recount and notes on the first pairing hardware accelerators

- First designs appeared circa 2005: [Grabher & Page CHES’05], [Kerins et al. CHES’05]

- Historically, the first designs targeted low-security Tate Pairing over Supersingular elliptic curves defined on ternary fields [a typical selection was $\mathbb{F}_3^{97}$]

- First designs did not take advantage of the inverse Frobenius map [extracting cube roots was considered expensive]

- Curiously, the first hardware implementations of the Tate pairing defined on binary field extensions appeared one year later: [Keller et al. ARC’06], [Shu et al. FPT’06].

- The first hardware designs for the Tate pairing over ordinary curves defined on large prime fields came out until 2009: [Fan et al. CHES’09], [Kammler et al. CHES’09]
Agenda

1 Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2 Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3 Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4 Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5 Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6 Wish list on hardware implementation of pairings (Some concrete open problems)
General Xilinx Virtex 5 Slice architecture
# Xilinx FPGA Families

<table>
<thead>
<tr>
<th></th>
<th>Virtex-5</th>
<th>Virtex-4</th>
<th>Virtex II Pro</th>
<th>Spartan 3 &amp; 3E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logic Cells</strong></td>
<td>up to 330K</td>
<td>12K-200K</td>
<td>3K-99K</td>
<td>1.7K-74K</td>
</tr>
<tr>
<td><strong>BRAM</strong></td>
<td>576</td>
<td>36-512</td>
<td>12-444</td>
<td>4-104</td>
</tr>
<tr>
<td>(18Kbits each)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Multipliers</strong></td>
<td>32 – 192(^1)</td>
<td>32-512</td>
<td>12-444</td>
<td>4-104</td>
</tr>
<tr>
<td><strong>DCM</strong></td>
<td>up to 18</td>
<td>4-20</td>
<td>4-12</td>
<td>2-18</td>
</tr>
<tr>
<td><strong>IOBs</strong></td>
<td>up to 1200</td>
<td>240-960</td>
<td>204-1164</td>
<td>63-633</td>
</tr>
<tr>
<td><strong>DSP Slices</strong></td>
<td>32-192</td>
<td>32-192</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td><strong>PowerPC Blocks</strong></td>
<td>N/A</td>
<td>0-2</td>
<td>0-2</td>
<td>–</td>
</tr>
<tr>
<td><strong>Max. freq.</strong></td>
<td>550MHz</td>
<td>500MHz</td>
<td>547 MHz</td>
<td>up to 300MHz</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>1.0V, 65(\eta)m, copper CMOS</td>
<td>1.2V, 90(\eta)m, triple-oxide process</td>
<td>1.5V, 130(\eta)m, 9-layer CMOS</td>
<td>1.2V, 90(\eta)m, triple-oxide process</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>$400 USD</td>
<td>From $300</td>
<td>From $139</td>
<td>From $2 up to $85</td>
</tr>
</tbody>
</table>

\(^1\)25 \times 18 embedded multipliers
Design Methodology for FPGA designs

- Design Entry
- FPGA Synthesis
- FPGA Place & Route
- FPGA Programming

- Functional Simulation
- Circuit Analysis
Measures of performance in reconfigurable Hardware devices

![Dilbert comic strip](image-url)

- We added a new performance test, but learned that the test itself is flawed.
- Now our product fails our own tests and our customers are asking to see the test results.
- Do I have permission to fake the test data? I didn’t even know data can be real.
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:

  \[
  \frac{\text{# of clock cycles}}{\text{clock cycle frequency}}
  \]
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:
  \[
  \frac{\text{\# of clock cycles}}{\text{clock cycle frequency}}
  \]

- **Throughput** defined as:
  \[
  \frac{\text{\# of bits processed} \cdot \text{clock cycle frequency}}{\text{\# of clock cycles}}
  \]
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:
  
  \[
  \frac{\text{# of clock cycles}}{\text{clock cycle frequency}}
  \]

- **Throughput** defined as:
  
  \[
  \frac{\text{# of bits processed} \cdot \text{clock cycle frequency}}{\text{# of clock cycles}}
  \]

- **Latency**: 
  
  \[
  \text{# of clock cycles required for producing the first computation}
  \]
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:
  \[
  \frac{\text{\# of clock cycles}}{\text{clock cycle frequency}}
  \]

- **Throughput** defined as:
  \[
  \frac{\text{\# of bits processed} \cdot \text{clock cycle frequency}}{\text{\# of clock cycles}}
  \]

- **Latency**: \# of clock cycles required for producing the first computation

- **Amount of hardware resources utilized by the design**. Including slices, dedicated memories, DSP slices, etc.
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:

  \[
  \frac{\text{# of clock cycles}}{\text{clock cycle frequency}}
  \]

- **Throughput** defined as:

  \[
  \frac{\text{# of bits processed} \cdot \text{clock cycle frequency}}{\text{# of clock cycles}}
  \]

- **Latency**: # of clock cycles required for producing the first computation

- **Amount of hardware resources utilized by the design**. Including slices, dedicated memories, DSP slices, etc.

- **Time-Area product**
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:

  \[ \frac{\text{# of clock cycles}}{\text{clock cycle frequency}} \]

- **Throughput** defined as:

  \[ \frac{\text{# of bits processed} \cdot \text{clock cycle frequency}}{\text{# of clock cycles}} \]

- **Latency**: # of clock cycles required for producing the first computation

- **Amount of hardware resources utilized by the design**: Including slices, dedicated memories, DSP slices, etc.

- **Time-Area product**

- **Power consumption, energy consumption**, ...
Measures of performance in reconfigurable Hardware devices

- **Computational time** defined as:
  
  \[
  \frac{\text{# of clock cycles}}{\text{clock cycle frequency}}
  \]

- **Throughput** defined as:
  
  \[
  \frac{\text{# of bits processed} \cdot \text{clock cycle frequency}}{\text{# of clock cycles}}
  \]

- **Latency**: # of clock cycles required for producing the first computation

- **Amount of hardware resources utilized by the design.** Including slices, dedicated memories, DSP slices, etc.

- **Time-Area product**

- **Power consumption, energy consumption, ...**

- **In the case of cryptographic designs:** Side-channel resistance
Parallel techniques in hardware

(a) One round

(b) $n$ round combinational logic
Parallel techniques in hardware
Parallel techniques in hardware: AES example
Agenda

1. Context and motivation
   - Bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
The Tate Pairing over Supersingular elliptic curves

We first define

- $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
- $E$, an elliptic curve defined over $\mathbb{F}_q$
- $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$

- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:

$$G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = O \}$$
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$

- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:
  $$G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = \mathcal{O} \}$$

- $G_{\tau} = \mu_{\ell}$, the group of $\ell$-th roots of unity in $\mathbb{F}_{q^k}^\times$:
  $$G_{\tau} = \{ U \in \mathbb{F}_{q^k}^\times \mid U^\ell = 1 \}$$
The Tate Pairing over Supersingular elliptic curves

- We first define
  - $\mathbb{F}_q$, a finite field, with $q = 2^m$ or $3^m$
  - $E$, an elliptic curve defined over $\mathbb{F}_q$
  - $\ell$, a large prime factor of $\#E(\mathbb{F}_q)$
- $G_1 = E(\mathbb{F}_q)[\ell]$, the $\mathbb{F}_q$-rational $\ell$-torsion of $E$:
  $$G_1 = \{ P \in E(\mathbb{F}_q) \mid \ell P = \mathcal{O} \}$$
- $G_\tau = \mu_\ell$, the group of $\ell$-th roots of unity in $\mathbb{F}_{q^k}^\times$:
  $$G_\tau = \{ U \in \mathbb{F}_{q^k}^\times \mid U^\ell = 1 \}$$
- $k$ is the embedding degree, the smallest integer such that $\mu_\ell \subseteq \mathbb{F}_{q^k}^\times$
  - usually large for ordinary elliptic curves
  - bounded in the case of supersingular elliptic curves
    (4 in characteristic 2; 6 in characteristic 3; and 2 in characteristic $> 3$)
Security considerations

\( \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{q^k}^{\times} \)
Security considerations

\[ \hat{e} : E(F_q)[\ell] \times E(F_q)[\ell] \to \mu_\ell \subseteq \mathbb{F}_{q^k}^\times \]

- Discrete logarithm in \( \mathbb{G}_1 = E(F_q)[\ell] \) (Pollard’s \( \rho \)):

\[ \sqrt{\ell} \approx \sqrt{q} \]
Security considerations

\[ \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{q^k} \]

- Discrete logarithm in \( G_1 = E(\mathbb{F}_q)[\ell] \) (Pollard's \( \rho \)):

\[ \sqrt{\ell} \approx \sqrt{q} = \exp \left( \frac{1}{2} \cdot (\ln q) \right) \]
Security considerations

\[ \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \to \mu_\ell \subseteq \mathbb{F}_{q_k}^\times \]

- Discrete logarithm in \( \mathbb{G}_1 = E(\mathbb{F}_q)[\ell] \) (Pollard’s \( \rho \)):

\[
\sqrt{\ell} \approx \sqrt{q} = \exp \left( \frac{1}{2} \cdot (\ln q) \right)
\]

- Discrete logarithm in \( \mathbb{G}_2 = \mu_\ell \subseteq \mathbb{F}_{q_k}^\times \) (FFS or NFS):

\[
\exp \left( c \cdot (\ln q^k)^{\frac{1}{3}} \cdot (\ln \ln q^k)^{\frac{2}{3}} \right)
\]
Security considerations

\[ \hat{e} : E(\mathbb{F}_q)[\ell] \times E(\mathbb{F}_q)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_q^\times \]

- **Discrete logarithm in** \( G_1 = E(\mathbb{F}_q)[\ell] \) (Pollard’s ρ):

  \[ \sqrt{\ell} \approx \sqrt{q} = \exp \left( \frac{1}{2} \cdot \ln q \right) \]

