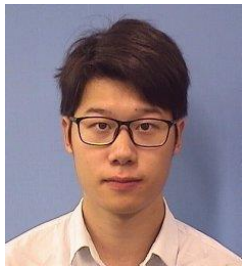
DPGen : Automated Program Synthesis for Differential Privacy

Yuxin Wang, Zeeyu Ding, Yingtai Xiao
Daniel Kifer, Danfeng Zhang

Pennsylvania State University

Reviewed by Sangjun Park in class IS661



Outline

Differential Privacy

Challenge

Solution

Evaluation

Conclusion

Differential Privacy

- Differential privacy is a mathematical technique for protecting individual privacy

Differential Privacy

- Differential privacy is a mathematical technique for protecting individual privacy
- This allows for the acquisition of useful statistical information at the group level while ensuring the protection of individual data

Differential Privacy

- Differential privacy is a mathematical technique for protecting individual privacy
- This allows for the acquisition of useful statistical information at the group level while ensuring the protection of individual data

high utility
no privacy

high privacy
no utility

Differential Privacy

- DP ensures that the database cannot be inferred.

Differential Privacy

- DP ensures that the database cannot be inferred.

Differential Privacy

- DP ensures that the database cannot be inferred.

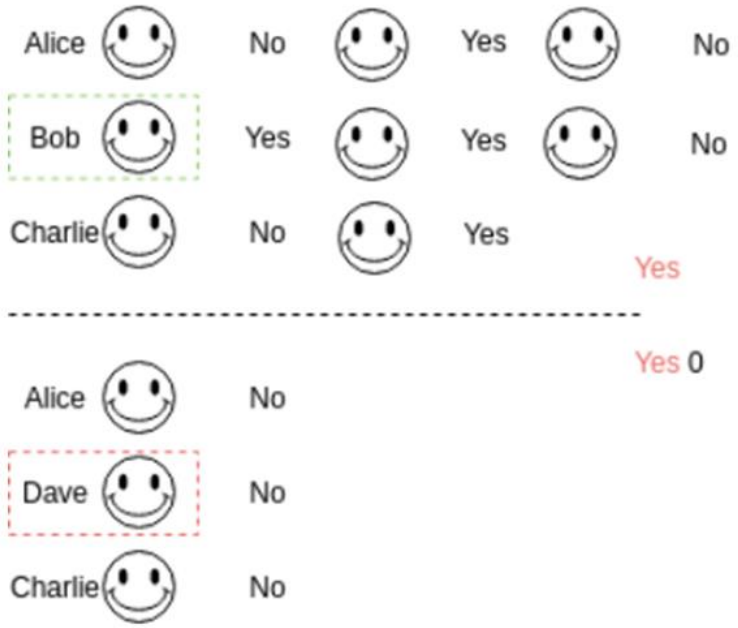
Others can infer the presence of BoB who have virus

Differential Privacy

- DP ensures that the database cannot be inferred.

