Formal Verification of Differentially Private Mechanisms

Marco Gaboardi
University at Buffalo, SUNY

Goal of formal verification: building programs that are correct.

Why correctness matters?

Why correctness matters?

An example:
DARPA HACMS (High Assurance Cyber Military Systems)

Infosec Institute

What does “correct” mean?

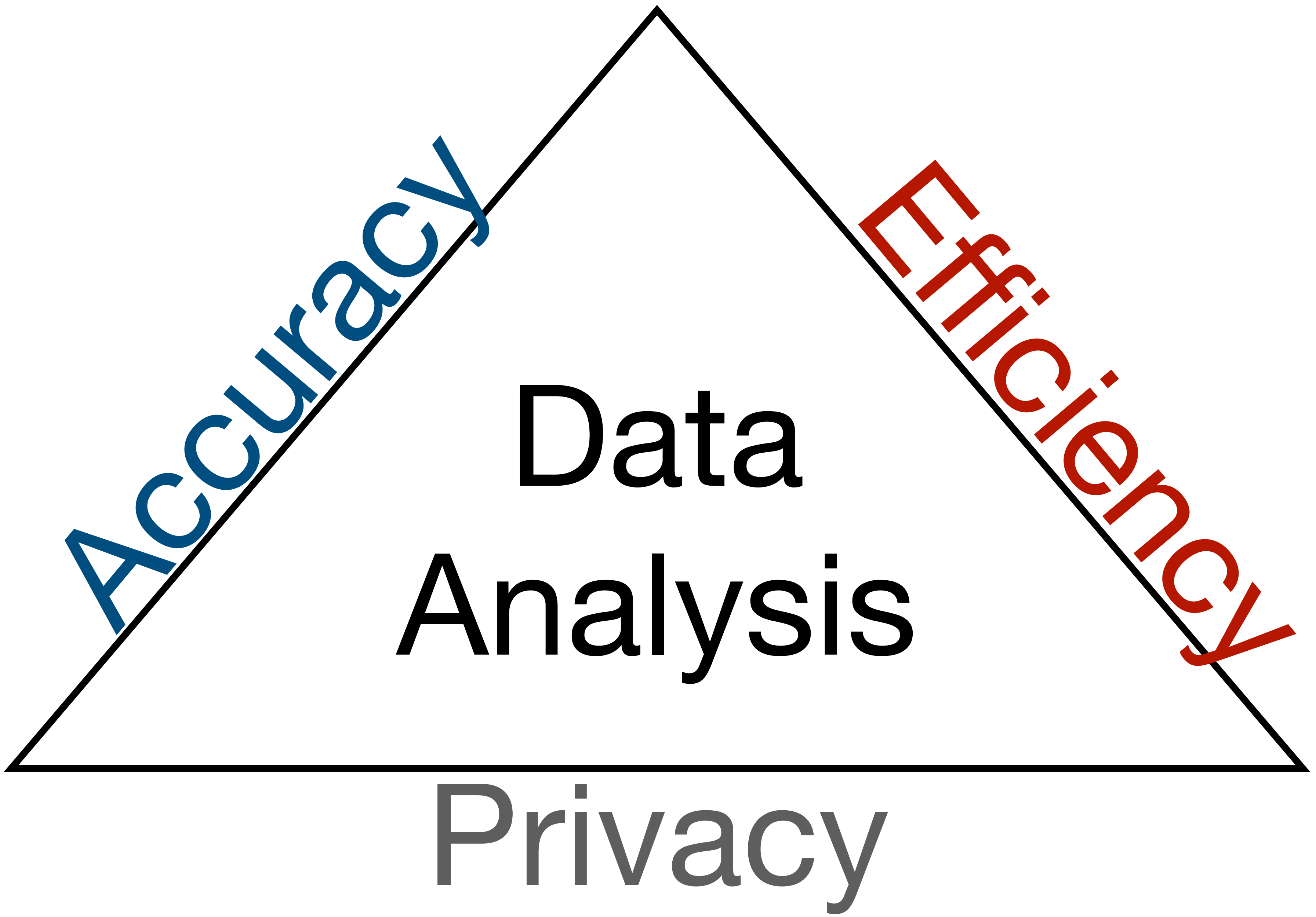
In traditional program verification, a program is correct if it respects the specification:

- What is computed (functional aspects)
- How it is computed (non-functional aspects).

What does correct mean for

differentially private applications?

Specification



Abstract?

or

Concrete?

Desiderata: building private, accurate, and efficient implementations that are secure and resilient to attacks.