- **Discrete logarithm in** \( G_2 = \mu_\ell \subseteq \mathbb{F}_q^\times \) (FFS or NFS):

  \[ \exp \left( c \cdot \left( \ln q^k \right)^{\frac{1}{3}} \cdot \left( \ln \ln q^k \right)^{\frac{2}{3}} \right) \]

- The discrete logarithm problem is usually easier in \( G_2 \) than in \( G_1 \)
  - current security: \( \sim 2^{80} \), equivalent to 80-bit symmetric encryption or RSA-1024
  - recommended security: \( \sim 2^{128} \) (AES-128, RSA-3072)
Security considerations for Symmetric Pairings

\[ \hat{e} : E(F_p^m)[\ell] \times E(F_p^m)[\ell] \to \mu_{\ell} \subseteq F_{p^{km}}^\times \]

- The discrete logarithm problem should be hard in both \( G_1 \) and \( G_T \)
Security considerations for Symmetric Pairings

\[ \hat{e} : E(\mathbb{F}_p^m)[\ell] \times E(\mathbb{F}_p^m)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^km}^\times \]

- The discrete logarithm problem should be hard in both \( G_1 \) and \( G_T \)

<table>
<thead>
<tr>
<th>Base field ( (\mathbb{F}_p^m) )</th>
<th>( \mathbb{F}_{2m} )</th>
<th>( \mathbb{F}_{3m} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lower security</strong> ((\sim 2^{64}))</td>
<td>( m = 239 )</td>
<td>( m = 97 )</td>
</tr>
<tr>
<td><strong>Medium security</strong> ((\sim 2^{80}))</td>
<td>( m = 373 )</td>
<td>( m = 163 )</td>
</tr>
<tr>
<td><strong>Higher security</strong> ((\sim 2^{128}))</td>
<td>( m = 1103 )</td>
<td>( m = 503 )</td>
</tr>
</tbody>
</table>

- \( \mathbb{F}_{2m} \): simpler finite field arithmetic
- \( \mathbb{F}_{3m} \): smaller field extension
Agenda
1. Context and motivation
   - Bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in hardware?
   - A quick overview of reconfigurable hardware devices
2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves
3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained
4. Case of Study #2: Estibals' composite $\eta_T$ pairing
   - Attacks
5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware
6. Wish list on hardware implementation of pairings (Some concrete open problems)
Barreto–Naehrig Curves

Defined by the equation $E : y^2 = x^3 + b$, where $b \neq 0$. Their embedding degree $k$ is equal to 12. The characteristic $p$ of the prime field, the group order $r$, and the trace of Frobenius $t_r$ of the curve are parametrized as follows:
Barreto–Naehrig Curves

Defined by the equation $E : y^2 = x^3 + b$, where $b \neq 0$. Their embedding degree $k$ is equal to 12. The characteristic $p$ of the prime field, the group order $r$, and the trace of Frobenius $t_r$ of the curve are parametrized as follows:

$$p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1,$$
$$r(t) = 36t^4 + 36t^3 + 18t^2 + 6t + 1,$$
$$t_r(t) = 6t^2 + 1,$$

where $t \in \mathbb{Z}$ is an arbitrary integer such that $p = p(t)$ and $r = r(t)$ are both prime numbers.
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \rightarrow E$ given by
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \rightarrow E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r],$

In practice, pairing computations can be restricted to points $P$ and $Q'$ that belong to $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_p^2)$, respectively, where, $E' / \mathbb{F}_p^2 : y^2 = x^3 + b/\xi$. 

Francisco Rodríguez-Henríquez
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r]$,
- $G_2 \subseteq E(\mathbb{F}_{p^{12}})[r]$,
Barreto–Naehrig Curves

Let \( E[r] \) denote the \( r \)-torsion subgroup of \( E \) and \( \pi_p \) be the Frobenius endomorphism \( \pi_p : E \to E \) given by\( \pi_p(x, y) = (x^p, y^p) \). We define,

- \( G_1 = E(\mathbb{F}_p)[r] \),
- \( G_2 \subseteq E(\mathbb{F}_{p^{12}})[r] \),
- \( G_\tau = \mu_r \subset \mathbb{F}_{p^{12}}^* \) (i.e. the group of \( r \)-th roots of unity).
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \rightarrow E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r],$
- $G_2 \subseteq E(\mathbb{F}_{p^{12}})[r],$
- $G_\tau = \mu_r \subseteq \mathbb{F}_{p^{12}}^* (i.e. the group of $r$-th roots of unity).
- The optimal ate pairing on the BN curve $E$ is given as,
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r]$,
- $G_2 \subseteq E(\mathbb{F}_{p^{12}})[r]$,
- $G_\tau = \mu_r \subset \mathbb{F}_{p^{12}}^*$ (i.e. the group of $r$-th roots of unity).

The optimal ate pairing on the BN curve $E$ is given as,

$$a_{opt} : G_2 \times G_1 \to G_3$$

$$(Q, P) \longmapsto (f_{6t+2}Q(P) \cdot l_{6t+2}Q, \pi_p(Q)(P) \cdot l_{6t+2}Q, \pi_p(Q), \pi_p^2(Q)(P))^{p^{12} - 1 \over r},$$

In practice, pairing computations can be restricted to points $P$ and $Q'$ that belong to $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^2})$, respectively, where, $E'/\mathbb{F}_{p^2}: y^2 = x^3 + b/\xi$. Francisco Rodríguez-Henríquez Hardware Implementation of Pairings (29 / 91)
Barreto–Naehrig Curves

Let $E[r]$ denote the $r$-torsion subgroup of $E$ and $\pi_p$ be the Frobenius endomorphism $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define,

- $G_1 = E(\mathbb{F}_p)[r],$
- $G_2 \subseteq E(\mathbb{F}_{p^{12}})[r],$
- $G_\tau = \mu_r \subset \mathbb{F}_{p^{12}}^*$ (i.e. the group of $r$-th roots of unity).
- The optimal ate pairing on the BN curve $E$ is given as,

$$a_{\text{opt}} : G_2 \times G_1 \longrightarrow G_3$$

$$(Q, P) \longmapsto (f_{6t+2}Q(P) \cdot l_{[6t+2]}Q, \pi_p(Q)(P) \cdot l_{[6t+2]}Q + \pi_p(Q)(P))^{p^{12} - 1}_r,$$

- In practice, pairing computations can be restricted to points $P$ and $Q'$ that belong to $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^2})$, respectively, where, $E'/\mathbb{F}_{p^2} : y^2 = x^3 + b/\xi$. 

## Computational costs of the tower extension field arithmetic

<table>
<thead>
<tr>
<th>Field</th>
<th>Add/Sub</th>
<th>Mult</th>
<th>Squaring</th>
<th>Inversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{p^2}$</td>
<td>$\tilde{a} = 2a$</td>
<td>$\tilde{m} = 3m + 3a + m_\beta$</td>
<td>$\tilde{s} = 2m + 3a + m_\beta$</td>
<td>$\tilde{i} = 4m + m_\beta + 2a + i$</td>
</tr>
<tr>
<td>$\mathbb{F}_{p^6}$</td>
<td>$3\tilde{a}$</td>
<td>$6\tilde{m} + 2m_\xi + 15\tilde{a}$</td>
<td>$2\tilde{m} + 3\tilde{s} + 2m_\xi + 8\tilde{a}$</td>
<td>$9\tilde{m} + 3\tilde{s} + 4m_\xi + 4\tilde{a} + \tilde{i}$</td>
</tr>
<tr>
<td>$\mathbb{F}_{p^{12}}$</td>
<td>$6\tilde{a}$</td>
<td>$18\tilde{m} + 6m_\xi + 60\tilde{a}$</td>
<td>$12\tilde{m} + 4m_\xi + 45\tilde{a}$</td>
<td>$25\tilde{m} + 9\tilde{s} + 12m_\xi + 61\tilde{a} + \tilde{i}$</td>
</tr>
<tr>
<td>$\mathbb{G}<em>{\Phi_6}(\mathbb{F}</em>{p^2})$</td>
<td>$6\tilde{a}$</td>
<td>$18\tilde{m} + 6m_\xi + 60\tilde{a}$</td>
<td>$9\tilde{s} + 4m_\xi + 30\tilde{a}$</td>
<td>Conjugate</td>
</tr>
</tbody>
</table>

---

Francisco Rodríguez-Henríquez

Hardware Implementation of Pairings (30 / 91)
Optimal ate pairing algorithm

**Input:** $P \in \mathbb{G}_1$ y $Q \in \mathbb{G}_2$.

**Output:** $a_{\text{opt}}(Q, P)$.

1. Write $s = 6t + 2$ as $s = \sum_{i=0}^{L-1} s_i2^i$, where $s_i \in \{-1, 0, 1\}$;
2. $T \leftarrow Q$, $f \leftarrow 1$;
3. **for** $i = L - 2$ to 0 **do**
4. \hspace{1em} $f \leftarrow f^2 \cdot l_{T,T}(P)$; $T \leftarrow 2T$;
5. \hspace{1em} **if** $s_i = -1$ **then**
6. \hspace{2em} $f \leftarrow f \cdot l_{T,-Q}(P)$; $T \leftarrow T - Q$;
7. \hspace{1em} **else if** $s_i = 1$ **then**
8. \hspace{2em} $f \leftarrow f \cdot l_{T,Q}(P)$; $T \leftarrow T + Q$;
9. \hspace{1em} **end if**
10. **end for**
11. $Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow \pi_{p^2}(Q)$;
12. $f \leftarrow f \cdot l_{T,Q_1}(P)$; $T \leftarrow T + Q_1$;
13. $f \leftarrow f \cdot l_{T,-Q_2}(P)$; $T \leftarrow T - Q_2$;
14. $f \leftarrow f(p^{12}-1)/r$;
15. **return** $f$;
Since $p \mod 12 \equiv 1$ we can build the towering up to the twelfth extension by adjoining irreducible binomial only.
Since $p \mod 12 \equiv 1$ we can build the towering up to the twelfth extension by adjoining irreducible binomial only.
Since $p \mod 12 \equiv 1$ we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

$$\beta = 1$$
Since $p \mod 12 \equiv 1$ we can build the towering up to the twelfth extension by adjoining irreducible binomial only.

$$\xi = u$$

$$\beta = 1$$
Since $p \mod 12 \equiv 1$ we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

\[ f \in \mathbb{F}_{p^{12}} \]
\[ f = g + hw = g_0 + h_0 w + g_1 w^2 + h_1 w^3 + \cdots 
\]

\[ \gamma = v 
\]
\[ \xi = u 
\]
\[ \beta = 1 
\]
Since $p \mod 12 \equiv 1$ we can build the towering up to the twelfth extension by adjoining irreducible binomial only.