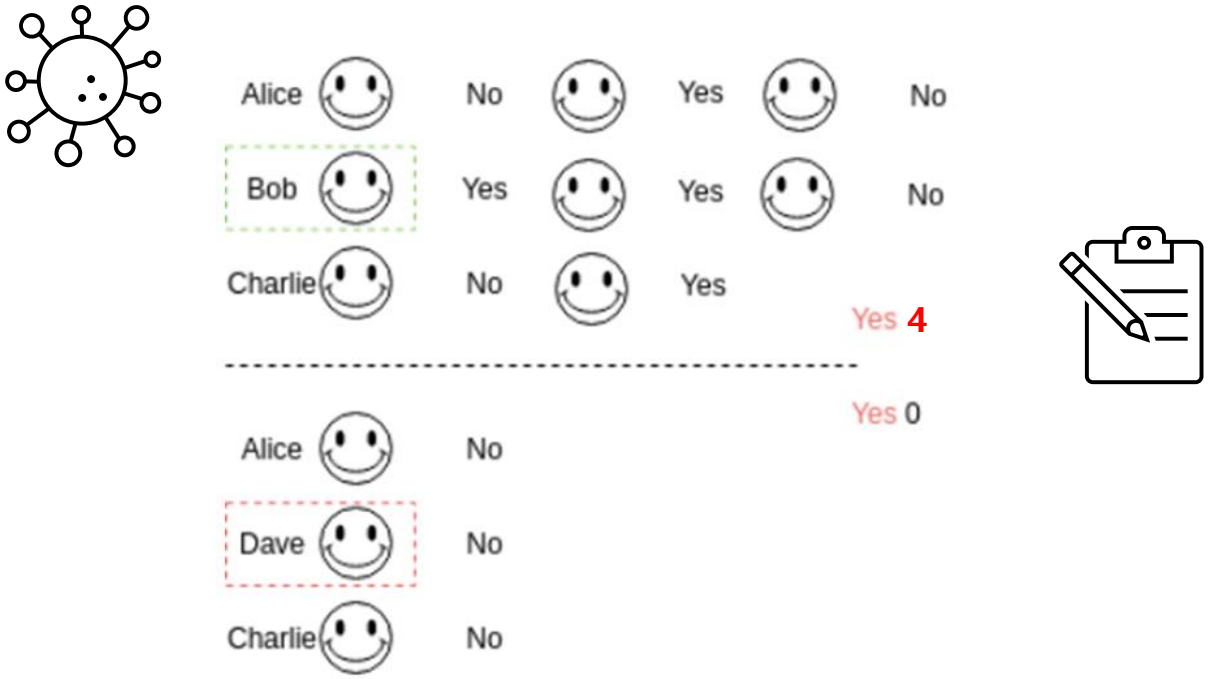
Differential Privacy

- DP ensures that the database cannot be inferred.



Differential Privacy

- DP ensures that the database cannot be inferred.

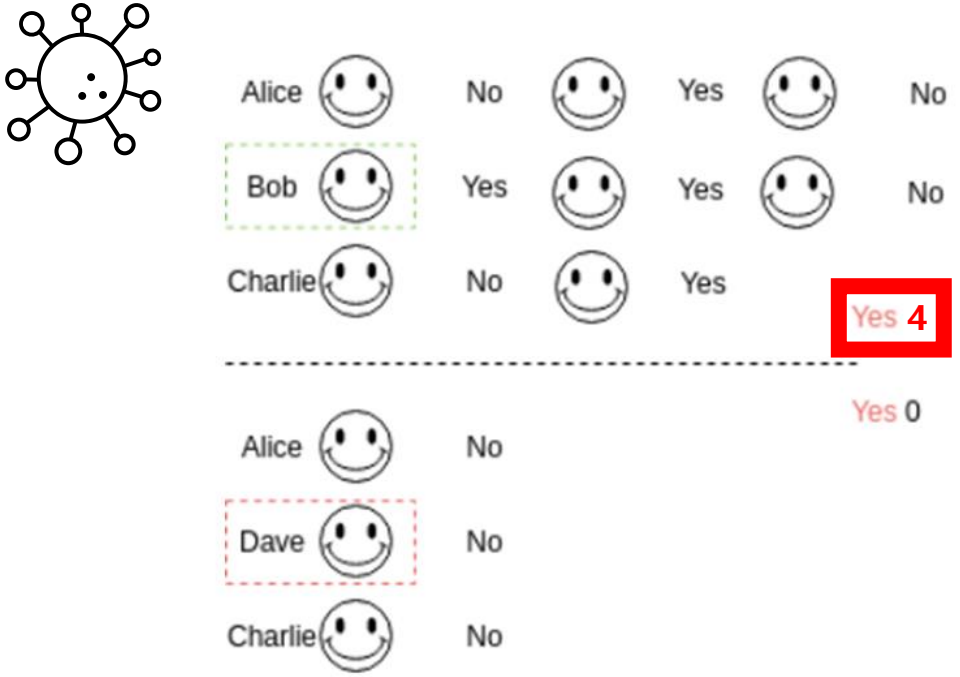


Differential Privacy

- DP ensures that the database cannot be inferred.