Byproduct

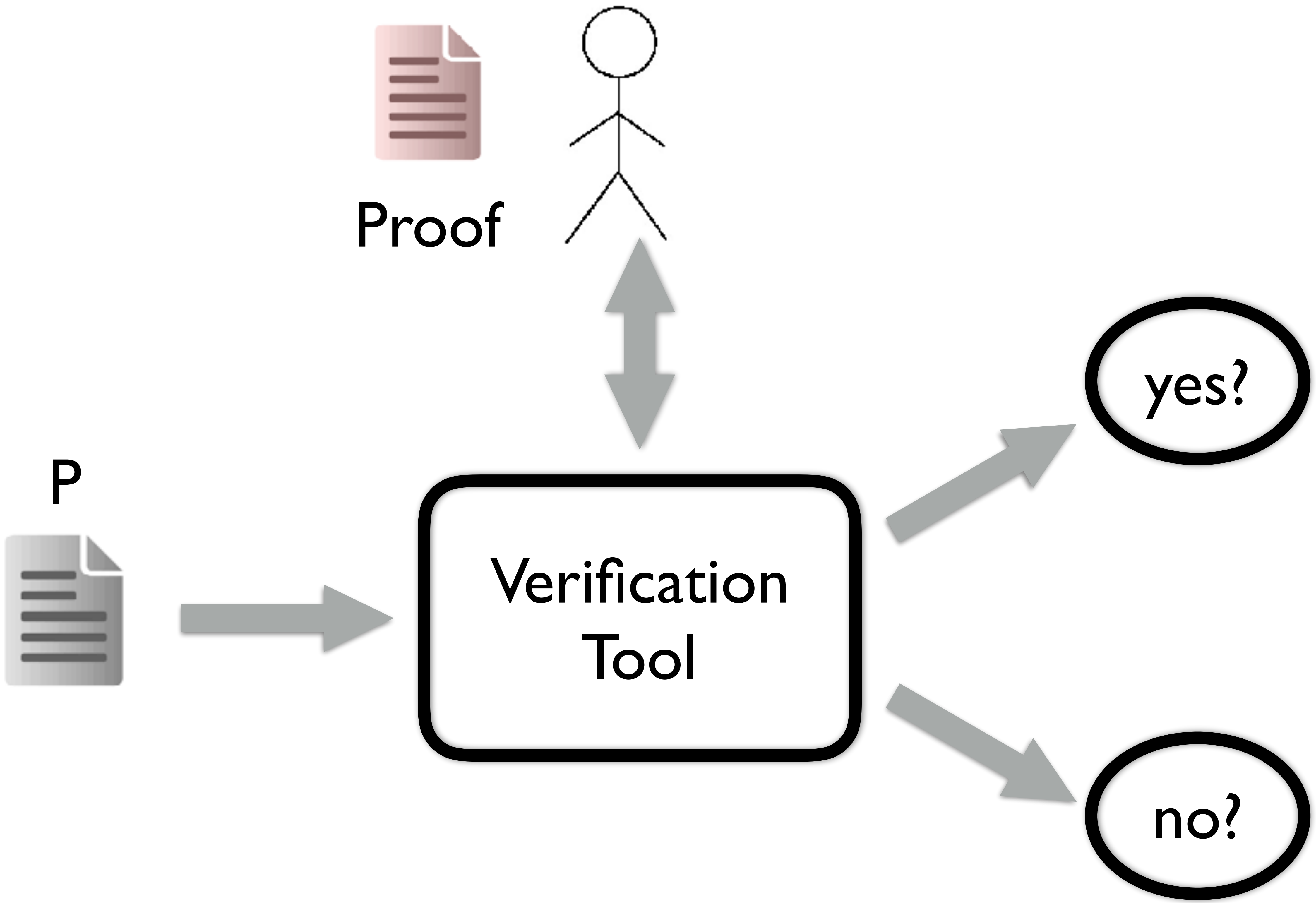
Systems that can help with the design of differentially private data analysis.