$f = g + hw \in \mathbb{F}_{p^{12}}$, with $g, h \in \mathbb{F}_{p^6}$.

but also

$g = g_0 + g_1 v + g_2 v^2$,  
$h = h_0 + h_1 v + h_2 v^2$,  
where $g_i, h_i \in \mathbb{F}_{p^2}$,  
for $i = 1, 2, 3$.  

\[
\begin{align*}
\mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2 - \gamma) \\
\mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3 - \xi) \\
\mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2 - \beta)
\end{align*}
\]
Since $p \mod 12 \equiv 1$ we can build the tower up to the twelfth extension by adjoining irreducible binomial only.

hence, we can write $f \in \mathbb{F}_{p^{12}}$ as

\[
\begin{align*}
f &= g + hw \\
&= g + hw \\
&= g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5.
\end{align*}
\]
Supersingular elliptic curves \textit{Vs.} Barreto–Naehrig curves

- **Definition:**
  \[ E/F_3 : \quad y^2 = x^3 - x + b, \; b \neq 0 \]

- **Supersingular curve**
  \[ \implies \text{Simpler curve arithmetic} \]

- **Ordinary curve**
  \[ \implies \text{(efficient tripling formulae)} \]

- **Definition:**
  \[ E/F_p : \quad y^2 = x^3 + b, \; b \neq 0, \]
  \[ p = 36\alpha^4 - 36\alpha^3 + 24\alpha^2 - 6\alpha + 1 \]
Supersingular elliptic curves  Vs. Barreto–Naehrig curves

Definition:

\[ E / \mathbb{F}_3 : \ y^2 = x^3 - x + b, \ b \neq 0 \]

Definition:

\[ E / \mathbb{F}_p : \ y^2 = x^3 + b, \ b \neq 0, \]
\[ p = 36\alpha^4 - 36\alpha^3 + 24\alpha^2 - 6\alpha + 1 \]

Supersingular curve ⇒ Simpler curve arithmetic

Ordinary curve (efficient tripling formulae)

Distortion map, modified pairing:

\[ \delta : E(\mathbb{F}_q)[\ell] \rightarrow E(\mathbb{F}_{q^k})[\ell] \]
\[ \hat{e}(P, Q) = e(P, \delta(Q)) \]

⇒ Symmetric pairing (BN cannot be used with all protocols)
### Supersingular elliptic curves Vs. Barreto–Naehrig curves

<table>
<thead>
<tr>
<th>Supersingular curve</th>
<th>Ordinary curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E / \mathbb{F}_3 : y^2 = x^3 - x + b, b \neq 0$</td>
<td>$E / \mathbb{F}_p : y^2 = x^3 + b, b \neq 0, \quad p = 36\alpha^4 - 36\alpha^3 + 24\alpha^2 - 6\alpha + 1$</td>
</tr>
</tbody>
</table>

- **Definition:**

  - Supersingular curve $\Rightarrow$ Simpler curve arithmetic (efficient tripling formulae)
  - Distortion map, modified pairing:
    
    $\delta : E(\mathbb{F}_q)[\ell] \rightarrow E(\mathbb{F}_q^{k})[\ell]$
    
    $\hat{e}(P, Q) = e(P, \delta(Q))$
    
    $\Rightarrow$ Symmetric pairing (BN cannot be used with all protocols)

- **Definition:**

  - Ordinary curve $\Rightarrow$ No distortion map
  - Modular arithmetic $\Rightarrow$ No carry, better suited to hardware implementation

- **Small characteristic field arithmetic**
  - $\Rightarrow$ No carry, better suited to hardware implementation

- **Modular arithmetic**
  - $\Rightarrow$ No carry, better suited to hardware implementation

- **Small characteristic field arithmetic** $\Rightarrow$ No carry, better suited to hardware implementation

- **Distortion map, modified pairing:**
  - $\delta : E(\mathbb{F}_q)[\ell] \rightarrow E(\mathbb{F}_q^{k})[\ell]$
  - $\hat{e}(P, Q) = e(P, \delta(Q))$
  - $\Rightarrow$ Symmetric pairing (BN cannot be used with all protocols)

- **Ordinary curve** $\Rightarrow$ No distortion map

- **Modular arithmetic** $\Rightarrow$ No carry, better suited to hardware implementation
Supersingular elliptic curves vs. Barreto–Naehrig curves

Definition:

\[ E / \mathbb{F}_3 : \quad y^2 = x^3 - x + b, \quad b \neq 0 \]

Supersingular curve ⇒ Simpler curve arithmetic

Definition:

\[ E / \mathbb{F}_p : \quad y^2 = x^3 + b, \quad b \neq 0, \]
\[ p = 36\alpha^4 - 36\alpha^3 + 24\alpha^2 - 6\alpha + 1 \]

Ordinary curve (efficient tripling formulae)

Distortion map, modified pairing:

\[ \delta : E(\mathbb{F}_q)[\ell] \rightarrow E(\mathbb{F}_{q^k})[\ell] \]
\[ \hat{e}(P, Q) = e(P, \delta(Q)) \]
⇒ Symmetric pairing (BN cannot be used with all protocols)

No distortion map

Small characteristic field arithmetic ⇒ No carry, better suited to hardware implementation

Small embedding degree \((k = 6)\) ⇒ Larger field of definition for the same security level.
\[ \mathbb{F}_q \text{ with } q \approx 3^{500} \]

Optimal embedding degree \((k = 12)\)
\[ \mathbb{F}_p \text{ with } p \text{ a 256-bit prime.} \]
Supersingular elliptic curves

- **Definition:**

  \[ E / \mathbb{F}_3 : \quad y^2 = x^3 - x + b, \ b \neq 0 \]

- **Supersingular curve**

  \[ \Rightarrow \text{Simpler curve arithmetic (efficient tripling formulae)} \]

- **Distortion map, modified pairing:**

  \[
  \delta : E(\mathbb{F}_q)[\ell] \to E(\mathbb{F}_{q^k})[\ell]
  \]

  \[
  \hat{e}(P, Q) = e(P, \delta(Q))
  \]

  \[ \Rightarrow \text{Symmetric pairing} \]

- **Small characteristic field arithmetic**

  \[ \Rightarrow \text{No carry, better suited to hardware implementation} \]

- **Small embedding degree (} k = 6) \]

  \[ \Rightarrow \text{Larger field of definition for the same security level.} \]

  \[ \mathbb{F}_q \text{ with } q \approx 3^{500} \]
Agenda

1. Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
Computing the reduced Tate pairing
\( \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}^\times_{p^{km}} \)
Computing the reduced Tate pairing

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)
Computing the reduced Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^km} \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^km}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)
Computing the reduced Tate pairing
\[ \hat{e} : E(\mathbb{F}_p^m)[\ell] \times E(\mathbb{F}_p^m)[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_p^{\times} \]

- Arithmetic over \( \mathbb{F}_p^m \):
  - polynomial basis: \( \mathbb{F}_p^m \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_p^{\times}_{km} \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_p^m \)

- Operations over \( \mathbb{F}_p^m \):
  - \( O(m) \) additions / subtractions
  - \( O(m) \) multiplications
  - \( O(m) \) Frobenius maps (\( a \mapsto a^p \), i.e. squarings or cubings)
  - 1 inversion
Computing the reduced Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^{km}}^{\times} \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^{km}}^{\times} \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)

- Operations over \( \mathbb{F}_{p^m} \):
  - \( O(m) \) additions / subtractions
  - \( O(m) \) multiplications
  - \( O(m) \) Frobenius maps (\( a \mapsto a^p \), i.e. squarings or cubings)
  - 1 inversion

- A first idea: an all-in-one unified operator:
  - shared resources
  - scalable architecture
Agenda

1. Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
Arithmetic over $\mathbb{F}_{3^m}$

- $f \in \mathbb{F}_3[x]$: degree-$m$ irreducible polynomial over $\mathbb{F}_3$

\[
f = x^m + f_{m-1}x^{m-1} + \cdots + f_1x + f_0
\]
Arithmetic over $\mathbb{F}_{3^m}$

- $f \in \mathbb{F}_3[x]$: degree-$m$ irreducible polynomial over $\mathbb{F}_3$
  
  $$f = x^m + f_{m-1}x^{m-1} + \cdots + f_1x + f_0$$

- $\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f)$

- $a \in \mathbb{F}_{3^m}$:
  
  $$a = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$$

- Each element of $\mathbb{F}_3$ stored using two bits [also called trits]
Addition over $\mathbb{F}_{3^m}$

- $r = a + b = (a_{m-1} + b_{m-1})x^{m-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0)$
Addition over $\mathbb{F}_3^m$

- $r = a + b = (a_{m-1} + b_{m-1})x^{m-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0)$
  - coefficient-wise additions over $\mathbb{F}_3$: $r_i = (a_i + b_i) \mod 3$
Addition over $\mathbb{F}_{3^m}$

