Bilinear Pairings

High Level Implementation - 2

CIMAT CONACYT

Luis J. Dominguez Perez

CRYPTO-CO, Julio 8 de 2016

Contenido, sección 1

Software implementations

There are already some libraries and software frameworks to efficiently compute the cryptographic pairings and the ancillary functions.

RELIC-toolkit

- RELIC is a modern cryptographic meta-toolkit with emphasis on efficiency and flexibility, and can be used to build efficient and usable cryptographic toolkits tailored for specific security levels and algorithmic choices.
- In terms of pairing-based cryptographic, RELIC implements several types of pairings and pairing-based protocols
- This is a stable, and mature library
- The main author made the OpenSSL implementation

`https://github.com/relic-toolkit/relic`

Panda

- PandA is a framework for pairing computations and arithmetic in the related groups.
- Provides the definition of an API together with tests and benchmarks.
- It follows the eBats API specification

https://www.microsoft.com/en-us/research/publication/panda-pairings-and-arithmetic/

MIRACL

- The MIRACL Crypto SDK – is a C software library, it is the gold standard open source SDK for elliptic curve cryptography (ECC). Besides PC, MIRACL enables developers to build security into highly constrained environments, including embedded, mobile apps and SCADA.

```
http://www.miracl.com/miracl-sdk
```

Sage math software

- Sage is a mathematical open-source software with GPL license. It combines the power of many open-source packages in a single Python interface.

- It tries to be a viable alternative to Magma, Maple, Mathematica, and Mathlab.

http://www.sagemath.org

Sage Installation

- The installation is easy, and there exists installers ready for OSX, Windows, Linux, and others.
- The installer is self-contained, contains any tool required for itself.

Sage Installation

- The installation is easy, and there exists installers ready for OSX, Windows, Linux, and others.
- The installer is self-contained, contains any tool required for itself.

- However, it requires too many disc space: $\sim$580 MB from the installer, and a bit more that 2.3GB on disc.
- In the case of Windows, the installer is just a Linux virtual disk with Sage already installed.

This means, you cannot use it for commercial products... but you can convert your code to pure Python/Cython

Magma

Magma is a large, well-supported software package designed for computations in algebra, number theory, algebraic geometry and algebraic combinatorics.

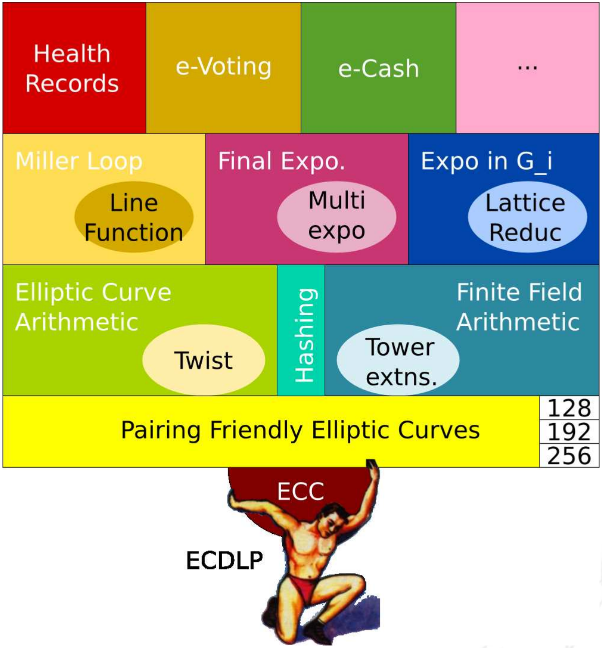
- Has support for cryptographically large numbers
- Is fairly fast enough to do some computations
- It has exclusivity of some *really hard to implement* algorithms
- It's costly unless you are in a developing country

http://magma.maths.usyd.edu.au/magma/

Plan B

- Hand-code in C/C++/ASM the finite field arithmetic
- ... and the rest of the functions (it’s fun, you should it, specially if you don’t trust third parties)

Contenido, sección 2


Health Records
e-Voting
e-Cash
...
Miller Loop
Line Function
Final Expo.
Multi expo
Expo in G_i
Lattice Reduc
Elliptic Curve Arithmetic
Twist
Hashing
Finite Field Arithmetic
Tower extns.
Pairing Friendly Elliptic Curves
128
192
256
ECC
ECDLP

Contenido, sección 3

Family of curves

The Barreto-Naehrig (BN) family of elliptic curves is ideal for implementation. These family of curves permits the use of small parameters, while achieving large security levels.

$$\begin{aligned} p(x) &= 36x^4 + 36x^3 + 24x^2 + 6x + 1; \\ r(x) &= 36x^4 + 36x^3 + 18x^2 + 6x + 1; \\ t(x) &= 6x^2 + 1. \end{aligned}$$

This family of curves has an embedding degree $k = 12$ and $\rho = \deg p(x) / \deg r(x) = 1$.

