Verifying Differential Privacy with EasyCrypt

Remy Wang

August 28, 2017

This note is prepared from my experience attempting to verify the differentially private minimum vertex cover algorithm in EasyCrypt. It's meant for people who work on similar mechanized formalization of differential privacy, or those who work to improve EasyCrypt. This work is supervised by Bas Spitters and supported by Claudio Orlandi at Aarhus University.

Contents

1 Getting started

1.1 Learning Differential Privacy

The best place to get up to speed on the foundations of differential privacy is the monograph [5]. It also introduces many of the algorithms that have been verified by sources below. For an informal, motivated introduction, see [4].

1.2 Understanding the algorithms

My (partial) proof of the vertex cover algorithm is mostly based on [2] (journal version). The paper contains detailed formulation of the algorithms. Better yet, they provided the mechanized proofs in CertiCrypt (the prototype of EasyCrypt in Coq). One of EasyCrypt's goals is to make those proofs more concise and intuitive.

Other sources that document the algorithms are [6] and Justin Hsu's thesis (work in progress). Note the formulation of differential privacy in [6] is however based on statistical distance instead of probabilistic coupling (see Chapters 3 & 4), and it presents the CertiCrypt system instead of EasyCrypt.

1.3 Setting up EasyCrypt

You will need a custom branch of EasyCrypt for differential privacy. At the time of writing, it was the "aprhl" branch[1] (for approximate probabilistic relational Hoare logic). Although the repo and the branch are public, they are under development and not published.

Mads Buch also provides a configured VM on his thesis website[2] if you want to skip the setup step for now and get up and running. Beware the VM includes an older version of EasyCrypt, so some up-to-date examples may not work.

If you run into problems, the helpful people on the mailing list will get you out of there. Beware only a handful people are familiar with the aprhl branch, so a good strategy is to try to get the master branch working and apply the same tricks to aprhl.

1.4 Learning EasyCrypt

Start from the EasyCrypt homepage[3] to know what it is and find links to resources. There's also installation instructions. The page links to 2 tutorial papers, one of which comes with EasyCrypt scripts. It also has slides and code from 3 summer schools for EasyCrypt. The slides assume some knowledge with Hoare Logic.

[1] EasyCrypt aprhl branch: `https://github.com/EasyCrypt/easycrypt/tree/aprhl`
[2] Mads's thesis: `http://madsbuch.com/thesis/`
[3] EasyCrypt homepage: `https://www.easycrypt.info/`

My favorite source to learn EasyCrypt is [3]. As stated above, he provides explanations of the proofs as well as a VM with immediately executable scripts. Again, the EasyCrypt installation is outdated, and so are the scripts. Updated scripts that should work on the new version are hosted at a gitlab repo[4] (you might need to ask for permission to access).

There is one more recent tutorial, which I haven't tried, found on the EasyCrypt wiki[5].

2 Proving the differentially private vertex cover algorithm

Working from [2], the following sketches my attempt to mechanize the vertex cover proof.

2.1 Infrastructural libraries

The vertex cover proof requires infrastructural lemmas about graphs. The common practice is to port from the Coq ssreflect library[6]. In this case, the relevant files are fingraph.v[7] and path.v[8]. I have ported part of path.v in this repo.

To see what the EasyCrypt ssreflect might look like, HOL Light shares a similar work flow, and people have ported part of ssreflect to HOL Light[9]. In the future, I may explore automatically porting proofs between different proof assistants and I welcome discussion.

2.2 Implementing the algorithm

We first define an abstract type `Ver` for vertices, then 2 variables: a finite set of vertices, and a relation over vertices to represent the graph. Note it may be necessary to define axioms to constrain the graph to be connected and undirected. We also define `N` to be the size of the graph, to simplify the proof

[4] aprhl example proof scripts: `https://gitlab.com/aprhl/aprhl-examples`

[5] Summer school '17: `https://www.easycrypt.info/trac/wiki/SchoolNancy2017`

[6] Coq ssreflect book: `https://math-comp.github.io/mcb/`

[7] fingraph.v: `https://github.com/math-comp/math-comp/blob/master/mathcomp/ssreflect/fingraph.v`

[8] path.v: `https://github.com/math-comp/math-comp/blob/master/mathcomp/ssreflect/path.v`

[9] HOL Light ssr: `https://github.com/flyspeck/flyspeck/tree/master/jHOLLight/Examples`

per [3] (Section 6.1.2). Otherwise, one would include `N` in the procedure as a local variable and proof it stays the same throughout the execution.

```
type Ver.

op V : Ver fset.
op E : Ver rel.
op N : {int | N = card V} as N_cardV.
```