- $r = a + b = (a_{m-1} + b_{m-1})x^{m-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0)$
  - coefficient-wise additions over $\mathbb{F}_3$: $r_i = (a_i + b_i) \mod 3$
  - addition over $\mathbb{F}_3$: small look-up tables
Addition, subtraction and accumulation over $\mathbb{F}_{3^m}$

- **sign selection**: multiplication by 1 or 2
- $-a \equiv 2a \pmod{3}$
- **feedback loop for accumulation**
Multiplication over $\mathbb{F}_{3^m}$

- Parallel-serial multiplication
  - multiplicand loaded in a parallel register
  - multiplier loaded in a shift register
- Most significant coefficients first (Horner scheme)
- $D$ coefficients processed at each clock cycle: $\left\lceil \frac{m}{D} \right\rceil$ cycles per multiplication
Multiplication over $\mathbb{F}_{3^m}$

- Example for $D = 3$ (3 coefficients per iteration):

```
x^{m-1} \quad \ldots \quad x^2 \quad x \quad 1

\times

\quad a
\quad b

\quad b_{m-1} \cdot a \cdot x^2
\quad b_{m-2} \cdot a \cdot x
\quad b_{m-3} \cdot a

\quad a
\quad b

\quad (b_{m-1} \cdot a \cdot x^2) \mod f
\quad (b_{m-2} \cdot a \cdot x) \mod f
\quad b_{m-3} \cdot a

\quad r \quad \text{(partial sum)}
```
Multiplication over $\mathbb{F}_{3^m}$

- Computing the partial products $b_j \cdot a$:
  - coefficient-wise multiplication over $\mathbb{F}_3$: $(b_j \cdot a_i) \mod 3$
  - multiplications over $\mathbb{F}_3$: small look-up tables
Multiplication over $\mathbb{F}_3^m$

- Computing the partial products $b_j \cdot a$:
  - coefficient-wise multiplication over $\mathbb{F}_3$: $(b_j \cdot a_i) \mod 3$
  - multiplications over $\mathbb{F}_3$: small look-up tables

- Multiplication by $x^j$: simple shift (only wires)
Multiplication over $\mathbb{F}_{3^m}$

- Computing the partial products $b_j \cdot a$:
  - coefficient-wise multiplication over $\mathbb{F}_3$: $(b_j \cdot a_i) \mod 3$
  - multiplications over $\mathbb{F}_3$: small look-up tables

- Multiplication by $x^j$: simple shift (only wires)

- Modulo $f$ reduction:
  - $f = x^m + f_{m-1}x^{m-1} + \cdots + f_1x + f_0$ gives
    $$x^m \equiv (-f_{m-1})x^{m-1} + \cdots + (-f_1)x + (-f_0) \pmod{f}$$
  - highest degree of polynomial to reduce: $m + D - 1$
  - if $f$ is carefully selected (e.g. a trinomial or pentanomial), only a few multiplications and additions over $\mathbb{F}_3$
Multiplication over $\mathbb{F}_{3^m}$

- Computing the partial products $b_j \cdot a$:
  - coefficient-wise multiplication over $\mathbb{F}_3$: $(b_j \cdot a_i) \mod 3$
  - multiplications over $\mathbb{F}_3$: small look-up tables
- Multiplication by $x^j$: simple shift (only wires)
- Modulo $f$ reduction:
  - $f = x^m + f_{m-1}x^{m-1} + \cdots + f_1x + f_0$ gives
    \[ x^m \equiv (-f_{m-1})x^{m-1} + \cdots + (-f_1)x + (-f_0) \pmod{f} \]
  - highest degree of polynomial to reduce: $m + D - 1$
  - if $f$ is carefully selected (e.g. a trinomial or pentanomial), only a few multiplications and additions over $\mathbb{F}_3$
  - example for $m = 97$: $f = x^{97} + x^{12} + 2$
Multiplication over $\mathbb{F}_{3^m}$

- Example for $D = 3$ (3 coefficients per iteration):
Frobenius map over $\mathbb{F}_{3^m}$: cubing

- Let $A$ be an arbitrary element of the field $\mathbb{F}_{3^m}$, that in canonical basis can be written as, $A = \sum_{i=0}^{m-1} a_i x^i$, $a_i \in \mathbb{F}_3$. Then, the polynomial cubing $A^3$, can be computed as,

$$A^3 = \left( \sum_{i=0}^{m-1} a_i x^i \right)^3 = \sum_{i=0}^{m-1} a_i x^{3i}$$

$$= \sum_{i=0}^{u} a_i x^{3i} + \sum_{i=u+1}^{2u+r-1} a_i x^{3i} + \sum_{i=2u+r}^{3u+r-1} a_i x^{3i}$$

$$= C_0 + x^{3u+r} C_1 + x^{6u+2r} C_2 = C_0 + x^m C_1 + x^{2m} C_2.$$

- Symbolic computation of the reduction:
  each coefficient of the result is a linear combination of the $a_i$’s

$$a^3 \mod f = \sum_{j=0}^{n-1} w_j \cdot \mu_j$$

with $w_j \in \mathbb{F}_3$, $\mu_j \in \mathbb{F}_{3^m}$, and $\mu_{j,i} \in \{0\} \cup \{a_{m-1}, \ldots, a_1, a_0\}$.
Frobenius map over $\mathbb{F}_{3^m}$

- Example for $m = 97$ and $f = x^{97} + x^{12} + 2$:

$$a^3 \mod f = (a_{32}x^{96} + a_{64}x^{95} + a_{96}x^{94} + \cdots + a_{33}x^2 + a_{65}x + a_0) \times 1$$
$$+ (0 + 0 + a_{88}x^{94} + \cdots + 0 + 0 + a_{89}) \times 1$$
$$+ (0 + 0 + a_{92}x^{94} + \cdots + 0 + 0 + a_{93}) \times 1$$
$$+ (0 + a_{60}x^{95} + 0 + \cdots + 0 + a_{61}x + 0) \times 2$$

Required hardware:
- Only wires to compute the $\mu_j$'s
- Multiplications over $\mathbb{F}_3$ for the weights $w_j$
- Multi-operand addition over $\mathbb{F}_3^m$
Example for $m = 97$ and $f = x^{97} + x^{12} + 2$:

$$a^3 \mod f = \left( a_{32}x^{96} + a_{64}x^{95} + a_{96}x^{94} + \cdots + a_{33}x^2 + a_{65}x + a_0 \right) \times 1$$

$$+ \left( 0 + 0 + a_{88}x^{94} + \cdots + 0 + 0 + a_{89} \right) \times 1$$

$$+ \left( 0 + 0 + a_{92}x^{94} + \cdots + 0 + 0 + a_{93} \right) \times 1$$

$$+ \left( 0 + a_{60}x^{95} + 0 + \cdots + 0 + a_{61}x + 0 \right) \times 2$$

$$= \left( a_{32}x^{96} + a_{64}x^{95} + a_{96}x^{94} + \cdots + a_{33}x^2 + a_{65}x + a_0 \right) \times 1$$

$$+ \left( 0 + a_{60}x^{95} + a_{88}x^{94} + \cdots + 0 + a_{61}x + a_{89} \right) \times 1$$

$$+ \left( 0 + a_{60}x^{95} + a_{92}x^{94} + \cdots + 0 + a_{61}x + a_{93} \right) \times 1$$
Frobenius map over $\mathbb{F}_{3^m}$

- Example for $m = 97$ and $f = x^{97} + x^{12} + 2$:

$$a^3 \mod f = (a_{32}x^{96} + a_{64}x^{95} + a_{96}x^{94} + \cdots + a_{33}x^2 + a_{65}x + a_0) \times 1$$

$$+ (0 + 0 + a_{88}x^{94} + \cdots + 0 + 0 + a_{89}) \times 1$$

$$+ (0 + 0 + a_{92}x^{94} + \cdots + 0 + 0 + a_{93}) \times 1$$

$$+ (0 + a_{60}x^{95} + 0 + \cdots + 0 + a_{61}x + 0) \times 2$$

$$= (a_{32}x^{96} + a_{64}x^{95} + a_{96}x^{94} + \cdots + a_{33}x^2 + a_{65}x + a_0) \times 1$$

$$+ (0 + a_{60}x^{95} + a_{88}x^{94} + \cdots + 0 + a_{61}x + a_{89}) \times 1$$

$$+ (0 + a_{60}x^{95} + a_{92}x^{94} + \cdots + 0 + a_{61}x + a_{93}) \times 1$$

- Required hardware:
  - only wires to compute the $\mu_j$'s
  - multiplications over $\mathbb{F}_3$ for the weights $w_j$
  - multi-operand addition over $\mathbb{F}_{3^m}$
Inverse Frobenius map over $\mathbb{F}_{3^m}$

Let $A$ be an arbitrary element of the field $\mathbb{F}_{3^m}$, that in canonical basis can be written as,

$$A = \sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{u} a_{3i} x^{3i} + x \cdot \sum_{i=0}^{u+r-2} a_{3i+1} x^{3i} + x^2 \cdot \sum_{i=0}^{u+r-2} a_{3i+2} x^{3i}.$$ 

Then, the cube root $\sqrt[3]{A}$, can be computed as [barreto04],

$$\sqrt[3]{A} = \sum_{i=0}^{u} a_{3i} x^i + x^{1/3} \cdot \sum_{i=0}^{u+r-2} a_{3i+1} x^i + x^{2/3} \cdot \sum_{i=0}^{u+r-2} a_{3i+2} x^i. \quad (2)$$

One can compute a cube root by finding the per-field constants $x^{1/3}$ and $x^{2/3}$. 
Irreducible Trinomials \( P(x) = x^m - x^k + 1 \), with \( m \equiv k \equiv r \mod 3 \)

Let us consider the ternary field \( \mathbb{F}_{3^m} \) generated by the trinomial \( P(x) = x^m - x^k + 1 \), irreducible over \( \mathbb{F}_3 \). Let us also assume that the extension degree \( m \) can be expressed as, \( m = 3u + r \), \( u \geq 1 \) and \( k = 3v + r \), \( 0 \leq v \leq u \), with \( m \equiv k \equiv r \mod 3 \), \( r \neq 0 \) and \( u - 2v \geq 1 \). In [barreto04] it was found that for \( r = 1 \) we have,

\[
x^{2/3} = -x^{u+1} + x^{v+1}; \quad x^{1/3} = x^{2u+1} + x^{u+v+1} + x^{2v+1}.
\]

whereas for \( r = 2 \) we have,

\[
x^{1/3} = -x^{u+1} + x^{v+1}; \quad x^{2/3} = x^{2u+2} + x^{u+v+2} + x^{2v+2}.
\]
Frobenius map over $\mathbb{F}_{3^m}$

- feedback loop for successive cubings
- sign selection for computing either $a^3$ or $-a^3$
Inversion over $\mathbb{F}_{3^m}$

- Extended Euclidean Algorithm?