Outline

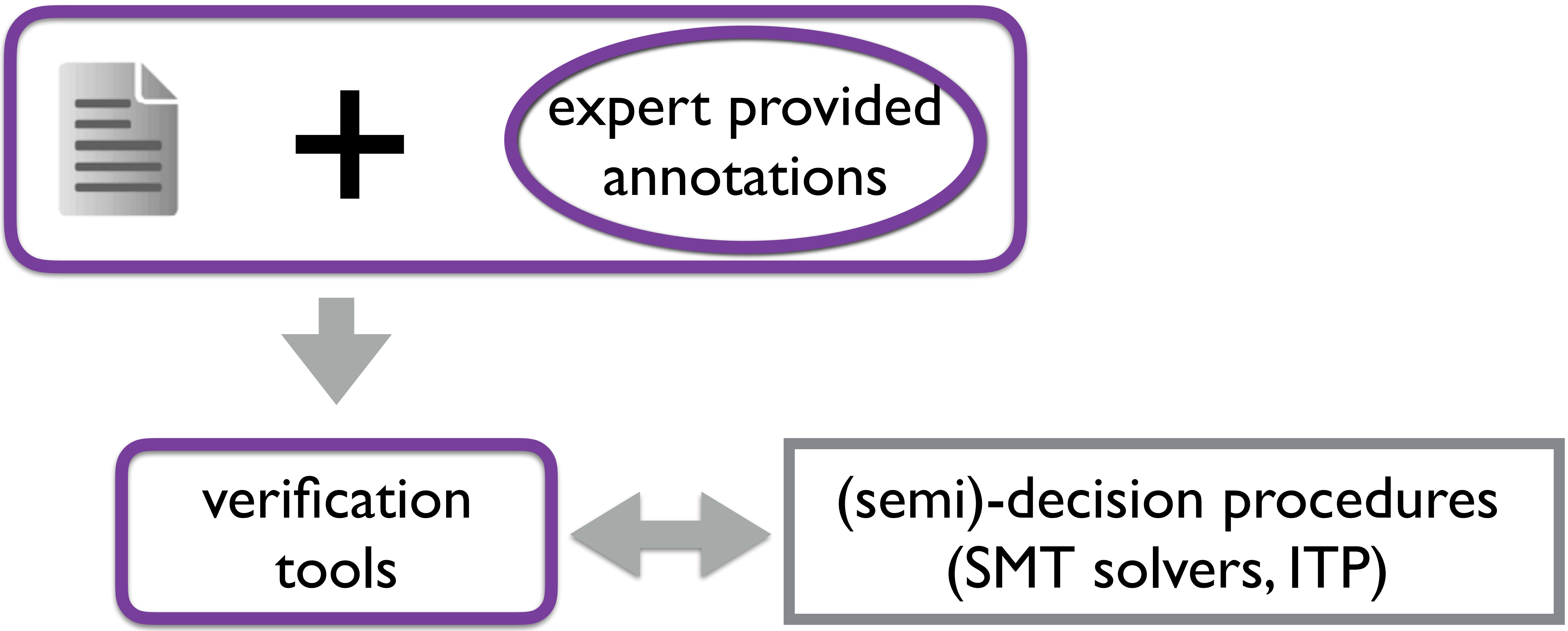
- Few words on program verification,
- Challenges in the verification of differential privacy,
- Verification methods developed so far,
- Looking forward.

A 10 thousand ft view on program verification…

Proofs vs Formal Proofs



Verification tools



An example

Consider a simple program squaring a given number m:

```
   {{ X = m }}
Y ::= 0;;
Z ::= 0;;
WHILE  Y ≠ X  DO
   Z ::= Z + X;;
   Y ::= Y + 1
END
   {{ Z = m*m }}
```

An example

A proof of correctness can be given as follows:

```
  {{ X = m }} ->>
  {{ 0 = 0*m ∧ X = m }}
Y ::= 0;;
  {{ 0 = Y*m ∧ X = m }}
Z ::= 0;;
  {{ Z = Y*m ∧ X = m }}
WHILE Y ≠ X DO
    {{ Z = Y*m ∧ X = m ∧ Y ≠ X }} ->>
    {{ Z+X = (Y+1)*m ∧ X = m }}
  Z ::= Z + X;
    {{ Z = (Y+1)*m ∧ X = m }}
  Y ::= Y + 1
    {{ Z = Y*m ∧ X = m }}
END
  {{ Z = Y*m ∧ X = m ∧ ~(Y ≠ X) }} ->>
  {{ Z = m*m }}
```

A lot of techniques to make this approach automated

Questions that program verification can help with

- Are our algorithms bug-free?
- Do implementations respect the algorithms?
- Is the system architecture bug-free?
- Is the code efficient?
- Is the actual machine code correct?
- Do the optimization preserve correctness?
- Is the full stack attack-resistant?

Some successful stories - I

- CompCert - a fully verified C compiler,
- Sel4, CertiKOS - formal verification of OS kernel
- A formal proof of the Odd order theorem,
- A formal proof of Kepler conjecture.