BLS curves

Barreto-Lynn-Scott, and Brezing-Weng presented a family of cyclotomic families of pairing-friendly elliptic curves, that permits the choice of the embedding degree, to generate the polynomials defining the family. The BLS family of curves with $k = 12$ is as follows:

$$\begin{aligned} p(x) &= (x^6 - 2x^5 + 2x^3 + x + 1)//3; \\ r(x) &= x^4 - x^2 + 1; \\ t(x) &= x + 1. \end{aligned}$$

This family of curves has $\rho = \deg p(x)/\deg r(x) = 1.5$.

See A taxonomy of pairing-friendly elliptic curves - Freeman, Scott, and Teske `http://eprint.iacr.org/2006/372`

Contenido, sección 4

Key size and attacks

NIST states that an 80-bit symmetric key is equivalent to a 160-bit one using discrete logs subgroups and elliptic curve groups. This is defined as a 80-bit security level, and it is not recommended for use after 2012. An 128-bit security level is recommended therefore after that year.

Equivalent symmetric key size		80	112	128	192	256
NIST	RSA	1024	2048	3072	7680	15360
	EC	160	224	256	384	512
ECRYPT	RSA	1248	2432	3248	7936	15424
	EC	160	224	256	384	512

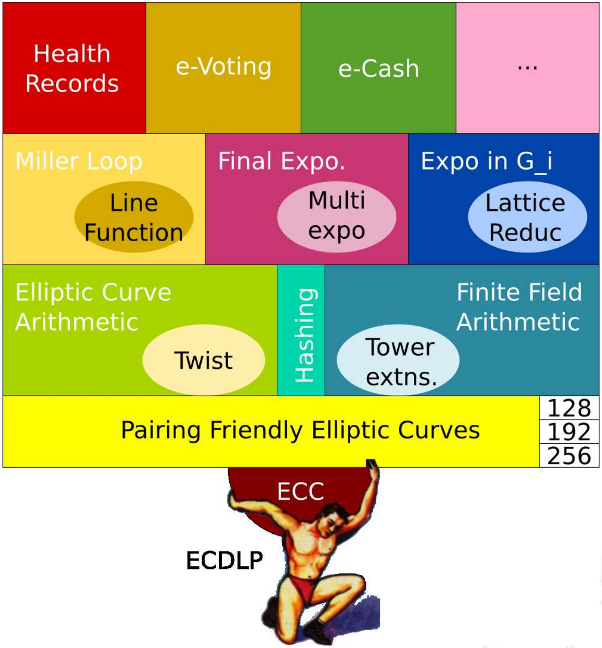
Recent Attacks

Barbulescu et al. has made public a new powerful attack
Implications:

- Key sizes for pairings has to be larger
- Alternative families of pairing friendly curves may be the new better option
- New variations on the pairing function could become interesting
- Some protocol implementation could change

However, this is work in progress

Contenido, sección 5


Health Records
e-Voting
e-Cash
...
Miller Loop
Line Function
Final Expo.
Multi expo
Expo in G_i
Lattice Reduc
Elliptic Curve Arithmetic
Twist
Hashing
Finite Field Arithmetic
Tower extns.
Pairing Friendly Elliptic Curves
128
192
256
ECC
ECDLP

Finite field arithmetic

We have a talk about efficient implementation of cryptographic primitives; however:

- Basic finite field arithmetic over $\mathbb{F}_p$, is the basic unit of operations; hence, improving this part gives acceleration to all of the protocol.

Finite field arithmetic - 2

Things to consider:

- Processor word size (integers): 8-bits to 64-bits number
- Number of CPU cores
- Arithmetic extensions available, and the number of units available
- Many core versus multi-core vs single-core
- Space to precompute values
- Desired security level

Multiprecision arithmetic

Depending on the computer, the internal word size varies. We need to:

- Hand-code our program for specific number size
- Let a library to adjust once, offline, the number size to use
- Use a software package that uses an infinite precision computation

Number representation

For a low-level implementation the integer representation uses the *small* words used by the processor:

- Let W be our target's processor word size, and let $\alpha \in \mathbb{F}_p$, with $|\alpha| \sim |p|$, and $\alpha_i \leq W$:

$$\alpha = (\alpha_0, \ldots, \alpha_n)$$

where $\alpha = \alpha_0 W^0 + \alpha_1 W^1 + \ldots + \alpha_{n-1} W^{n-1}$, with $n = \lceil (\lfloor \mathrm{Log}_2(p) \rfloor + 1)/W \rceil$, and $|\alpha_i| < W$.

