Pythonによる楕円曲線・ペアリング暗号ライブラリの実装とアプリケーション

緑川 志穂

who?

- 緑川 志穂　@elliptic_shiho
- 0x10 歳
- 高卒認定を取得し, 高校を辞めて日々数学とか CTF とか暗号とか
- 楕円曲線から数学を本格的に始めた

why?

- Python で使用できるペアリング実装については既にいくつか動かせるコードが存在する

→ ただしほとんどが sage[1] 依存, 又は自らの学習用途レベルでとどまっている

→ 小さい値でのみテストしているためか, 実用的な速度で動かないものが多い

[1] 数式処理系ソフトウェアや NumPy, Cython を用いて Python で数学をするためのパッケージ

why?

- さらに, ペアリングを扱える暗号処理を目的とした汎用的なライブラリは存在しなかった.

  - 独自・論文で提起された暗号の実装, 簡単に扱える暗号化ライブラリとしてなら存在した

  - それでも, ペアリングまで扱えるものは見つからなかった

- 以上から, 「楕円曲線の暗号に対して汎用的に使用可能なライブラリ」として開発を始めた

楕円曲線

- ここでは, $E : y^2 = x^3 + ax + b$ という方程式によって定義される 3 次曲線を楕円曲線とする
  - 定義体の標数が 2, 3 で無いことが条件
- 暗号では基本的に定義体が $\mathbb{F}_p$(またはその拡大体) の時を考える

楕円曲線

- なぜ楕円曲線を使うのか
  →計算や代数的構造の複雑さ
- 例えば, 点 $P = (x, y)$ があるとき, $Q = (r, s) = P + P$ は次のように計算される

$$\ell = \frac{3x^2 + a}{2y}$$
$$r = \ell^2 - 2x$$
$$s = y + \ell(r - x)$$

- 更に, $Q = nP$ という式があり, P, Q が判明している時に n を求めることが非常に難しいという性質がある (楕円曲線離散対数問題 (ECDLP))
  - 現在楕円暗号と呼ばれている暗号のほとんどはこれを安全性の根拠としており, 「攻撃」と呼ばれるものは基本的に全て ECDLP を解くアルゴリズムになる

ペアリング

- ペアリングについても少しだけ
- 楕円曲線の n ねじれ点とは, $nP = \mathcal{O}$ を満たすような P のこと
- 楕円曲線 E の n ねじれ点全体の集合を $E[n]$ と表す
- 更に, $\bmod p$ 上の 1 の m 乗根全体の集合を $\mu_m \subset \overline{\mathbb{F}_p}$ とする
  - $\overline{\mathbb{F}_p}$ は $\mathbb{F}_p$ の代数的閉包

ペアリング

- この時, ペアリングを $e_n : E[n] \times E[n] \to \mu_n$ のような写像とする.

  - 言い換えれば, 楕円曲線上の点のペアから対応する有限体の値への写像

- 実現する方法はいくつかあって, Weil ペアリングとか Tate ペアリングとか η ペアリングなんかがあります

  - 光成さんの世界最速実装は Optimal Ate ペアリングと呼ばれるものです

- このライブラリでは Weil ペアリングと Tate ペアリングを実装しました.

ペアリング

- なぜペアリングなのか?

  - ペアリングには双線形性という性質があり, 非常に応用しやすいため

- P, Q, R をそれぞれ E の n ねじれ点とするとき, $e(P, Q) = g$ なら

$$
\begin{aligned}
e(P+Q, R) &= e(P, Q) + e(R, Q) \\
e(P, Q+R) &= e(P, Q) + e(P, R) \\
e(aP, Q) &= e(P, aQ) &&= e(P, Q)^a \\
e(aP, bQ) &= e(bP, aQ) &&= g^{ab}
\end{aligned}
$$

  というような性質がある

実装

- 応用例は後にして, 実装の話を...
- 数値は long 型で管理, 値ごとにクラスを作るよくある構成

  - F = FiniteField (7); F(2) + F(5) == F(0)
- Python のクラスと継承を利用した抽象化を同時に学んだ
- しかし, かなり抽象的な書き方にしたがために速度面と複雑さがひどいことに...