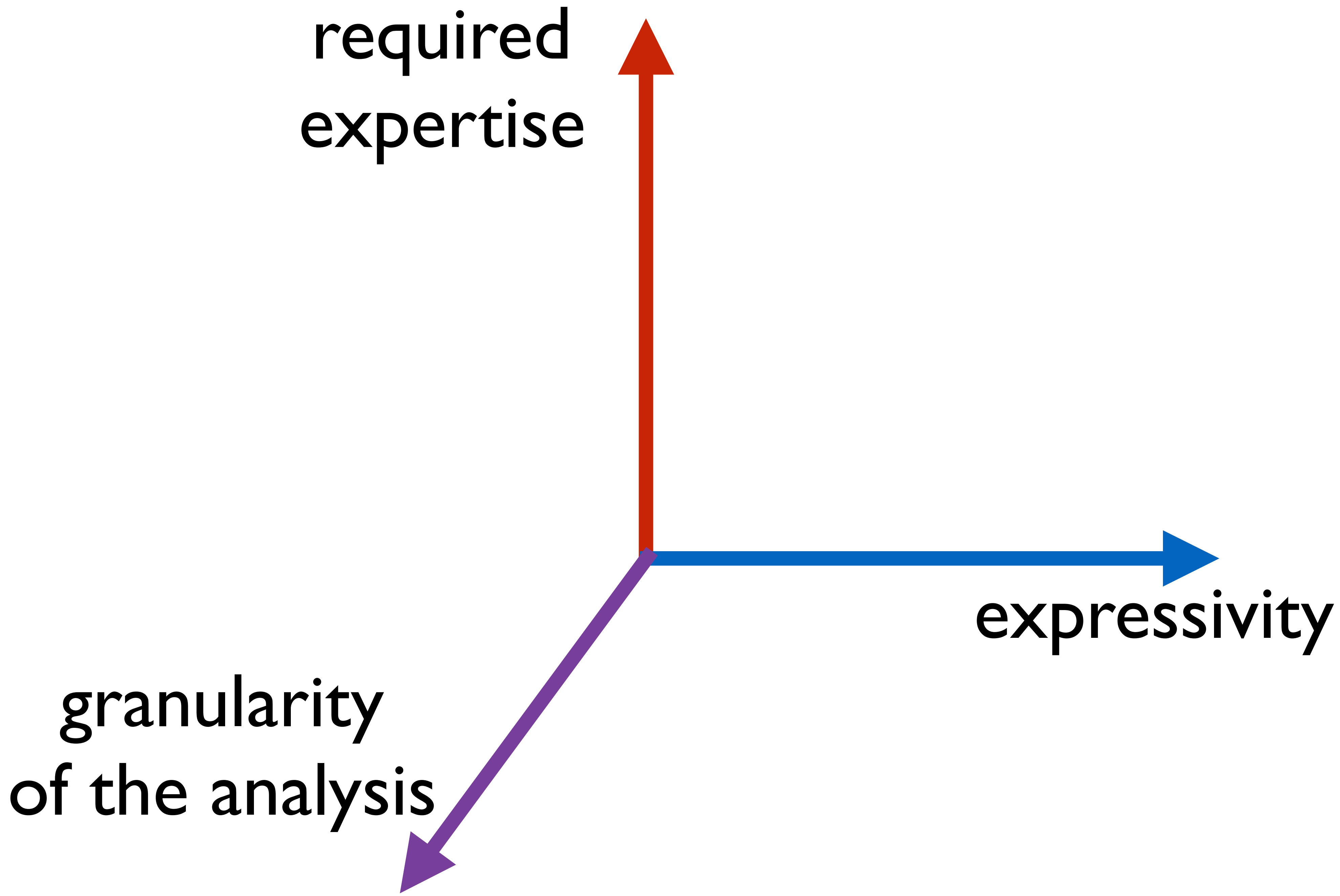
Years of work from very specialized researchers!

Some successful stories - II

- Automated verification for Integrated Circuit Design.
- Automated verification for Floating point computations,
- Automated verification of Boeing flight control - Astree,
- Automated verification of Facebook code - Infer.

The years of work go in the design of the techniques!

Verification trade-offs



How things can go wrong in Differential Privacy….

The challenges of differential privacy

Given $\varepsilon,\delta \geq 0$, a mechanism $M: db \to O$ is (ε,δ)-differentially private iff $\forall b_1, b_2 :db$ differing in one record and $\forall S \subseteq O$:

$$\Pr[M(b_1) \in S] \leq \exp(\varepsilon) \cdot \Pr[M(b_2) \in S] + \delta$$

- Relational reasoning,
- Probabilistic reasoning,
- Quantitative reasoning

Example 1: the sparse vector case

Algorithm 1 An instantiation of the SVT proposed in this paper.

Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots, c.$

$\epsilon_1 = \epsilon/2, \quad \rho = \text{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \text{Lap}(2c\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T_i + \rho$ **then**
    Output $a_i = \top$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \bot$

Algorithm 2 SVT in Dwork and Roth 2014 [8].

Input: $D, Q, \Delta, T, c.$

$\epsilon_1 = \epsilon/2, \quad \rho = \text{Lap}(c\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \text{Lap}(2c\Delta/\epsilon_1)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top, \rho = \text{Lap}(c\Delta/\epsilon_2)$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \bot$

Algorithm 3 SVT in Roth's 2011 Lecture Notes [15].

Input: $D, Q, \Delta, T, c.$

$\epsilon_1 = \epsilon/2, \quad \rho = \text{Lap}(\Delta/\epsilon_1),$
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \text{Lap}(c\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = q_i(D) + \nu_i$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \bot$

Algorithm 4 SVT in Lee and Clifton 2014 [13].

Input: $D, Q, \Delta, T, c.$

$\epsilon_1 = \epsilon/4, \quad \rho = \text{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \text{Lap}(\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \bot$

Algorithm 5 SVT in Stoddard et al. 2014 [18].

Input: $D, Q, \Delta, T.$

$\epsilon_1 = \epsilon/2, \quad \rho = \text{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ **do**
  $\nu_i = 0$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top$
  else
    Output $a_i = \bot$

Algorithm 6 SVT in Chen et al. 2015 [1].

Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots.$

$\epsilon_1 = \epsilon/2, \quad \rho = \text{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ **do**
  $\nu_i = \text{Lap}(\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T_i + \rho$ **then**
    Output $a_i = \top$
  else
    Output $a_i = \bot$

Min Lyu, Dong Su, Ninghui Li:
Understanding the Sparse Vector Technique for Differential Privacy. PVLDB (2017)

Example 2: the rounding case


On Significance of the Least Significant Bits For Differential Privacy

Ilya Mironov

Abstract

We describe a new type of vulnerability present in many implementations of differentially private mechanisms. In particular, all four publicly available general purpose systems for differentially private computations are susceptible to our attack.

The vulnerability is based on irregularities of floating-point implementations of the privacy-preserving Laplacian mechanism. Unlike its mathematical abstraction, the textbook sampling procedure results in a porous distribution over double-precision numbers that allows one to breach differential privacy with just a few queries into the mechanism.

We propose a mitigating strategy and prove that it satisfies differential privacy under some mild assumptions on available implementation of floating-point arithmetic.