Such a number could be implemented with using a computer array, as a C struct including the number of WORDS.

References

These are the two main sources for finite field arithmetic implementation for Pairings:

- High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves - Beuchat, Gonzalez-Diaz, Mitsunari, Okamoto, Rodriguez-Henriquez, and Teruya `http://eprint.iacr.org/2010/354`
- Faster Explicit Formulas for Computing Pairings over Ordinary Curves - Aranha, Karabina, Longa, Gebotys, and López `http://eprint.iacr.org/2010/526`

Basic operations

Once we have defined how to implement the basic arithmetic, we need to implement the basic operations:

- Addition
- Subtraction
- Multiplication
- Division (a.k.a. multiplication by the inverse)

Basic operations - 2

- Addition, and subtraction can be implemented component by component.
- For the multiplication we use the Montgomery multiplication
- Modular division is multiplication by the multiplicative inverse.

For the square root, we refer to Gora Adj work:
`https://eprint.iacr.org/2012/685`

Towers

- An element $\alpha \in \mathbb{F}_{p^k}$ can be represented as a polynomial up to degree $k-1$ with coefficients in $\mathbb{F}_p$ modulo an irreducible polynomial $\in \mathbb{F}_p[X]$.
- For efficiency purposes, this irreducible polynomial should be simple.

See Constructing Tower Extensions for the implementation of Pairing-Based Cryptography - Benger, and Scott.
`https://eprint.iacr.org/2009/556`

Construction

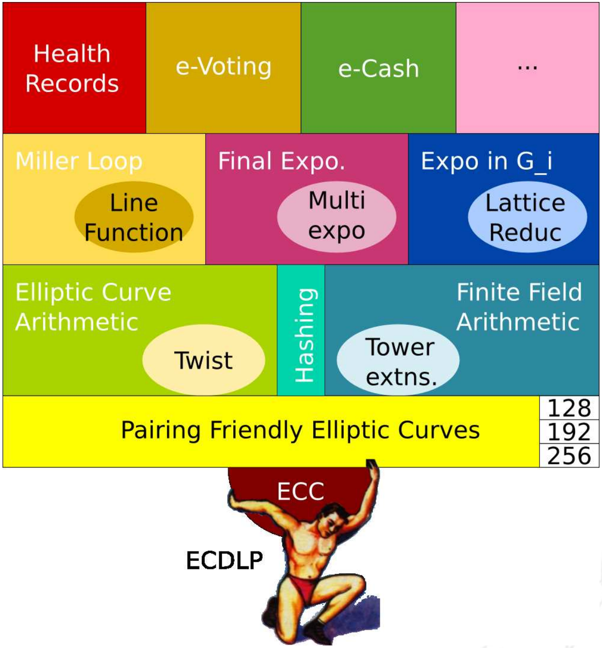
- These fields are constructed using a tower of sub-extensions using binomials as irreducible polynomials.

k	Construction	Tower
8	KSS	1-2-4-8
12	6.8	1-2-4-12
18	6.12	1-3-6-18
24	6.6	1-2-4-8-24
36	6.14	1-2-6-12-36

Other operations in the Tower

- In certain elliptic curve families there are special optimizations
- In the cyclotomic subgroups, the squaring of a number in this field extension could be dramatically improved, some optimizations depends on the x-parameter defining the p-prime in $\mathbb{F}_p$, where others just rely on the tower construction choice.

Contenido, sección 6


Health Records
e-Voting
e-Cash
...
Miller Loop
Line Function
Final Expo.
Multi expo
Expo in G_i
Lattice Reduc
Elliptic Curve Arithmetic
Twist
Hashing
Finite Field Arithmetic
Tower extns.
Pairing Friendly Elliptic Curves
128
192
256
ECC
ECDLP

EC arithmetic

To optimize the arithmetic of the elliptic curve, we usually

- change the point representation from affine coordinates to projective coordinates

Sometimes the basic version of the algorithms expect points in projective coordinates, or with mixed representation.