実装

- ところで, 先ほどのペアリングの定義では μ_n は代数的閉包だと言いました
- $\mathbb{F}_p$ が直接代数的閉包になることはまずない

  - そのため, 拡大体を考えないといけない

  - というより, やりたい目標は基本的に二次拡大体を扱っていることに気づいた

  → 2 次拡大体は 2 次元ベクトル空間になるので, 内部型として単なる整数型を扱っていたのでは表せない

  →今の実装じゃダメ

実装

- タプルを用いて数を管理する方式に変更するものの, 変更箇所の多さ・暗黙の了解から書いていた箇所が複数あり, 最後の最後までエラー・バグ続き...
- コーディング時の見通しの甘さが目立った

高速化

- バグは全て潰せたものの...
  遅い!
- 当時のログより抜粋すると

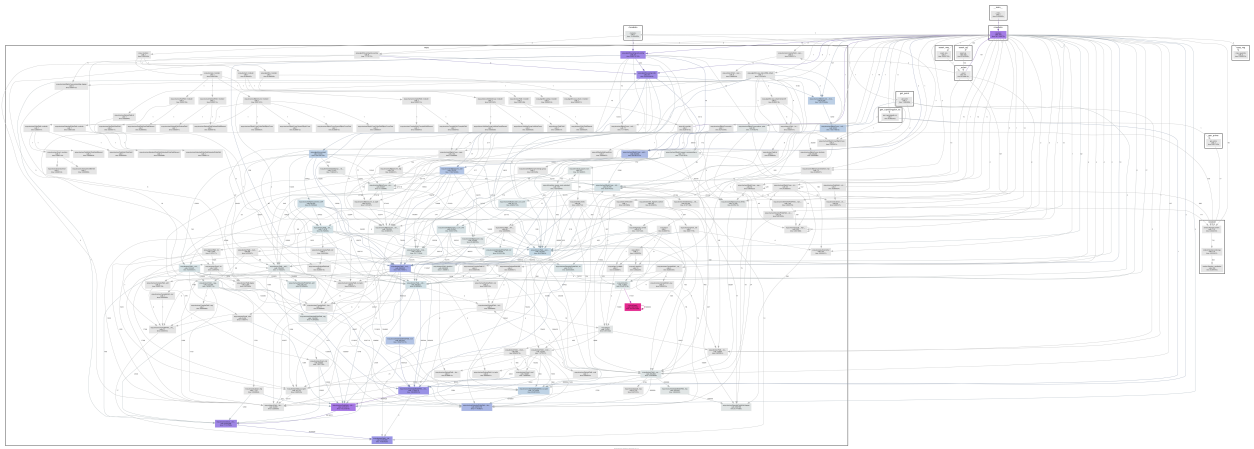
```
...
[+] 141 Test(s) finished. 141 Test(s) success, 0 Test(s
) fail.

real    0m8.717s
user    0m8.718s
sys     0m0.004s
```