Our solution: Fermat's little theorem

\[ a^{m-2} = a^{3m-2} \text{ on } \mathbb{F}_{3^m} (a \neq 0) \]

- Algorithm by Itoh and Tsujii
- Requires only multiplications and cubings over $\mathbb{F}_{3^m}$
- Only one inversion for the full pairing: delay overhead is negligible ($< 1\%$)
Inversion over $\mathbb{F}_{3^m}$

- Extended Euclidean Algorithm?
  - fast computation
  - ... but need for additional hardware

\[ a - 1 = a^{3m - 2} \text{ on } \mathbb{F}_{3^m} (a \neq 0) \]

Our solution:
- algorithm by Itoh and Tsujii
- requires only multiplications and cubings over $\mathbb{F}_{3^m}$
- only one inversion for the full pairing: delay overhead is negligible ($< 1\%$)
Inversion over $\mathbb{F}_{3^m}$

- Extended Euclidean Algorithm?
  - fast computation
  - ... but need for additional hardware
- Our solution: Fermat’s little theorem

$$a^{-1} = a^{3^m-2} \quad \text{on } \mathbb{F}_{3^m} \ (a \neq 0)$$
Inversion over $\mathbb{F}_{3^m}$

- Extended Euclidean Algorithm?
  - fast computation
  - ... but need for additional hardware

- Our solution: Fermat’s little theorem

$$a^{-1} = a^{3^m-2} \quad \text{on } \mathbb{F}_{3^m} \ (a \neq 0)$$

- algorithm by Itoh and Tsujii
- requires only multiplications and cubings over $\mathbb{F}_{3^m}$
Inversion over $\mathbb{F}_{3^m}$

- **Extended Euclidean Algorithm?**
  - **fast computation**
  - ... but need for additional hardware

- **Our solution: Fermat’s little theorem**
  
  $$a^{-1} = a^{3^m - 2} \quad \text{on } \mathbb{F}_{3^m} (a \neq 0)$$

  - algorithm by Itoh and Tsujii
  - requires only multiplications and cubings over $\mathbb{F}_{3^m}$
  - only one inversion for the full pairing: delay overhead is negligible ($< 1\%$)
The full processing element

For the Tate pairing:

- limited parallelism between additions, multiplications, and Frobenius maps

Can we share hardware resources between the three operators?

Francisco Rodríguez-Henríquez

Hardware Implementation of Pairings (50 / 91)
What can we share?

- Input and output registers
- Partial product generators:
  - sign selection for the addition / subtraction
  - partial products for the multiplication
  - multiplication by the $w_j$'s for the Frobenius map
- Multi-operand addition tree
- Feedback loops for accumulation
Our unified operator
Field Towering $\mathbb{F}_{3^{6m}}$

$2m$ bits

$\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f(x))$

$\mathbb{F}_3 \cong \mathbb{Z}/3\mathbb{Z} = \{0, 1, 2\}$
Field Towering $\mathbb{F}_{3^{6m}}$

$\mathbb{F}_{3^{2m}} \cong \mathbb{F}_{3^m}[\sigma]/(\sigma^2 + 1)$

$\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f(x))$

$\mathbb{F}_3 \cong \mathbb{Z}/3\mathbb{Z} = \{0, 1, 2\}$
Field Towering $\mathbb{F}_{3^6m}$

$\mathbb{F}_{3^6m} \cong \mathbb{F}_{3^2m}[\rho]/(\rho^3 - \rho - b)$

$\mathbb{F}_{3^2m} \cong \mathbb{F}_{3^m}[\sigma]/(\sigma^2 + 1)$

$\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f(x))$

$\mathbb{F}_3 \cong \mathbb{Z}/3\mathbb{Z} = \{0, 1, 2\}$
Field Towering $\mathbb{F}_{3^{6m}}$
Computation of the Tate pairing

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \rightarrow \mu_\ell \subseteq \mathbb{F}_{p^{km}}^* \]
Computation of the Tate pairing
\( \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^km}^\times \)

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)
Computation of the Tate pairing
\( \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^km} \)

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)
- Arithmetic over \( \mathbb{F}_{p^km}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)
Computation of the Tate pairing

\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^{km}}^{\times} \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)
- Arithmetic over \( \mathbb{F}_{p^{km}}^{\times} \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)
- Operations over \( \mathbb{F}_{p^m} \):

<table>
<thead>
<tr>
<th>Base field (( \mathbb{F}_{p^m} ))</th>
<th>( \mathbb{F}_{2^m} )</th>
<th>( \mathbb{F}_{2^{313}} )</th>
<th>( \mathbb{F}_{3^m} )</th>
<th>( \mathbb{F}_{3^{127}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+/-)</td>
<td>27(\left\lfloor \frac{m}{2} \right\rfloor ) + 75</td>
<td>4287</td>
<td>119(\left\lfloor \frac{m}{4} \right\rfloor ) + 260</td>
<td>3949</td>
</tr>
<tr>
<td>(\times)</td>
<td>7(\left\lfloor \frac{m}{2} \right\rfloor ) + 29</td>
<td>1121</td>
<td>25(\left\lfloor \frac{m}{4} \right\rfloor ) + 93</td>
<td>868</td>
</tr>
<tr>
<td>(a^p)</td>
<td>6m + 9</td>
<td>1887</td>
<td>17(\left\lfloor \frac{m}{2} \right\rfloor ) + 8</td>
<td>1079</td>
</tr>
<tr>
<td>(a^{-1})</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Computation of the Tate pairing
\[ \hat{e} : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^m})[\ell] \to \mu_\ell \subseteq \mathbb{F}_{p^{km}}^\times \]

- Arithmetic over \( \mathbb{F}_{p^m} \):
  - polynomial basis: \( \mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/(f(x)) \)
  - \( f(x) \), degree-\( m \) polynomial irreducible over \( \mathbb{F}_p \)

- Arithmetic over \( \mathbb{F}_{p^{km}}^\times \):
  - tower-field representation
  - only arithmetic over the underlying field \( \mathbb{F}_{p^m} \)

- Operations over \( \mathbb{F}_{p^m} \):

<table>
<thead>
<tr>
<th>Base field ( (\mathbb{F}_{p^m}) )</th>
<th>( \mathbb{F}_2^m )</th>
<th>( \mathbb{F}_{2^{313}} )</th>
<th>( \mathbb{F}_3^m )</th>
<th>( \mathbb{F}_{3^{127}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+/-)</td>
<td>27( \lceil \frac{m}{2} \rceil + 75 )</td>
<td>4287</td>
<td>119( \lceil \frac{m}{4} \rceil + 260 )</td>
<td>3949</td>
</tr>
<tr>
<td>( \times )</td>
<td>7( \lceil \frac{m}{2} \rceil + 29 )</td>
<td>1121</td>
<td>25( \lceil \frac{m}{4} \rceil + 93 )</td>
<td>868</td>
</tr>
<tr>
<td>( a^p )</td>
<td>6( m + 9 )</td>
<td>1887</td>
<td>17( \lceil \frac{m}{2} \rceil + 8 )</td>
<td>1079</td>
</tr>
<tr>
<td>( a^{-1} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Software not well suited to small characteristic: need for hardware acceleration
Agenda

1. Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
The best area-time product of the literature...
... But still quite slow
(or not the fastest, at least!)
Agenda

1. Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
Weil Descent-based attacks

- We now consider:
  \[ E(\mathbb{F}_{3^m \cdot n})[\ell] \text{ with } m \text{ prime and } n \text{ small} \]

- Weil descent (or Weil restriction to scalar) apply:
  \[ E(\mathbb{F}_{3^m \cdot n}) \simeq W_E(\mathbb{F}_{3^m}) \]
Weil Descent-based attacks

- We now consider:
  \[ E(\mathbb{F}_{3^m \cdot n})[\ell] \] with \( m \) prime and \( n \) small

- Weil descent (or Weil restriction to scalar) apply:
  \[ E(\mathbb{F}_{3^m \cdot n}) \cong W_E(\mathbb{F}_{3^m}) \]

- Gaudry–Hess–Smart attack:
  - \( W_E(\mathbb{F}_{3^m}) \) might map to \( \text{Jac}(C) \), with \( C \) a curve of genus at least \( n \)
  - index calculus algorithm: solve DLP in \( O(3^m)^{2 - \frac{2}{n}} \)
Weil Descent-based attacks

- We now consider:
  \[ E(\mathbb{F}_{3^m \cdot n})[\ell] \text{ with } m \text{ prime and } n \text{ small} \]

- Weil descent (or Weil restriction to scalar) apply:
  \[ E(\mathbb{F}_{3^m \cdot n}) \cong W_E(\mathbb{F}_{3^m}) \]