Twist of a curve

Definition

A twist curve E' defined over $\mathbb{F}_{p^e}$, with $e = \frac{k}{d}$, is another elliptic curve isomorphic to E defined over $\mathbb{F}_{p^k}$.

Form of the twisted curve

The formulae for an elliptic curve E is $y^2 = x^3 + a.x + b$, defined over $\mathbb{F}_p$, whereas the formulae for the twisted curve E' depends on the degree of the twist, and are as follows:

$$E' : y^2 = x^3 + \frac{ax}{D^2} + \frac{b}{D^3}b \qquad \text{for } d = 2$$

$$E' : y^2 = x^3 + \frac{ax}{D} \qquad \text{for } d = 4$$

$$E' : y^2 = x^3 + \frac{b}{D} \qquad \text{for } d = 6$$

where $D \in \mathbb{F}_{p^e}$ such that $W^d - D$ is irreducible over $\mathbb{F}_{p^e}[W]$.

Hashing to E

To hash into ordinary elliptic curves, you use two methods:

- Non-determinic method: try-and-increment
- Deterministic method: Indifferentiable Deterministic Hashing to Elliptic and Hyperelliptic Curves - Farashahi, Fouque, Shparlinski, Tibouchi, and Voloch `http://eprint.iacr.org/2010/539`

The deterministic method can be easily extended to $\mathbb{G}_2$.

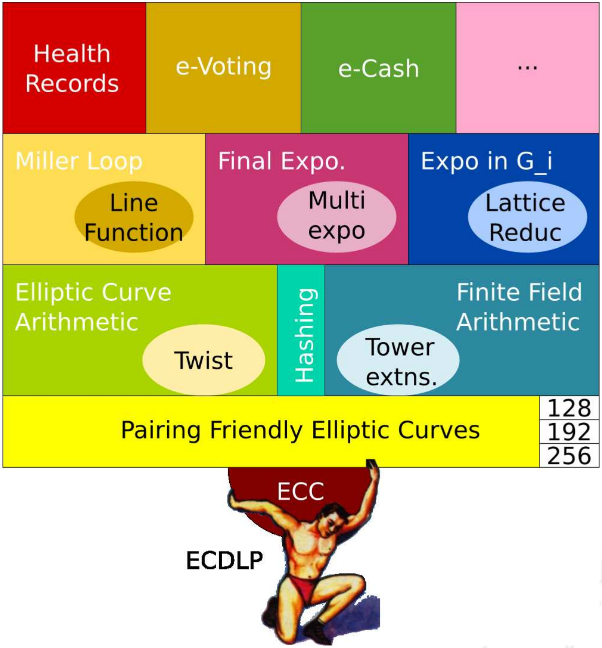
Hashing to Pairing groups

Once we have a point in the curve, then we need to map this point to the corresponding group.

- Hashing to $\mathbb{G}_1$ is cheap, we get the cofactor of the group
- Hashing to $\mathbb{G}_2$ is more elaborate since the field is substantially larger than the subgroup size

Faster Hashing to $\mathbb{G}_2$ - Fuentes-Castañeda, Knapp, Rodríguez-Henríquez. `http://cacr.uwaterloo.ca/techreports/2011/cacr2011-26.pdf`

Contenido, sección 7


Health Records
e-Voting
e-Cash
...
Miller Loop
Line Function
Final Expo.
Multi expo
Expo in G_i
Lattice Reduc
Elliptic Curve Arithmetic
Twist
Hashing
Finite Field Arithmetic
Tower extns.
Pairing Friendly Elliptic Curves
128
192
256
ECC
ECDLP

The Tate pairing

Miller [1986] discovered a function to compute cryptographic pairings. The **Tate pairing** requires one application of the *Miller loop*.

Apply $\lfloor Log_2(r) \rfloor - 1$ times the double-and-add, line-and- tangent algorithm.

The Tate pairing is a map
$E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k}) \to \mathbb{F}_{p^k}^*/(\mathbb{F}_{p^k}^*)^r$ defined as:

Definition

$$e_r : (P, Q) \mapsto \langle P, Q \rangle_r = f_{r,P}(Q)^{(p^k-1)/r}$$

The Optimal Pairing

The pairing lattice introduced by Hess, is a uniquely defined monic function of low degree.

A pairing function $e(\cdot, \cdot)$ is called Optimal Pairing if it can be computed in $\log_2 r/\varphi(k) + \varepsilon(k)$ basic Miller iterations, with $\varepsilon(k) \leq \log_2 k$.