高速化

- 流石になんとか高速化したい

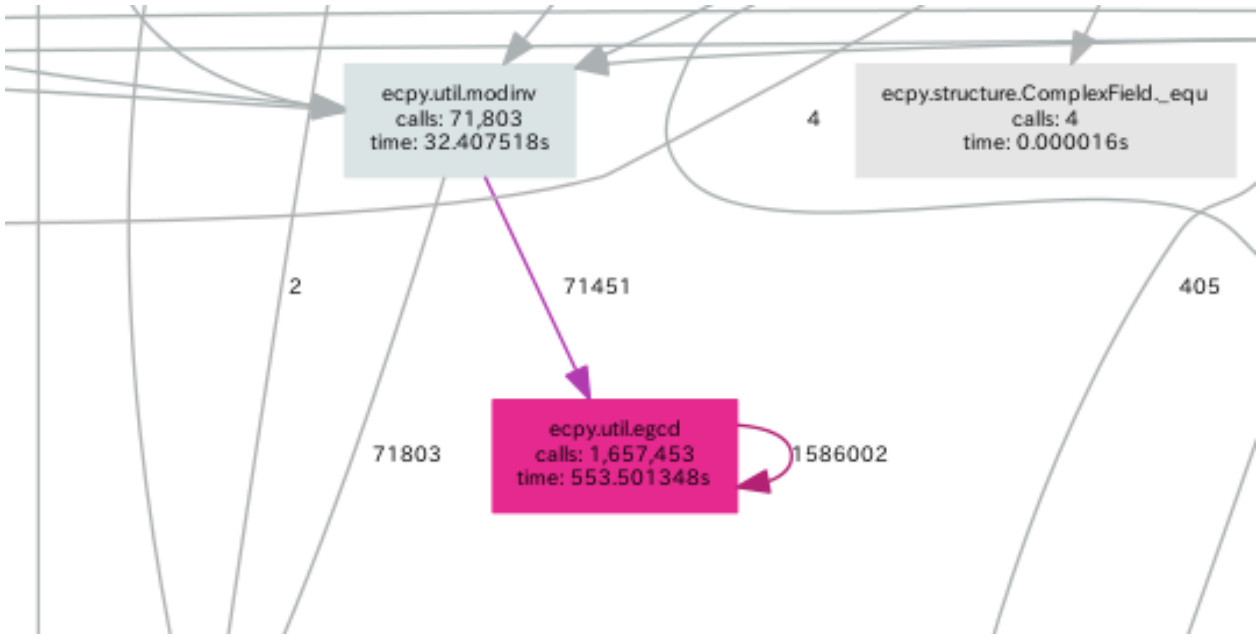
  - 調べてみると, Call graph というものを取ってみると良いらしい, ということで使ってみた

- ツールの影響ですごく時間がかかるので, ツールの挟まれた時間での計測結果



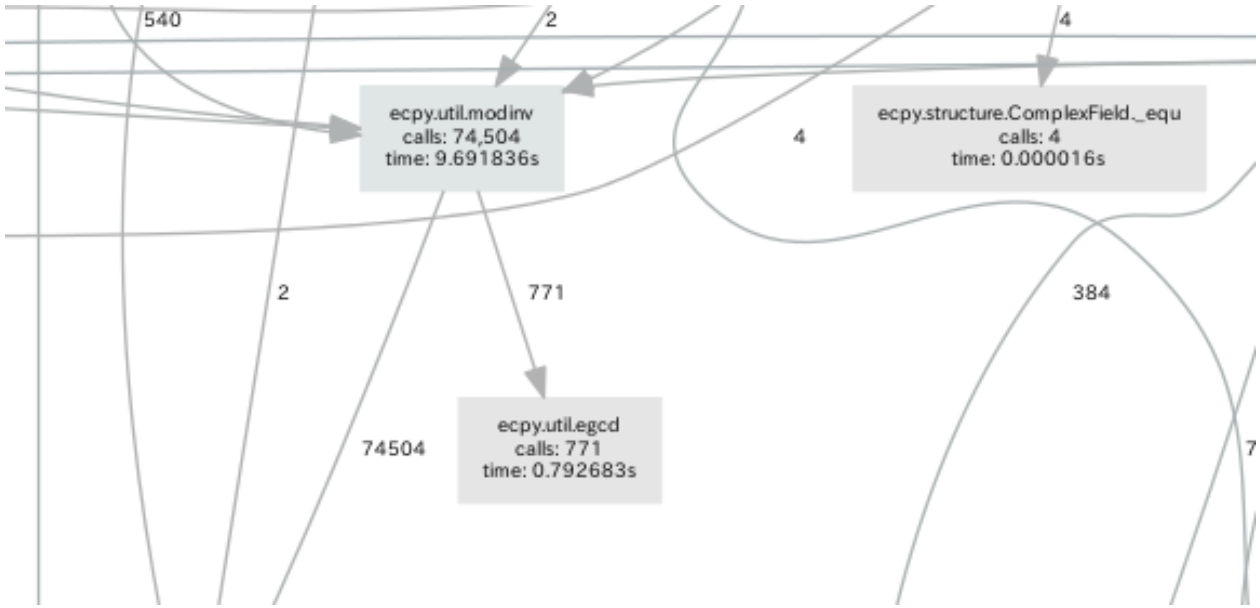
高速化

- 予想以上に複雑
  - ここ, なんか再帰すごくない?