Next, define the privacy parameters. `eps_i` is a place-holder "budget per iteration"; a complete proof needs a budget that changes every iteration.

```
op eps : {real | 0%r < eps} as eps_gt0.
op eps_i : real = (eps/(N+1)%r).
op delt : {real | 0%r < delt} as delt_gt0.
```

Define the density function, which relies on a definition of the degree of a vertex in (V, E).

```
op deg (v : Ver, vs : Ver fset, es : Ver rel) : int =
  card (filter (fun x => es x v) vs).

op omega (i : int) : real =
  (4 / eps * ( N / (N - i)) ^ -2)%r.

op df_ (x : Ver, vs : Ver fset, es : Ver rel, w : real) : real
    =
 (((deg x vs es) + w)
 / (Mrplus.sum (fun y => (deg y vs es) + w) vs))%r.

op df x (vs : Ver fset) es w = if mem vs x then df_ x vs es
    else 0%r.
```

From which we define the `pick` operator. We slightly modify the `choose` operator to be compatible with a different proof based on new sampling rules.

```
op pick (vs : Ver fset, es : Ver rel, w : real) : Ver distr =
  mk (fun v => df v vs es w).
```

A function to remove a vertex from the edge list.

```
op relrm (r : 'a rel, x : 'a) : 'a rel =
  fun (a b) => a <> x /\ b <> x /\ r a b.
```

Finally, the vertex cover procedure that calls `pick` for `n` iterations to construct a permutation of the vertices (then the vertex cover needs to be recovered from this permutation, see [2]).

```
=====================================PSEUDOCODE
n <- |V| ; pi <- [ ] ; i <- 0 ;
while i < n do
    v <$ Pick(V, E; w_i) ;
    pi <- v :: pi ;
    V <- V \ { v } ; E <- E \ ( { v } x V ) ;
    i <- i + 1
end
=====================================PSEUDOCODE

module VC = {
  proc vcover() : Ver fset = {
      var pi <- fset0;
      var v : Ver;
      var vs <- V;
      var es <- E;

      while(vs <> fset0){
        w_i <- omega (n - size vs)
        v <$ pick vs es w_i;
        pi <- (fset1 v) `&` pi;
        vs <- vs `\` fset1 v;
        es <- relrm es v;
      }
      return pi;
  }
}.
```

we also declare input graphs `g1, g2`, and vertices `t, u` between which there is an edge in `g1` but not in `g2`.

2.3 Proof sketch

We want to prove that on input graphs that share the same vertices and differ only in one edge, the algorithm preserves $(\epsilon, 0)$ differential privacy. In EasyCrypt, that reads:

```
lemma vc_dp : aequiv[[eps & 0%r] VC.vcover ~ VC.vcover :
                       V{1} = V{2} /\
                       E{2} = relU1 t u E{1}
                   ==> ={res} ].
```

First call `proc` to unfold `VC.vcover`, then use `sp` (unimplemented at the time of writing) or `seq` to consume the variable declarations / definitions up to the while loop.

From here the proof differs from the one presented in [2] which used a custom rule for the while loop. The proof proposed here also relies on sampling rules internal to EasyCrypt that was under implementation at the time of writing. The new proof is due to Justin Hsu.

We need the following sampling rules (to be built into EasyCrypt):

```
[PickSame]
{ V<1> = V<2> /\ E<1> + {(t, u)} = E<2> /\ g<1> = g<2> }

v <$ Pick(V, E; g) ~_{2 * g<1> / |V<1>|, 0} v <$ Pick(V, E; g)

{ v<1> \notin { t, u } --> v<1> = v<2> }

[PickDiff]
{ V<1> = V<2> /\ E<1> + (t, u) = E<2> /\ g<1> = g<2> }

v <$ Pick(V, E; g) ~_{log(1 + 1/g<1>), 0} v <$ Pick(V, E; g)

{ v<1> \in { t, u } --> v<1> = v<2> }

[PickId]
{ V<1> = V<2> /\ E<1> = E<2> /\ g<1> = g<2> }

v <$ Pick(V, E; g) ~_{0, 0} v <$ Pick(V, E; g)

{ v<1> = v<2> }
```

where `|V| == n - i` and `g` is some function of `n`, `i`, and `epsilon`.