For some families of elliptic curves, the optimal pairing can be the pairing lattice, the R-ate pairing, or even the ate pairing (for families with small trace of the Frobenius, such is the case of the BLS curves).

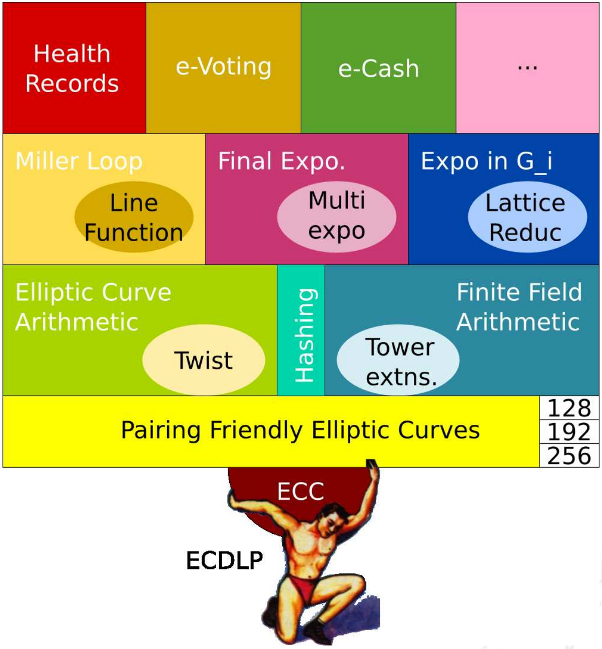
The final exponentiation

One of the most expensive operations in the pairing computation is the final exponentiation by $(p^k - 1)/r$ in the extension field $\mathbb{F}^*_{p^k}$. This is required in the computation of the Tate family of pairing functions on ordinary elliptic curves.

The method uses multi addition-chain method, and then generates the code using vector chains with the Olivos method.

Faster Hashing to $\mathbb{G}_2$ - Fuentes-Castañeda, Knapp, Rodríguez-Henríquez. `http://cacr.uwaterloo.ca/techreports/2011/cacr2011-26.pdf`

Contenido, sección 8


Health Records
e-Voting
e-Cash
...
Miller Loop
Line Function
Final Expo.
Multi expo
Expo in G_i
Lattice Reduc
Elliptic Curve Arithmetic
Twist
Hashing
Finite Field Arithmetic
Tower extns.
Pairing Friendly Elliptic Curves
128
192
256
ECC
ECDLP

Scalar-point multiplication in $\mathbb{G}_1$

Gallant, Lambert and Vanstone find out a method to do the scalar-point multiplication by breaking the scalar n into two smaller scalars, under special situations, the operation is faster.

Let E be a curve defined over the field $\mathbb{F}_p$ with zero denoted by $\mathcal{O}$.

An endomorphism of E is a rational map $\phi : E \to E$ satisfying $\phi(\mathcal{O}) = \mathcal{O}$.

There exists an endomorphism such that the following holds:

$$[n]P = [n_0]P + [n_1]\lambda P$$

where, $|n_i| \approx |\sqrt{n}|$.

Scalar-point multiplication in $\mathbb{G}_2$

Galbraith and Scott, showed a technique for generalizing the GLV method for higher powers of the endomorphism for the groups $\mathbb{G}_2$ and $\mathbb{G}_T$.

To get an m-dimensional expansion

$$n \equiv n_0 + n_1\lambda + \cdots + n_{m-1}\lambda^{m-1} \pmod{r}$$

of $[n]P$, one must decompose n with powers of λ sufficiently different and modulo r, an associated prime number to the curve.

Exponentiation in $\mathbb{G}_T$

In the case of exponentiation in $\mathbb{G}_T$, an efficiently computable endomorphism is also available.

We decompose the operation as:

$$e^n = e^{n_0} \cdot e^{n_1^p} \cdot e^{n_2^{p^2}} \cdots e^{n_1^{p^{m-1}}}$$

where $e \in \mathbb{G}_T$, $n \in \mathbb{Z}_r$, m is the degree of the decomposition, and the exponentiation to the p is done using the Frobenius endomorphism method.

See Exponentiation in pairing-friendly groups using homomorphisms - Galbrait, and Scott. `http://eprint.iacr.org/2008/117`

SAGE implementation

- For a sample SAGE implementation, visit:
  `https://bitbucket.org/luisjdominguezp/bn-sage`

You can convert this implementation to C/C++, but it needs the specific multiprecision arithmetic.

- Questions?