高速化

- 拡張ユークリッドの互除法の関数
  - 逆数計算で予想以上に呼び出されている
- 最初のほうで実装したため, 簡単だった再帰を利用した方法で書いていたのが原因
- ループを用いて書き直し, ついでに Fermat の小定理の応用で呼び出し回数を減らしてみようとした結果



- しかし...

高速化

- Fermat の小定理を利用した方法は実は無意味だった
  - 計算量が関数呼び出しオーバーヘッドを上回っていた
- 代わりにメモ化を素数判定や逆数計算に入れたり, 無駄な処理を消したりしていると少し速くなった
- 教訓: Python は基本遅いしループも遅いが関数呼び出しは更に遅い
  - 関数呼び出しは内部的には Eval と同じなのでバイトコード内で完結するループよりはかなり重いのは確か
  - VM 言語の宿命

高速化

- 非常に大量に呼び出される箇所について, スーパークラスのコンストラクタの呼び出しを省くために処理をベタ書き
  - 継承の意義...
  - プログラミングのしやすさと CPU 目線の読みやすさは大抵相反する
- 関数を極力呼ばないようにコードフローを整理
- 使える所はできるだけビルトイン関数を呼ぶようにする
- line_profile を用いて簡単に行ごとのプロファイルを取って調べ, その結果からコードの書き直し
- PEP8 という Python 界全般において有名なコーディング規約を知ったのでそれに準拠するようにコードをリファクタリングしたところ予想外に速くなった
  - 綺麗なコードは速い

高速化

- Tate ペアリングでテストをした所, 次のような結果に
- 旧

```
real    0m8.717s
user    0m8.718s
sys     0m0.004s
```

- 新

```
real    0m2.823s
user    0m2.815s
sys     0m0.008s
```

- 約 3 倍?

  - time コマンドの結果なので, 正確さはそれほどかもしれませんが...

高速化

- 光成さん「ペアリングだけの計算時間は?」
  - と言われたので, そちらについても書くことに
- timeit というモジュールを使って 100 回回した平均値を取りました
- 旧

```
weil: 189763.33 usec/pass
tate: 45951.32 usec/pass
```

- 新

```
weil: 71188.95 usec/pass
tate: 15711.19 usec/pass
```

高速化

- 例として sage の結果を見てみる

```
sage: %timeit P.weil_pairing(Q, l)
10 loops, best of 3: 48.1 ms per loop
sage: %timeit P.tate_pairing(Q, l, 2)
10 loops, best of 3: 24.1 ms per loop
```

- Tate ペアリングだけならいい勝負?

  - とはいえ, sage は内部で Cython や外部プログラムを利用して演算を高速化している (Pure Python ではない) ため同等に比べられるかは疑問

高速化

- あまり関係は無いですが, PyPy(Python コードを自動で JIT して実行してくれるツール) を利用した所何もせずとも速度が半分以下に...
- Python の場合既に PyPy みたいな JIT ソリューションや Numpy や Cython 等が存在するため Pure な Python での高速化はあまり重視されないことが多い

  → 来年度も継続させて頂けるようなので, その際に C++で実装し直して更に高速化する予定です.
- 高速化は楽しくて辛い

アプリケーション開発

- ライブラリだけ作っても寂しいので, ライブラリを利用するアプリケーションを作りました.
  - ID-based Encryption
  - Boneh-Lynn-Shacham(BLS) Short Signature

ID-Based Encryption

- ID 情報が鍵の役割をする暗号で, 暗号化する人, 復号する人, 第三者機関として鍵管理センターが必要.