Now take `pi*` to be any concrete permutation of `[n]`. We split the original loop into three pieces:

```
while i < n /\ pi*[i] \notin { t, u }
      do ... end
while i < n /\ pi*[i] \in { t, u }
            /\ { t, u } \notin pi*[0, ..., i - 1]
      do ... end
while i < n /\ pi*[i] \in { t, u }
            /\ { t, u } \in pi*[0, ..., i - 1]
      do ... end
```

We then need to apply `PickSame` in the first loop, `PickDiff` in the second loop, and `PickId` in the last loop. The invariants are:

```
 Phi_< ==
     pi<1> = pi*[0, ..., i<1>]
     --> pi<1> = pi<2> /\ V<1> = V<2> /\ E<1> + (t, u) = E<2>
         /\ \neg (i<1> < n<1> /\ pi*(i<1>) \notin { t, u })
     --> Phi_=

 Phi_= ==
     pi<1> = pi*[0, ..., i<1>]
```

```
    --> pi<1> = pi<2> /\ V<1> = V<2> /\ E<1> + (t, u) = E<2>
        /\ \neg (i<1> < n<1> /\ pi*(i<1>) \in { t, u }
        /\ { t, u } \notin pi*[0, ..., i - 1])
    --> Phi_>

Phi_> ==
    pi<1> = pi*[0, ..., i<1>]
    --> pi<1> = pi<2> /\ V<1> = V<2> /\ E<1> + (t, u) = E<2>
        /\ \neg (i<1> < n<1> /\ pi*(i<1>) \in { t, u }
        /\ { t, u } \in pi*[0, ..., i - 1])
    --> i<1> = n<1>
```

along with the normal invariants:

```
i<1> = i<2> /\ n<1> = n<2> /\ |pi<1>| = i<1> /\ |V<1>| = n<1> -
    i<1>
```

Then we need to prove `pi<1> = pi* --> pi<2> = pi*` in the postcondition at the end, and hence conclude `pi<1> = pi<2>` by pointwise equality (`pweq`) and hence differential privacy.

3 Desiderata

Completing some features / tactics of EasyCrypt would greatly simplify the task of verifying differential privacy. This section lists possible directions for improvement.

3.1 The `rnd` tactic for aprhl

Almost all proofs of differentially private algorithms in EasyCrypt so far require custom sampling rules built into the language. The Laplace algorithms in [3] calls `lap` for all random assignments, and the proof of vertex cover sketched above requires the sampling rules for `pick`. A more general approach may use the standard `rnd` extended for aprhl, and generate goals about the resulting distributions as in [2] (Section 6.4). Note that the `rand*` rule presented there would only work together with the `assert` extension to the language, because it only produces (approximate) equality as post conditions. To avoid the extension, one possibility is to generalize the `rand*` rule to produce conditionals, and include the assertion in the premise of the conditional. For example, instead of $\Phi =$ `pi<1>=pi<2>`, we have $\Phi =$ `i<1>=n<1> --> pi<1>=pi<2>`.

3.2 Other tactics

The following tactics could also be handy, though can be substituted by other existing tactics.

- `sp`: Strongest precondition, used to consume ordinary assignments in the prefix of a program. This is complement to `wp`, which is implemented for aprhl. `sp` can be simulated by `seq` with the appropriate post conditions.
- `call`: Expand procedure calls. Currently not compatible with aprhl. Not used in the vertex cover proof.
- A normalization operator: it is common to define distributions by proportion ("with probability proportional to"), e.g. the definition of the distribution $(f)^{\#}$ in section III.A. in [1]. It would be natural to have a normalization operator to reflect that in EasyCrypt.
- For loops: could be a syntax sugar for while loops. See EasyCrypt tracker[10].

3.3 Company Coq for EasyCrypt

Finally, adding support for Company Coq (auto-completion etc.) would also make the tool much more pleasant to use. See discussion on GitHub[11].

References

[1] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, and Pierre-Yves Strub. Proving differential privacy in hoare logic. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 411–424. IEEE Computer Society, 2014.

[2] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9:1–9:49, 2013.

[3] Mads Buch. Formalizing differential privacy. Master's thesis, Aarhus University, jun 2017.

[10] Feature request for for loops: `https://www.easycrypt.info/trac/ticket/17341`

[11] Feature request for company coq: `https://github.com/ProofGeneral/PG/issues/43`

[4] Vito D'Orazio, James Honaker, and Gary King. Differential privacy for social science inference. 2015.

[5] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[6] Federico Olmedo. *Approximate Relational Reasoning for Probabilistic Programs*. PhD thesis, Technical University of Madrid, Jan 2014.