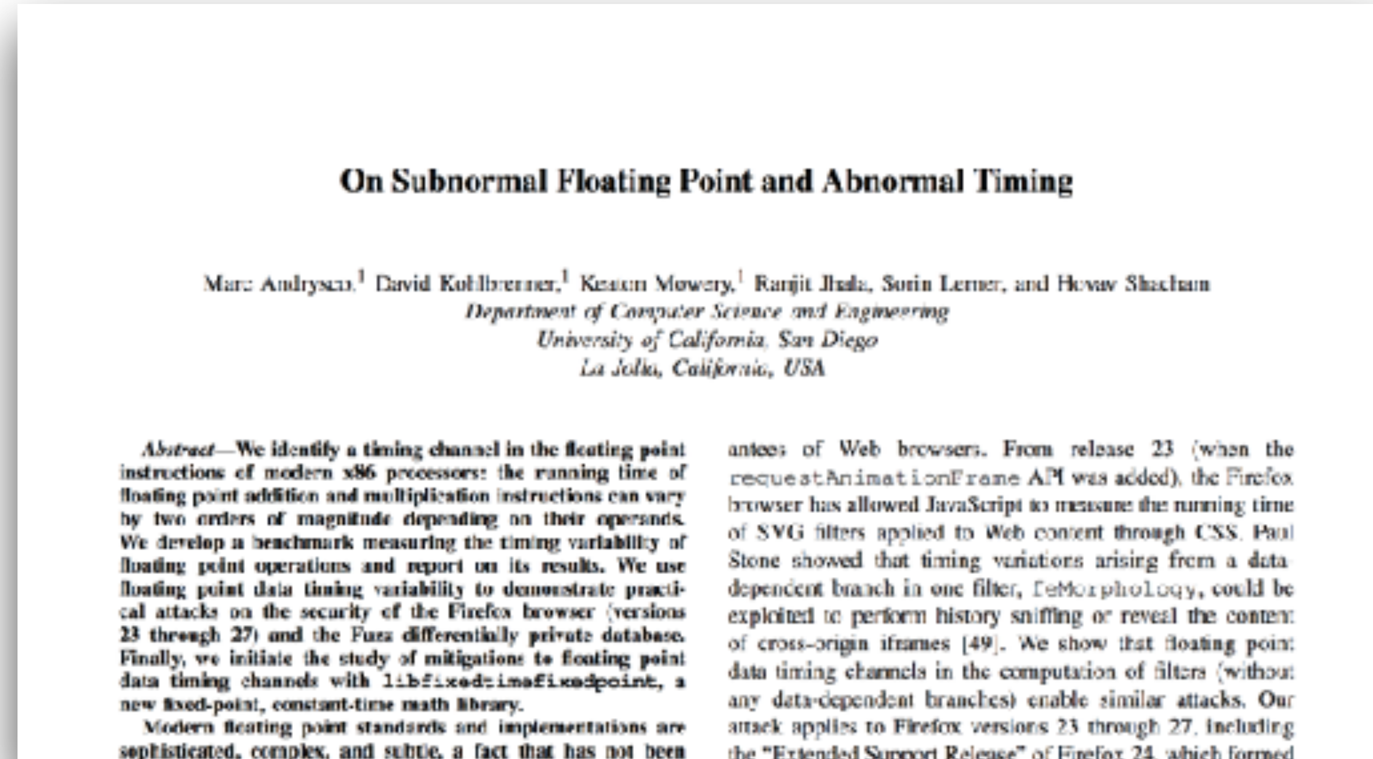
1 Introduction

The line of work on privacy of statistical databases starting with seminal papers by Dwork et al. [DN04, Dwo06, DMNS06] advanced a general approach towards achieving privacy via randomization of outputs. Instead of

- Attack based on irregularities of floating point implementations of the Laplace mechanism,
- A solution: snapping mechanism
- How about other mechanisms?

Ilya Mironov:
On significance of the least significant bits for differential privacy. ACM CCS 2012

Example 3: the floating point case



On Subnormal Floating Point and Abnormal Timing

Marc Andrysco,[1] David Kohlbrenner,[1] Keaton Mowery,[1] Ranjit Jhala, Sorin Lerner, and Hovav Shacham
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California, USA

***Abstract*—We identify a timing channel in the floating point instructions of modern x86 processors: the running time of floating point addition and multiplication instructions can vary by two orders of magnitude depending on their operands. We develop a benchmark measuring the timing variability of floating point operations and report on its results. We use floating point data timing variability to demonstrate practical attacks on the security of the Firefox browser (versions 23 through 27) and the Fuzz differentially private database. Finally, we initiate the study of mitigations to floating point data timing channels with `libfixedtimefixedpoint`, a new fixed-point, constant-time math library.**

Modern floating point standards and implementations are sophisticated, complex, and subtle, a fact that has not been

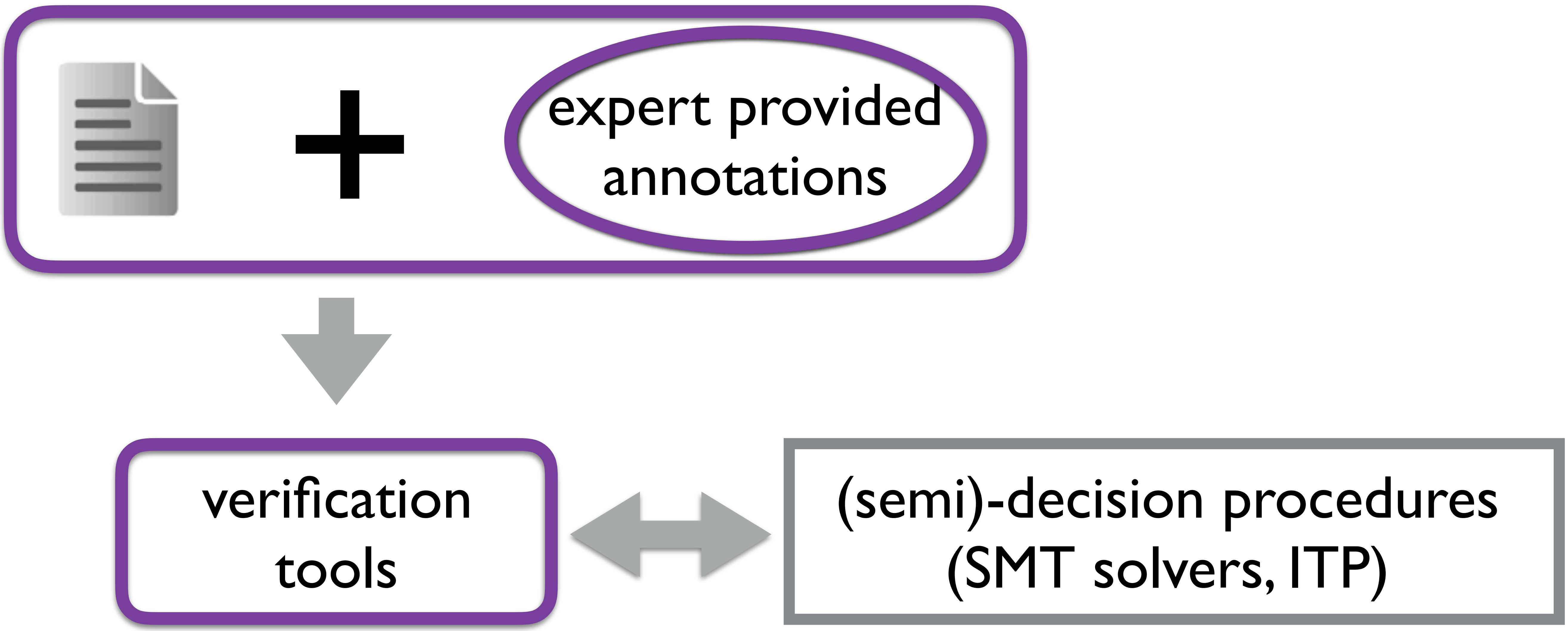
antees of Web browsers. From release 23 (when the `requestAnimationFrame` API was added), the Firefox browser has allowed JavaScript to measure the running time of SVG filters applied to Web content through CSS. Paul Stone showed that timing variations arising from a data-dependent branch in one filter, `feMorphology`, could be exploited to perform history sniffing or reveal the content of cross-origin iframes [49]. We show that floating point data timing channels in the computation of filters (without any data-dependent branches) enable similar attacks. Our attack applies to Firefox versions 23 through 27, including the "Extended Support Release" of Firefox 24, which formed