- Gaudry–Hess–Smart attack:
  - \( W_E(\mathbb{F}_{3^m}) \) might map to \( \text{Jac}(C) \), with \( C \) a curve of genus at least \( n \)
  - index calculus algorithm: solve DLP in \( O(3^m)^{2-\frac{2}{n}} \)

- Static Diffie–Hellman problem
  - leakage when reusing private key (e.g. ElGamal encryption)
  - Granger’s attack: complexity in \( O(3^m)^{1-\frac{1}{n+1}} \)
  - revoke key after a certain amount of use is an effective workaround
Suitable curves for 128-bit security level

<table>
<thead>
<tr>
<th>$p^m$</th>
<th>$n$</th>
<th>$\log_2 \ell$</th>
<th>Pollard’s $\rho$</th>
<th>FFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^{503}$</td>
<td>1</td>
<td>697</td>
<td>342</td>
<td>132</td>
</tr>
<tr>
<td>$3^{97}$</td>
<td>5</td>
<td>338</td>
<td>163</td>
<td>130</td>
</tr>
<tr>
<td>$3^{67}$</td>
<td>7</td>
<td>612</td>
<td>300</td>
<td>129</td>
</tr>
<tr>
<td>$3^{53}$</td>
<td>11</td>
<td>672</td>
<td>330</td>
<td>140</td>
</tr>
<tr>
<td>$3^{43}$</td>
<td>13</td>
<td>764</td>
<td>376</td>
<td>138</td>
</tr>
</tbody>
</table>
## Suitable curves for 128-bit security level

<table>
<thead>
<tr>
<th>$p^m$</th>
<th>$n$</th>
<th>$\log_2 \ell$</th>
<th>Pollard’s $\rho$</th>
<th>FFS</th>
<th>GHS</th>
<th>SDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^{503}$</td>
<td>1</td>
<td>697</td>
<td>342</td>
<td>132</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$3^{97}$</td>
<td>5</td>
<td>338</td>
<td>163</td>
<td>130</td>
<td>245</td>
<td>128</td>
</tr>
<tr>
<td>$3^{67}$</td>
<td>7</td>
<td>612</td>
<td>300</td>
<td>129</td>
<td>182</td>
<td>92</td>
</tr>
<tr>
<td>$3^{53}$</td>
<td>11</td>
<td>672</td>
<td>330</td>
<td>140</td>
<td>152</td>
<td>77</td>
</tr>
<tr>
<td>$3^{43}$</td>
<td>13</td>
<td>764</td>
<td>376</td>
<td>138</td>
<td>125</td>
<td>63</td>
</tr>
</tbody>
</table>
Suitable curves for 128-bit security level

<table>
<thead>
<tr>
<th>$p^m$</th>
<th>$n$</th>
<th>$\log_2 \ell$</th>
<th>Pollard’s $\rho$</th>
<th>FFS</th>
<th>GHS</th>
<th>SDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^{503}$</td>
<td>1</td>
<td>697</td>
<td>342</td>
<td>132</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$3^{97}$</td>
<td>5</td>
<td>338</td>
<td>163</td>
<td>130</td>
<td>245</td>
<td>128</td>
</tr>
<tr>
<td>$3^{67}$</td>
<td>7</td>
<td>612</td>
<td>300</td>
<td>129</td>
<td>182</td>
<td>92</td>
</tr>
<tr>
<td>$3^{53}$</td>
<td>11</td>
<td>672</td>
<td>330</td>
<td>140</td>
<td>152</td>
<td>77</td>
</tr>
<tr>
<td>$3^{43}$</td>
<td>13</td>
<td>764</td>
<td>376</td>
<td>138</td>
<td>125</td>
<td>63</td>
</tr>
</tbody>
</table>
Experimental setup

- Full Tate pairing computation over $E(\mathbb{F}_{3^{97.5}})$

<table>
<thead>
<tr>
<th></th>
<th>$\times$</th>
<th>$+$</th>
<th>$(.)^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{3^{97}}$</td>
<td>37289</td>
<td>253314</td>
<td>21099</td>
</tr>
</tbody>
</table>
Experimental setup

- Full Tate pairing computation over $E(\mathbb{F}_{3^{97 \cdot 5}})$

<table>
<thead>
<tr>
<th></th>
<th>$\times$</th>
<th>$+$</th>
<th>$(\cdot)^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{3^{97}}$</td>
<td>37289</td>
<td>253314</td>
<td>21099</td>
</tr>
</tbody>
</table>

- Finite field coprocessor
  - Prototyped on Xilinx Virtex-4 LX FPGAs
  - Post-place-and-route timing and area estimations
Calculation time

![Graph showing calculation time vs. security bits]

- 4.47 ms / $\mathbb{F}_{2^{163.7}}$
- 2.11 ms / $\mathbb{F}_{3^{97.5}}$

Security [bits]

Calculation time [$\mu$s]
Motivations
Motivations

- **High speed** is more important than **low resources** for some cryptographic applications
- Explore the other end of the **area vs. time tradeoff**:
  - faster but larger than the unified operator
  - what about the **area-time product**?
Motivations

- High speed is more important than low resources for some cryptographic applications
- Explore the other end of the area vs. time tradeoff:
  - faster but larger than the unified operator
  - what about the area-time product?
- Accelerate the computation by extracting as much parallelism as possible...
- ... Without increasing dramatically the resource requirements
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) \]
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  $$\hat{e}(P, Q) = \eta_T(P, Q)$$
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps:

$$\hat{e}(P, Q) = \eta_T(P, Q)^M$$
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  - via Miller’s algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^m}^\times$, with $N = \#E(\mathbb{F}_{p^m})$
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  - via Miller’s algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^k}^\times$, with $N = \# E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^k - 1)/N$
  - required to obtain a unique value for each congruence class
  - example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):
    \[ M = \frac{3^{6m} - 1}{3^m + 1 \pm 3^{(m+1)/2}} = (3^{3m} - 1)(3^m + 1)(3^m + 1 \mp 3^{(m+1)/2}) \]
  - exploit the special form of the exponent: *ad-hoc* algorithm
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[ \hat{e}(P, Q) = \eta_T(P, Q)^M \]

- Computation of the $\eta_T$ pairing
  - via Miller's algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^km}^\times$, with $N = \#E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^{km} - 1)/N$
  - required to obtain a unique value for each congruence class
  - example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):
    \[ M = \frac{3^{6m} - 1}{3^m + 1 \pm 3^{(m+1)/2}} = (3^{3m} - 1)(3^m + 1)(3^m + 1 \mp 3^{(m+1)/2}) \]
  - exploit the special form of the exponent: ad-hoc algorithm

- Two distinct computational requirements
Computation of the $\eta_T$ pairing

- The Tate pairing over $E(\mathbb{F}_{p^m})$ is computed in two main steps
  \[
  \hat{e}(P, Q) = \eta_T(P, Q)^M
  \]

- Computation of the $\eta_T$ pairing
  - via Miller’s algorithm: loop of $(m + 1)/2$ iterations
  - result only defined modulo $N$-th powers in $\mathbb{F}_{p^km}^\times$, with $N = \#E(\mathbb{F}_{p^m})$

- Final exponentiation by $M = (p^{km} - 1)/N$
  - required to obtain a unique value for each congruence class
  - example in characteristic 3 ($k = 6$ and $N = 3^m + 1 \pm 3^{(m+1)/2}$):
    \[
    M = \frac{3^{6m} - 1}{3^m + 1 \pm 3^{(m+1)/2}} = (3^{3^m} - 1)(3^m + 1)\left(3^m + 1 \mp 3^{(m+1)/2}\right)
    \]
    - exploit the special form of the exponent: *ad-hoc* algorithm

- Two distinct computational requirements $\Rightarrow$ use two distinct coprocessors
Reduced Tate pairing

\[ \mu_\ell \subseteq F \times 3^6m \]

Non-reduced pairing (iterative computation)

Final algorithm

Input: two points \( P \) and \( Q \) in \( E(F^{3m})[\ell] \)

Output: an \( \ell \)-th root of unity in the extension \( F \times 3^6m \)

Two very different steps

Francisco Rodríguez-Henríquez
Reduced Tate pairing

- **Input:** two points $P$ and $Q$ in $E(F_{3^m})[\ell]$
Reduced Tate pairing

- **Input**: two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$
- **Output**: an $\ell$-th root of unity in the extension $\mathbb{F}_{3^{6m}}$
Reduced Tate pairing

- **Input:** two points $P$ and $Q$ in $E(\mathbb{F}_{3^m})[\ell]$
- **Output:** an $\ell$-th root of unity in the extension $\mathbb{F}_{3^6m}^\times$
- **Two very different steps**
Reduced Tate pairing

- **Input**: two points $P$ and $Q$ in $E(F_{3^m})[\ell]$
- **Output**: an $\ell$-th root of unity in the extension $F_{3^6m}^\times$
- **Two very different steps**
Reduced Tate pairing

- Input: two points $P$ and $Q$ in $E(F_{3^m})[\ell]$
- Output: an $\ell$-th root of unity in the extension $F_{3^6m}^\times$
- Two very different steps
Reduced Tate pairing

- **Input**: two points \( P \) and \( Q \) in \( E(\mathbb{F}_{3^m})[\ell] \)
- **Output**: an \( \ell \)-th root of unity in the extension \( \mathbb{F}_{3^6m}^\times \)
- **Two very different steps**
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
- Pipeline the data between the two coprocessors
Two coprocessors for the $\eta_T$ pairing

- The two operations are purely sequential
- Only one active coprocessor at every moment
- Pipeline the data between the two coprocessors
  - both of them are kept busy
  - higher throughput
Two coprocessors for the $\eta_T$ pairing