Differential Privacy

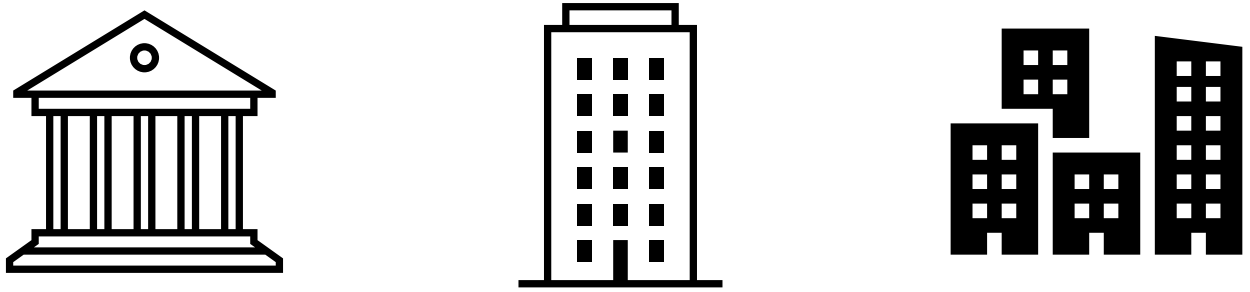
- DP ensures that the database cannot be inferred.



Others cannot infer the presence of BoB who have virus

Differential Privacy

- Widely accepted mathematical definition of privacy
- Adoted by companies and government agencies

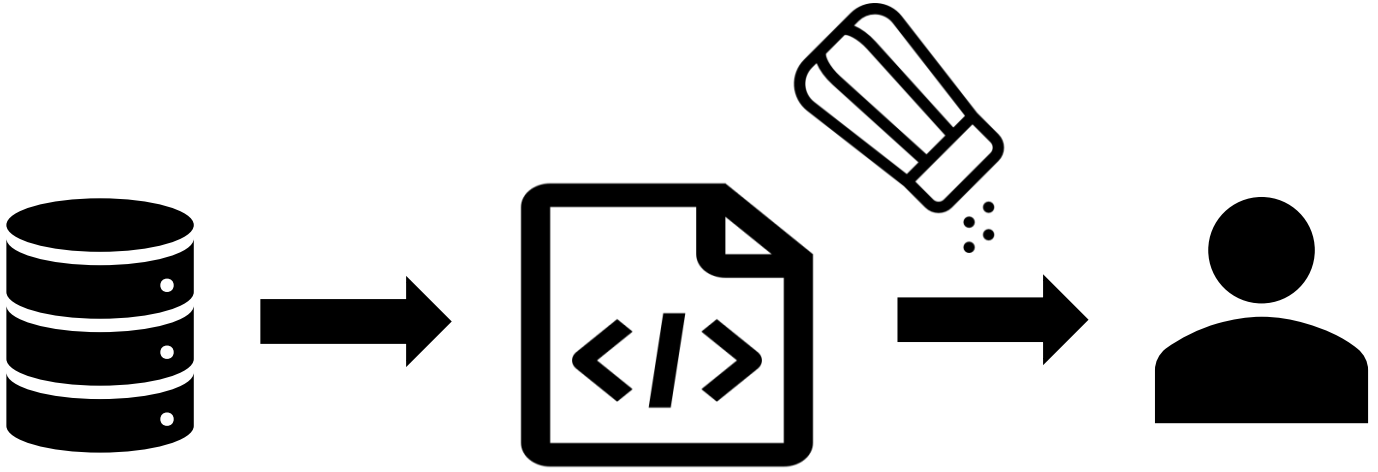


Designing DP is Hard

- Often accomplished by carefully introducing noise to the data output
- Require expert knowledge to ensure DP
- Sometimes even experts can make mistakes

Synthesize for Ensuring DP

- The noise was automatically synthesized at the source code level to ensure compliance with differential privacy
- To satisfy differential privacy, instead of adding noise directly to the database, noise was added in the code that processes the data