- Timing attack based on x86 difference of addition/multiplication running time difference,
- A solution: a constant time library.

Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, Hovav Shacham:
On Subnormal Floating Point and Abnormal Timing. IEEE Symposium on Security and Privacy 2015

What we have so far…

A 10 thousand ft view on program verification



Verification tools

- They handle well logical formulas, numerical formulas and their combination,
- They offer limited support for probabilistic reasoning.

We need a good abstraction of the problem.

Compositional Reasoning about the Privacy Budget

Sequential Composition
Let M_i be ϵ_i-differentially private $(1 \leq i \leq k)$.
Then $M(x) = (M_1(x), \ldots, M_k(x))$ is $\sum_{i=0}^{k} \epsilon_i$.

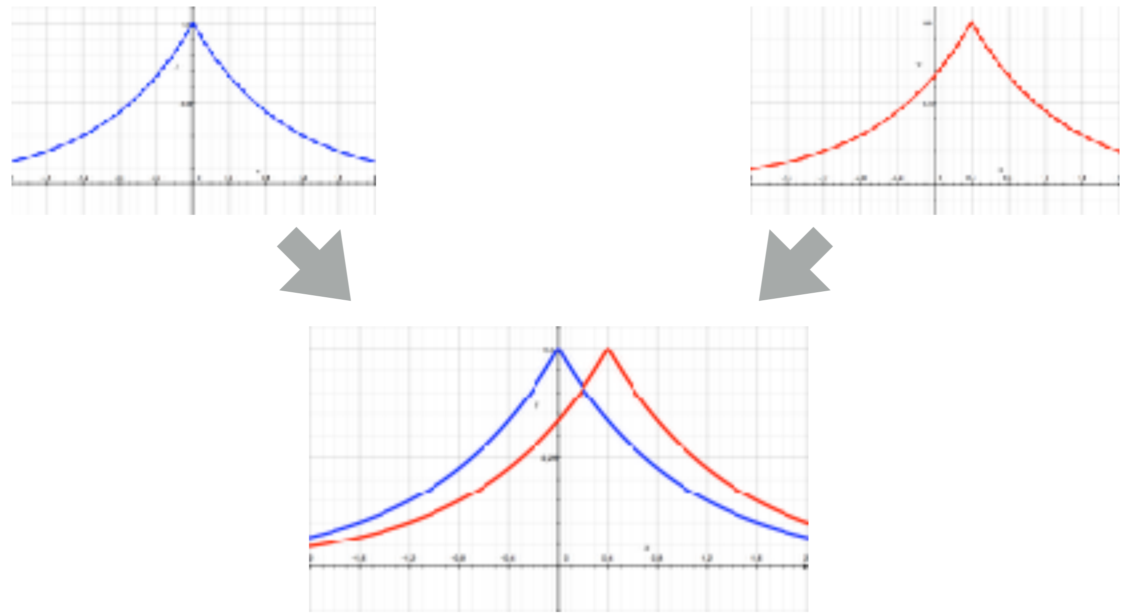
- We can reason about the privacy budget,
- If we have basic components for privacy we can just focus on counting,
- It requires a limited reasoning about probabilities,
- Implemented in different tools, e.g. PINQ(McSherry'10), Airavat (Roy'10), etc.