- The two operations are **purely sequential**
- Only **one active coprocessor at every moment**
- **Pipeline the data between the two coprocessors**
  - both of them are kept busy
  - higher throughput
- **Balance the computation time between the two coprocessors**
ηT pairing algorithm

\[ η_T : E(\mathbb{F}_{3^m})[\ell] \times E(\mathbb{F}_{3^m})[\ell] \rightarrow \mathbb{F}_{36m}^× \]

- **Three tasks** per iteration:
  1. update the coordinates
  2. compute the line equation
  3. accumulate the new factor

- **Total cost:** 17 ×, 4 Frobenius/inverse Frobenius and 30 + over \( \mathbb{F}_{3^m} \)
\( \eta_T \) pairing algorithm

\[
\eta_T : E(\mathbb{F}_{3^m})[\ell] \times E(\mathbb{F}_{3^m})[\ell] \to \mathbb{F}_{3^{6m}}^\times
\]

- **Three tasks** per iteration:
  1. update the coordinates
  2. compute the line equation
  3. accumulate the new factor

- **Total cost:** \( 17 \times, 4 \) Frobenius/inverse Frobenius and \( 30 + \) over \( \mathbb{F}_{3^m} \)

- **Cost of the inverse Frobenius:** Same as the Frobenius

```plaintext
for i ← 0 to (m - 1)/2 do
  1. \( x_P \leftarrow \sqrt[3]{x_P} \); \( y_P \leftarrow \sqrt[3]{y_P} \)
  2. \( x_Q \leftarrow x_Q^3 \); \( y_Q \leftarrow y_Q^3 \)
  3. \( t \leftarrow x_P + x_Q \); \( u \leftarrow y_P y_Q \)
  4. \( S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 \)
  5. \( R \leftarrow R \cdot S \)
end for
```

2 inv. Frobenius
2 Frobenius
\( (\mathbb{F}_{3^m}) \)

\( 2 \times, 1 + \) (\( \mathbb{F}_{3^m} \))

15 \times, 29 + (\( \mathbb{F}_{3^m} \))
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times$, 2 Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+$: cheap and fast operations
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times 2$ Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+:\text{cheap and fast operations}$
  - critical operation: $\times$
- Need for a fast parallel multiplier: Karatsuba
Accelerating the $\eta_T$ pairing

- Total cost: $17 \times$, 2 Frobenius and inverse Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration
  - Frobenius/inverse Frobenius and $+$: cheap and fast operations
  - Critical operation: $\times$

- Need for a fast parallel multiplier: Karatsuba

\[
A^H B^L + A^L B^H = (A^H + A^L)(B^H + B^L) - A^H B^H - A^L B^L
\]
A 128-bit three-stage pipelined Karatsuba multiplier architecture
A parallel Karatsuba multiplier

- **fully parallel**: all sub-products are computed in parallel
- **pipelined architecture**: higher clock frequency, one product per cycle
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
- sub-products recursively implemented as Karatsuba-Ofman multipliers
A parallel Karatsuba multiplier

- **fully parallel**: all sub-products are computed in parallel
- **pipelined architecture**: higher clock frequency, one product per cycle
- **sub-products recursively implemented as Karatsuba-Ofman multipliers**
- **support for other variants**: odd-even split, 3-way split,
A parallel Karatsuba multiplier

- fully parallel: all sub-products are computed in parallel
- pipelined architecture: higher clock frequency, one product per cycle
- sub-products recursively implemented as Karatsuba-Ofman multipliers
- support for other variants: odd-even split, 3-way split, ...
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle

Challenge: keep the multiplier busy at all times
Careful scheduling to avoid pipeline bubbles:
- ensure that multiplication operands are always available
- avoid memory congestion issues

We managed to accomplish that: our processor computes Miller loop in just $17 \cdot (m + 3) / 2$ clock cycles (considering the initialization phase)
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle

- Challenge: keep the multiplier busy at all times

We managed to accomplish that: our processor computes Miller loop in just $17 \cdot (m + 3)/2$ clock cycles (considering the initialization phase).
Accelerating the $\eta_T$ pairing

- $\eta_T$ coprocessor based on a single large multiplier:
  - parallel Karatsuba architecture
  - 7-stage pipeline
  - one product per cycle
- Challenge: keep the multiplier busy at all times
- Careful scheduling to avoid pipeline bubbles (idle cycles):
  - ensure that multiplication operands are always available
  - avoid memory congestion issues
- We managed to accomplish that: our processor computes Miller loop in just $17 \cdot (m + 3)/2$ clock cycles (considering the initialization phase)
Modified Algorithm

$$\eta_T : E(\mathbb{F}_{3^m})[\ell] \times E(\mathbb{F}_{3^m})[\ell] \rightarrow \mathbb{F}_{3^{6m}}$$

for $i \leftarrow 0$ to $(m - 1)/2$ do

1. $x_P \leftarrow x_P$ ; $y_P \leftarrow y_P$
   $x_Q \leftarrow x_Q^9 + 1$ ; $y_Q \leftarrow -y_Q^9$

2. $t \leftarrow x_P + x_Q$ ; $u \leftarrow y_P y_Q$
   $S \leftarrow -t^2 + u\sigma - t\rho - \rho^2$

3. $R \leftarrow R \cdot S$

4. $R \leftarrow R^3$

end for

for $i \leftarrow 0$ to $(m - 1)/2$ do

1. $x_P \leftarrow \sqrt[3]{x_P}$ ; $y_P \leftarrow \sqrt[3]{y_P}$
   $x_Q \leftarrow x_Q^3$ ; $y_Q \leftarrow y_Q^3$

2. $t \leftarrow x_P + x_Q$ ; $u \leftarrow y_P y_Q$
   $S \leftarrow -t^2 + u\sigma - t\rho - \rho^2$

3. $R \leftarrow R \cdot S$

end for
Modified Algorithm

\[ \eta_T : E(F_{3^m})[\ell] \times E(F_{3^m})[\ell] \rightarrow F_{3^{6m}}^\times \]

- Modified algorithm: 17 \times, 2 Frobenius, 2 inverse Frobenius and 30 + over \( F_{3^m} \)
- Previous algorithm: 17 \times, 10 Frobenius and 38 +
- Cost of the inverse Frobenius: Same as the Frobenius

```plaintext
for i ← 0 to (m − 1)/2 do

1. \( x_p \leftarrow \sqrt[3]{x_p} \); \( y_p \leftarrow \sqrt[3]{y_p} \)
   \( x_Q \leftarrow x_Q^3 \); \( y_Q \leftarrow y_Q^3 \)  
   2 inv. Frobenius
   2 Frobenius  \( (F_{3^m}) \)

2. \( t \leftarrow x_p + x_Q \); \( u \leftarrow y_p y_Q \)
   \( S \leftarrow -t^2 \pm u\sigma - t\rho - \rho^2 \)  
   2 \times, 1 +  \( (F_{3^m}) \)

3. \( R \leftarrow R \cdot S \)  
   15 \times, 29 +  \( (F_{3^m}) \)

end for
```
A parallel operator for the $\eta_T$ pairing
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{3^{6m}}$ and

$$M = (3^{3m} - 1)(3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{36m}^\times$ and

$$M = (3^{3m} - 1) (3^m + 1) \left( 3^m + 1 \mp 3^{(m+1)/2} \right)$$

- Operations over $\mathbb{F}_{3m}$: $73 \times, 3m + 3$ Frobenius, $3m + 175 +$, and $1$ inversion
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{36m}^\times$ and

$$M = (3^{3m} - 1)(3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$

- Operations over $\mathbb{F}_{3m}$: $73 \times$, $3m + 3$ Frobenius, $3m + 175 +$, and 1 inversion ($\sim \log m \times$ and $m - 1$ Frobenius)
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}_{36m}^\times$ and

$$M = (3^{3m} - 1) (3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)$$

- Operations over $\mathbb{F}_{3m}$: $73 \times$, $3m + 3$ Frobenius, $3m + 175 +$, and $1$ inversion ($\sim \log m \times$ and $m - 1$ Frobenius)

- Cost of the $\eta_T$ pairing:
  - $(m + 1)/2$ iterations
  - $17 \times$, $10$ Frobenius and $30 +$ over $\mathbb{F}_{3m}$ per iteration
The final exponentiation

- Compute $\hat{e}(P, Q)$ as $\eta_T(P, Q)^M$ with $\eta_T(P, Q) \in \mathbb{F}^{\times}_{3^{6m}}$ and
  \[
  M = (3^{3m} - 1) (3^m + 1) \left(3^m + 1 \mp 3^{(m+1)/2}\right)
  \]

- Operations over $\mathbb{F}_{3^m}$: $73 \times$, $3m + 3$ Frobenius, $3m + 175 +$, and $1$ inversion ($\sim \log m \times$ and $m - 1$ Frobenius)

- Cost of the $\eta_T$ pairing:
  - $(m + 1)/2$ iterations
  - $17 \times$, $10$ Frobenius and $30 +$ over $\mathbb{F}_{3^m}$ per iteration

- The final exponentiation is much cheaper than the $\eta_T$ pairing

- Challenge for the final exponentiation:
  - computation in the same time as the $\eta_T$ pairing
  - ... using as few resources as possible
The final exponentiation

- Design the **smallest architecture** possible supporting all the required operations over $\mathbb{F}_{3^m}$
- **purely sequential** scheduling

Although some parallelism is required. We found out that the usage of the inverse Frobenius operator is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly).
The final exponentiation

- Design the smallest architecture possible supporting all the required operations over $\mathbb{F}_{3^m}$
- purely sequential scheduling
- Although some parallelism is required.