Challenging Example

```
def ReportMax(query, size):
        i=0, best=0, out=0

        while (i < size):
                if (query[i] > best || i == 0):
                        out = i
                        best = query[i]
                i+=1
        return out
```

Challenging Example

```
def ReportMax( query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
        i+=1
    return out
```

query : list of query answers
size : length of query

Challenging Example

```
def ReportMax( query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
        i+=1
    return out
```

query : list of query answers
size : length of query

Iterate all query to find max value element

Challenging Example

```
def ReportMax( query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
        i+=1
    return out
```

query : list of query answers
size : length of query

Iterate all query to find max value element

Return max values's index

Challenging Example

- To ensure DP, we need to add random noise in the process logic

```
def ReportMax(query, size):
    i=0, best=0, out=0
    noise = noise_generator()

    while (i < size):
        if (query[i] + noise > best || i == 0):
            out = i
            best = query[i] + noise
        i+=1
    return out
```

Challenging Example

- To ensure DP, we need to add random noise in the process logic

```
def ReportMax(query, size):
    i=0, best=0, out=0
    noise = noise_generator()

    while (i < size):
        if (query[i] + noise > best || i == 0):
            out = i
            best = query[i] + noise
        i+=1
    return out
```

- Location:
  1) which variables need noise?
  2) where should we add noise?

Challenging Example

- To ensure DP, we need to add random noise in the process logic

```
def ReportMax(query, size):
    i=0, best=0, out=0
    noise = noise_generator()

    while (i < size):
        if (query[i] + noise > best || i == 0):
            out = i
            best = query[i] + noise
        i+=1
    return out
```

- Location:
  1) which variables need noise?
  2) where should we add noise?

Challenging Example

- To ensure DP, we need to add random noise in the process logic

```
def ReportMax(query, size):
    i=0, best=0, out=0
    noise = noise_generator()

    while (i < size):
        if (query[i] + noise > best || i == 0):
            out = i
            best = query[i] + noise
        i+=1
    return out
```

- Location:
  1) which variables need noise?
  2) where should we add noise?

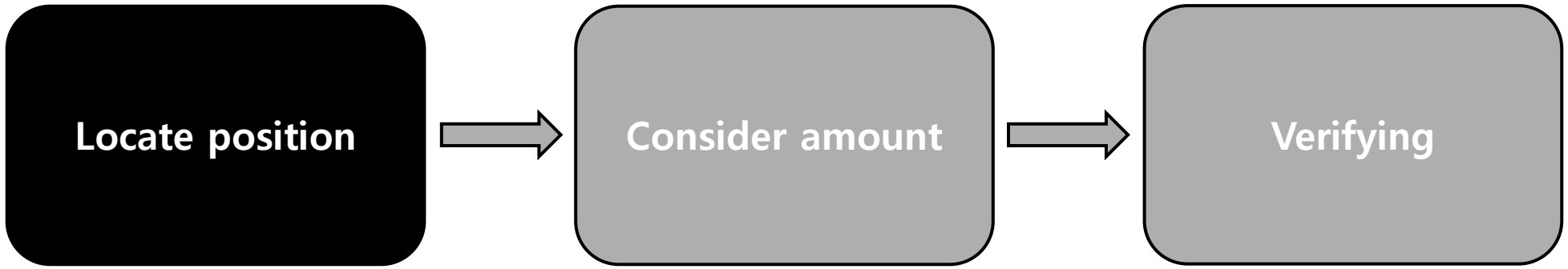
- Amount:
  How much noise should we add?