  - 暗号化したい人は相手の ID 情報から公開鍵を計算し, 暗号化する

  - 復号したい人は, 鍵管理センターから自分の ID に対応する秘密鍵を計算してもらい, それを用いて復号.

  - 鍵管理センターではいくつかの秘密のパラメータを保持しているため, 任意の人間の秘密鍵を計算できてしまうのが難点

- 今回は鍵管理センターに当たるものは無く, ID 情報だけで暗号化・復号ができるようにしてあります.

  - 本来はセンターに対応するサーバーを立ててやるべきですが, 準備が...

- DEMO

Boneh-Lynn-Shacham Short Signature

- 「短い署名」の通り, 今までよりも短い長さで同等の安全性を確保できるような署名
- ECDLP を根拠としており、ペアリングの性質をうまく利用したものとなっている

  - デモ版ではパラメータが 512bit のため, 短いと言えるかは怪しい...?
- 元論文では定義体を $\mathbb{F}_{3^k}$ としていたが, 今回標数 2, 3 は考えないことにしてあるため, 普通の素数を用いた実装になっている.
- 仕組み自体は簡単なのでソースコードを読んでみてください
- DEMO

Boneh-Lynn-Shacham Short Signature

準備 楕円曲線 E, 点 P, 秘密鍵 k を生成し、E, P, kP を公開する.

署名 署名したい文 m について, $k \cdot MapToPoint(m)$ を計算する.

検証 署名 sig と検証したい文 m' について, $a = \hat{e}(P, sig)$,
$b = \hat{e}(kP, MapToPoint(m'))$ を計算し, $a = b$ なら承認, $a \neq b$ なら棄却.

- ここで, $\hat{e}(\cdot, \cdot)$ は対称ペアリング $\hat{e}(\cdot, \cdot) = e(\cdot, \phi(\cdot))$

- $a = \hat{e}(P, sig) = \hat{e}(P, MapToPoint(m))^k$,
$b = \hat{e}(kP, MapToPoint(m')) = \hat{e}(P, MapToPoint(m'))^k$ なので
$m = m'$ の時に $a = b$.
- 詳しく知りたければ論文読んでください (丸投げ)

まとめ

- Python 上で動作する暗号ライブラリを開発した.

  - ペアリング, 楕円曲線特化
- 速度がそれなりに出るものを目指し, 一応 sage とも張り合えるようになった.
- 実際にライブラリを利用するアプリケーションを開発し, 速度的にも実用的になった.
- 来年度も継続し, 今回開発したライブラリを C++で書いて更に高速化を目指したい.
- 開発したものは以下のリポジトリにあります

Github: elliptic-shiho/ecpy

参考文献

- クラウドを支えるこれからの暗号技術 - 光成 滋生 著
- 暗号理論と楕円曲線 - 辻井 重男, 笠原 正雄, 有田 正剛, 境 隆一, 只木 孝太郎, 趙 晋輝, 松尾 和人 編/著
- The Arithmetic of Elliptic Curve - J.H.Silverman 著
- 楕円曲線論入門 - J.H.Silverman, J.T.Tate 著
- 整数論 1 - 雪江 明彦 著

参考文献

- Pairing for beginners - Craig Costello
- Short Signature from the Weil Pairing - Dan Boneh, Ben Lynn, Hovav Shacham
- Identity-Based Encryption from the Weil Pairing - Dan Boneh, Matthew Franklin
- Bilinear Pairings in Cryptography - Dennis Meffert
- Square root computation over even extension fields - Gora Adj, Francisco Rodriguez-Henriquez
- Pairing-Based Cryptography - Martijn Maas
- Weak Curve In Elliptic Curve Cryptography - Peter Novotney
- Adleman-Manders-Miller Root Extraction Method Revisited - Zhengjun Cao, Qian Sha, Xiao Fan
- ペアリングに関する最近の研究動向 - 岡本 栄司, 岡本 健, 金山 直樹
- 計算機代数入門 - 野呂 正行
- ペアリングの定義 - 光成 滋生
- 他多数

ありがとうございました.