We found out that the usage of the inverse Frobenius operator is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly)

New coprocessor with two arithmetic units:
- a standalone multiplier, based on a parallel-serial scheme
- a unified operator supporting addition/subtraction, inverse Frobenius map and inverse double Frobenius map
The final exponentiation

- Design the **smallest architecture** possible supporting all the required operations over $\mathbb{F}_{3^m}$
- **purely sequential** scheduling
- Although **some parallelism** is required.
- We found out that the usage of the **inverse Frobenius operator** is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly)
The final exponentiation

- Design the **smallest architecture** possible supporting all the required operations over $\mathbb{F}_{3^m}$
- purely sequential scheduling
- Although **some parallelism** is required.
- We found out that the usage of the inverse Frobenius operator is advantageous for computing the final exponentiation (as long as the irreducible polynomials are inverse-Frobenius friendly)
- New coprocessor with **two arithmetic units:**
  - a **standalone multiplier**, based on a parallel-serial scheme
  - a **unified operator** supporting addition/subtraction, inverse Frobenius map and inverse double Frobenius map
A coprocessor for the final exponentiation

Francisco Rodríguez-Henríquez

Hardware Implementation of Pairings
Agenda

1. Context and motivation
   - bilinear pairings defined over elliptic curves: Basic definitions
   - But.... Why should one bother implementing pairings in Hardware?
   - A quick overview of reconfigurable hardware devices

2. Computing the Tate Pairing
   - The Tate Pairing over Supersingular elliptic curves
   - The Tate Pairing over ordinary elliptic curves

3. Case of Study #1: A compact implementation of the $\eta_T$ pairing
   - Computing the reduced Tate pairing
   - Arithmetic over $\mathbb{F}_{3^m}$
   - Results Obtained

4. Case of Study #2: Estibals’ composite $\eta_T$ pairing
   - Attacks

5. Case of Study #3: A fast implementation of the $\eta_T$ pairing
   - Implementation Results in Hardware

6. Wish list on hardware implementation of pairings (Some concrete open problems)
Hardware accelerators

Calculation time [µs]

Security [bits]

Virtex-II Pro

Virtex-4 LX

6.2 µs / $\mathbb{F}_{397}$
12.8 µs / $\mathbb{F}_{3^{193}}$
16.9 µs / $\mathbb{F}_{3^{313}}$
675.5 µs / $\mathbb{F}_{2^{557}}$
100.8 µs / $\mathbb{F}_{2^{457}}$
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published. (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.
Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.

(a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.

However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.
Hardware implementation notes

- Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.
- (a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.
- However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.
- Although it was not discussed here, we also implemented the Tate pairing over char 2. Experimentally, we observed that our char 2 and char 3 accelerators achieve almost the same time performance.
Our Xilinx FPGA implementation, significantly improved the computation time of all the hardware pairing coprocessors for supersingular curves previously published.

(a bit Surprisingly) our architecture also enjoys the best area/time trade-off performance among supersingular pairing accelerators.

However, because we exceeded the FPGA’s capacity, we could only achieve up to 109 bits of security.

Although it was not discussed here, we also implemented the Tate pairing over char 2. Experimentally, we observed that our char 2 and char 3 accelerators achieve almost the same time performance.

In the design process of our char 2 accelerator we found the following undocumented family of square-root friendly irreducible pentanomials: $f(x) = x^m + x^{m-d} + x^{m-2d} + x^d + 1$. 

Francisco Rodríguez-Henríquez

Hardware Implementation of Pairings
## Hardware implementation of pairings: comparison Table

<table>
<thead>
<tr>
<th>Design</th>
<th>bits</th>
<th>platform</th>
<th>alg.</th>
<th>area</th>
<th>freq [MHz]</th>
<th>cycles 10^3</th>
<th>delay [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheung'11</td>
<td>126</td>
<td>Xilinx Virtex 6</td>
<td>RNS par.</td>
<td>7032 slices 32 DSPs</td>
<td>250</td>
<td>143.1</td>
<td>0.57</td>
</tr>
<tr>
<td>Fan'11</td>
<td>ate 128</td>
<td>128</td>
<td>Xilinx Virtex 6</td>
<td>HMM par.</td>
<td>4014 slices 42 DSPs</td>
<td>210</td>
<td>245.4</td>
</tr>
<tr>
<td>Estibals'10</td>
<td>Tate $\mathbb{F}_{3^{5.97}}$ 128</td>
<td>Xilinx Virtex 4</td>
<td>ternary field</td>
<td>4755 slices 7 BRAMs</td>
<td>192</td>
<td>428.9</td>
<td>2.23</td>
</tr>
<tr>
<td>Aranha’10</td>
<td>ate $\mathbb{F}_{2^{367}}$ 128</td>
<td>Xilinx Virtex 4</td>
<td>binary field</td>
<td>4518 slices</td>
<td>220</td>
<td>774.0</td>
<td>3.52</td>
</tr>
<tr>
<td>Beuchat’11</td>
<td>$\eta_T \mathbb{F}_{2^{691}}$ 105</td>
<td>Xilinx Virtex 4</td>
<td>binary field</td>
<td>78874 slices</td>
<td>130</td>
<td>2.44</td>
<td>0.02</td>
</tr>
<tr>
<td>Ghosh’11</td>
<td>$\eta_T \mathbb{F}_{2^{1223}}$ 128</td>
<td>Xilinx Virtex 6</td>
<td>binary field</td>
<td>15167 slices</td>
<td>250</td>
<td>76.0</td>
<td>0.19</td>
</tr>
<tr>
<td>Beuchat’10</td>
<td>ate 126</td>
<td>core i7 multi-core</td>
<td>Montg.</td>
<td>-</td>
<td>2800</td>
<td>2330</td>
<td>0.83</td>
</tr>
<tr>
<td>Aranha’11</td>
<td>ate 126</td>
<td>Phenom II</td>
<td>Montg.</td>
<td>-</td>
<td>3000</td>
<td>1562</td>
<td>0.52</td>
</tr>
</tbody>
</table>
Some concrete open problems

Performance Review

YOU NEED TO GET BETTER AT ANTICIPATING PROBLEMS.

IF I COULD ANTICIPATE PROBLEMS, I WOULDN'T HAVE AGREED TO WORK FOR YOU.

YOU SEEM ANGRY. I DID NOT SEE THAT COMING.
Some concrete open problems

0 To design a 128-bit security BN pairing hardware accelerator faster than the fastest software implementation
   Idea: Try to revisit the classical integer Montgomery multiplication?
Some concrete open problems

0 To design a 128-bit security BN pairing hardware accelerator faster than the fastest software implementation
   Idea: Try to revisit the classical integer Montgomery multiplication?

1 To design a 128-bit security $\eta_T$ pairing accelerator faster and with better area-time tradeoff than the one reported in
   [Ghosh-Chowdhury-Das CHES’11]
Some concrete open problems

0. To design a 128-bit security BN pairing hardware accelerator faster than the fastest software implementation.
   Idea: Try to revisit the classical integer Montgomery multiplication?

1. To design a 128-bit security $\eta_T$ pairing accelerator faster and with better area-time tradeoff than the one reported in [Ghosh-Chowdhury-Das CHES’11]

2. Have a look on what’s going on with higher genus
   [see for example the optimal pairing over supersingular genus-2 binary hyperelliptic curves eprint 2010/559]
Some concrete open problems

3 Side channel attacks on pairings.
Little has been done on this topic, it appears that at least in some cases there is a lack of good security notions and confidential targets. For example, all the parameters involved in the verification primitive of the BLS Short signature protocol are public:

$\hat{e}(D, Q) = \hat{e}(S, P)$

where $D = H(m)$ is the message digest, $S = aD$ is its signature and $P, Q = aP$ are the generator and public key of the signer, respectively. See for example [Page and Vercauteren eprint 2004/283]
Some concrete open problems

3 Side channel attacks on pairings.
Little has been done on this topic, it appears that at least in some cases there is a lack of good security notions and confidential targets. For example, all the parameters involved in the verification primitive of the BLS Short signature protocol are public:

\[ \hat{e}(D, Q) = \hat{e}(S, P), \]

where \( D = H(m) \) is the message digest, \( S = aD \) is its signature and \( P, Q = aP \) are the generator and public key of the signer, respectively. See for example [Page and Vercauteren eprint 2004/283]
Some concrete open problems

4 Better and more efficient algorithms
   Improvements in the Miller computation, Better Final exponentiation, etc. For recent improvements see for example:
Some concrete open problems

4 Better and more efficient algorithms
   Improvements in the Miller computation, Better Final exponentiation, etc. For recent improvements see for example:
   [Karabina eprint 2010/542], [Aranha-Karabina-Longa-Gebotys-López Eurocrypt 2011], [FuentesC-Knapp-RodríguezH SAC 2011],
   [Pereira-Simplício-Naehrig-Barreto JSS 2011]

5 Compute pairings within the context of protocols
   Besides pairings there exist other relevant building blocks/primitives for pairing-based cryptography: hashing to $G_1$ and $G_2$, performing scalar multiplication in $G_1$ and $G_2$, etc. See for example:
Some concrete open problems

4 Better and more efficient algorithms
   Improvements in the Miller computation, Better Final exponentiation, etc. For recent improvements see for example:
   [Karabina eprint 2010/542], [Aranha-Karabina-Longa-Gebotys-López Eurocrypt 2011], [FuentesC-Knapp-RodríguezH SAC 2011],
   [Pereira-Simplício-Naehrig-Barreto JSS 2011]

5 Compute pairings within the context of protocols
   Besides pairings there exist other relevant building blocks/primitives for pairing-based cryptography: hashing to $G_1$ and $G_2$, performing scalar multiplication in $G_1$ and $G_2$, etc. See for example:
   [Galbraith-Lin-Scott JoC 2011], [Scott eprint 2011/334],
   [FuentesC-Knapp-RodríguezH SAC 2011]
Some concrete open problems

6 Symmetric pairings Vs. Asymmetric pairings
Some concrete open problems

6 Symmetric pairings Vs. Asymmetric pairings
Factors to be considered: overall efficiency in the protocols, side-channel resistance, role in the security assumptions of the protocols.
Thank you for your attention