Solution Steps

- They offer 3 steps for synthesize DP program

Solution Steps

- They offer 3 steps for synthesize DP program



Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Taint the user input varable

Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Taint the user input varable

Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Taint the user input varable

Tracking the data flow of taint value

- Explicit Data flow
- Implicit Data flow of branch condition

Locate noise position

- **Which** variables need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Taint the user input varable

Tracking the data flow of taint value

- Explicit Data flow
- Implicit Data flow of branch condition

Candidate for where noise can be added

Locate noise position

- **Where** should need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Candidates variables: query[i], best

Locate noise position

- **Where** should need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Candidates variables: query[i], best

They offer two kinds of rules

Locate noise position

- **Where** should need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Candidates variables: query[i], best

They offer two kinds of rules

1. **Before Use**

Locate noise position

- **Where** should need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0

    while (i < size):
        query = query + noise
        best   = best + noise
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Candidates variables: query[i], best

They offer two kinds of rules

1. **Before Use**

Locate noise position

- **Where** should need to be noised?

```
def ReportMax(query, size):
    i=0, best=0, out=0
    best = best + noise

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

Candidates variables: query[i], best

They offer two kinds of rules

1. **Before Use**
2. **After Definition**

Solution Steps

- They offer 3 steps for synthesize DP program

Consider amount of noise

- Need to find appropriate value of noise

```
def ReportMax(query, size):
    i=0, best=0, out=0
    best = best + noise

    while (i < size):
        if (query[i] > best || i == 0):
            out = i
            best = query[i]
    i+=1
    return out
```

How much noise is added?

Consider amount of noise

- Need to find appropriate value of noise to ensure **Utility** and **Privacy**

Consider amount of noise

- They determine the amount of noise with PSO
- PSO algorithm employs multiple particles to search the solution space iteratively, in which a position is a candidate solution

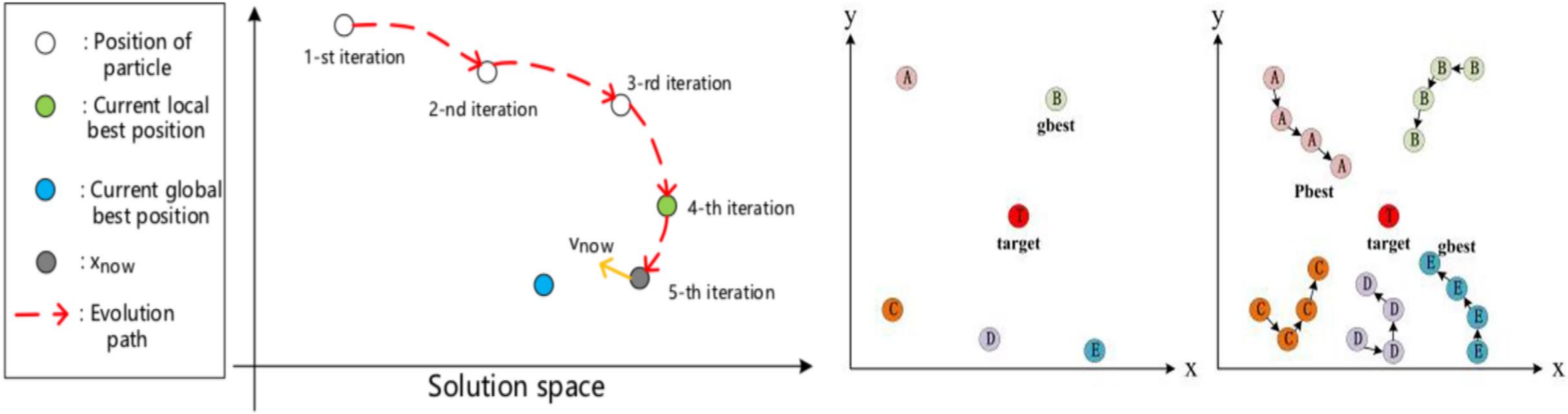
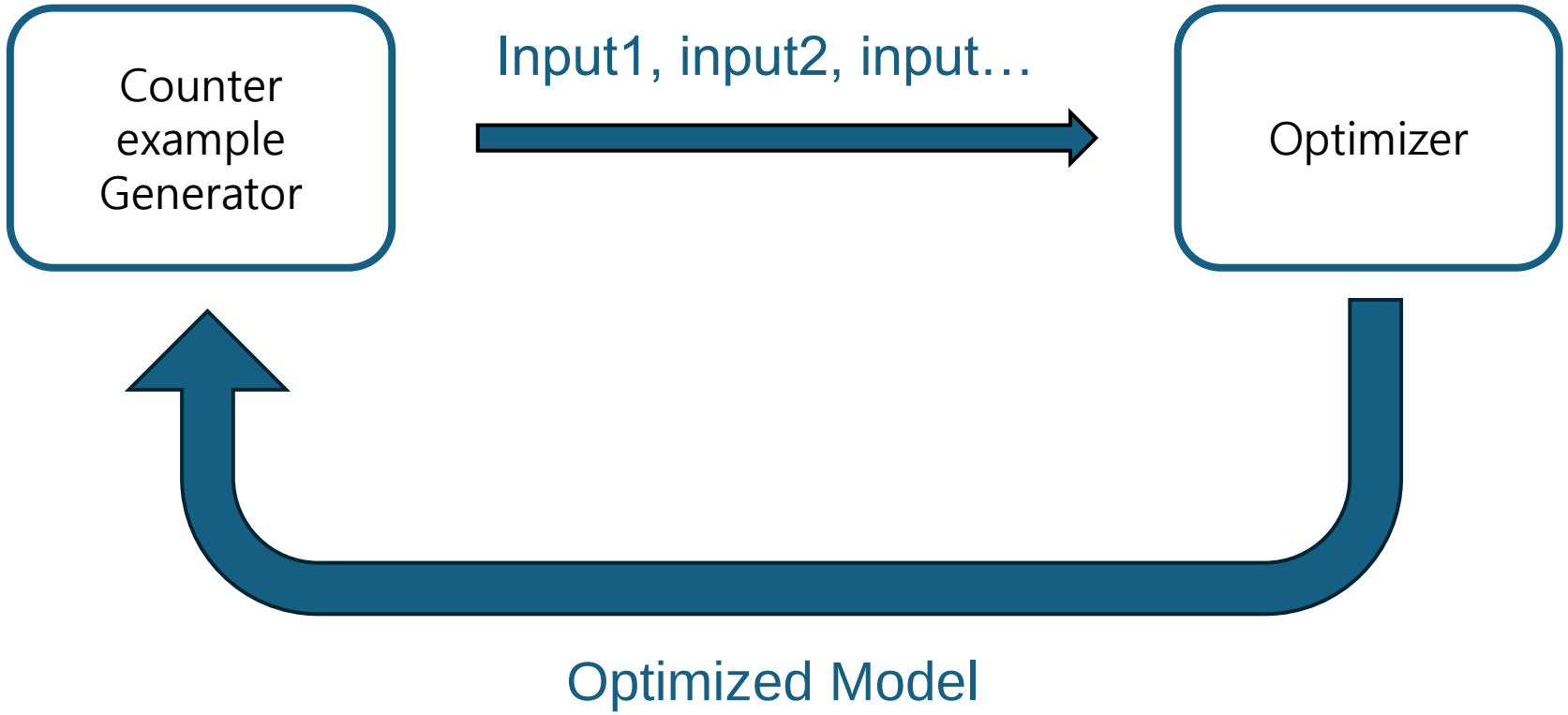


Figure from MOPT-AFL in usenix security 2017

Consider amount of noise

- Deployed Counter-example Guided Inductive Synthesis



Solution Steps

- They offer 3 steps for synthesize DP program

Verify Code

- Use CPAChecker to verify the synthesized DP code

Evaluation

- Compared with prior work (KOLAHAL), DPGen demonstrates better performance

Mechanism	DPGen	KOLAHAL
ReportNoisyMax	120s	1920s
PartialSum	10s	900s
SmartSum	25s	5460s
SVT	29s	2640s
SVT-Inverse	38s	N/A
GapSVT	25s	N/A
NumSVT	35s	N/A
SVT-WhilePriv	617s	N/A
AdaptiveSVT	3026s	N/A

Conclusion

- DPGen: An automated program synthesizer for Differential Privacy
- DPGen synthesizes code that satisfies Differential Privacy more effectively and in a faster time compared to previous research
- In my Opinion
  - I didn't know this field existed, but now that I do, it's more interesting than I thought.

Thank you