Compositional reasoning about sensitivity

$$GS(f) = \max_{v \sim v'} |f(v) - f(v')|$$

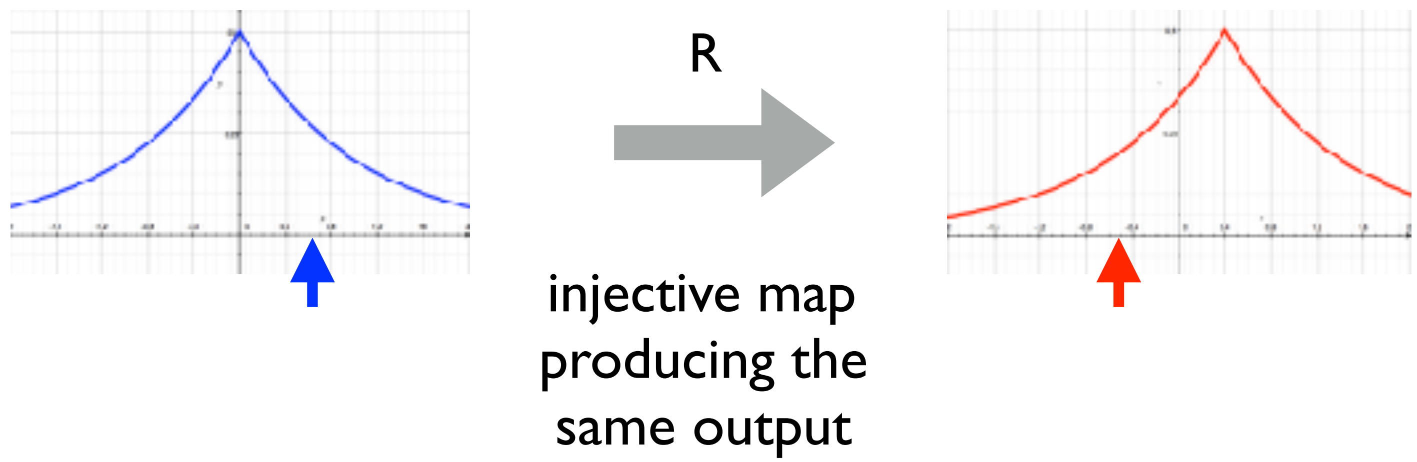
- It allows to decompose the analysis/construction of a DP program,
- It requires a limited reasoning about probabilities,
- Similar reasoning as basic composition.
- Implemented using type-checking in Fuzz (Reed&Pierce'10),
- Recently extended to AdaptiveFuzz (Winograd-cort&co'17).

Reasoning about DP via Approximate Probabilistic



- Generalize pointwise-observations to other relations allowing more general relational reasoning,
- More involved reasoning about divergences,
- Formal proof of the correctness of sparse vector,
- Implemented in EasyCrypt and HOARe^2 (Barthe&al'13,'15)
- Recently extended to zCDP, RDP (Sato&al'17)
- New, fully automated version (Albarghouthi&Hsu'17)

Semi-automated DP proofs using Randomness Assignments



- Permits to build more flexible reasoning about correspondences between the programs, and the privacy budget,
- requires few annotations and can be combined with other tools making it almost automated,
- the proof of sparse vector only requires 2 lines of annotations,
- implemented in LightDP (Zhang&Kifer'17)

Other works

- Bisimulation based methods (Tschantz&al - Xu&al)
- Fuzz with distributed code (Eigner&Maffei)
- Satisfiability modulo counting (Friedrikson&Jha)
- Bayesian Inference (BFGGHS)
- Accuracy bounds (BGGHS)
- Continuous models (Sato)
- zCDP (BGHS)
- ….
- Many other systems.

Looking forward…

Abstract?

or

Concrete?

Basic Mechanism Implementation

- We aim at verifying end-to-end a basic, realistic mechanism (from the algorithm to the code),
- We focus on a mechanism for the local model of differential privacy (simpler mechanisms, practically relevant),
- We are looking at mechanisms that have good privacy-utility tradeoff, and are efficient,
- We focus first on a machine independent approach, and add consider more concrete models later.

Private Heavy Hitter

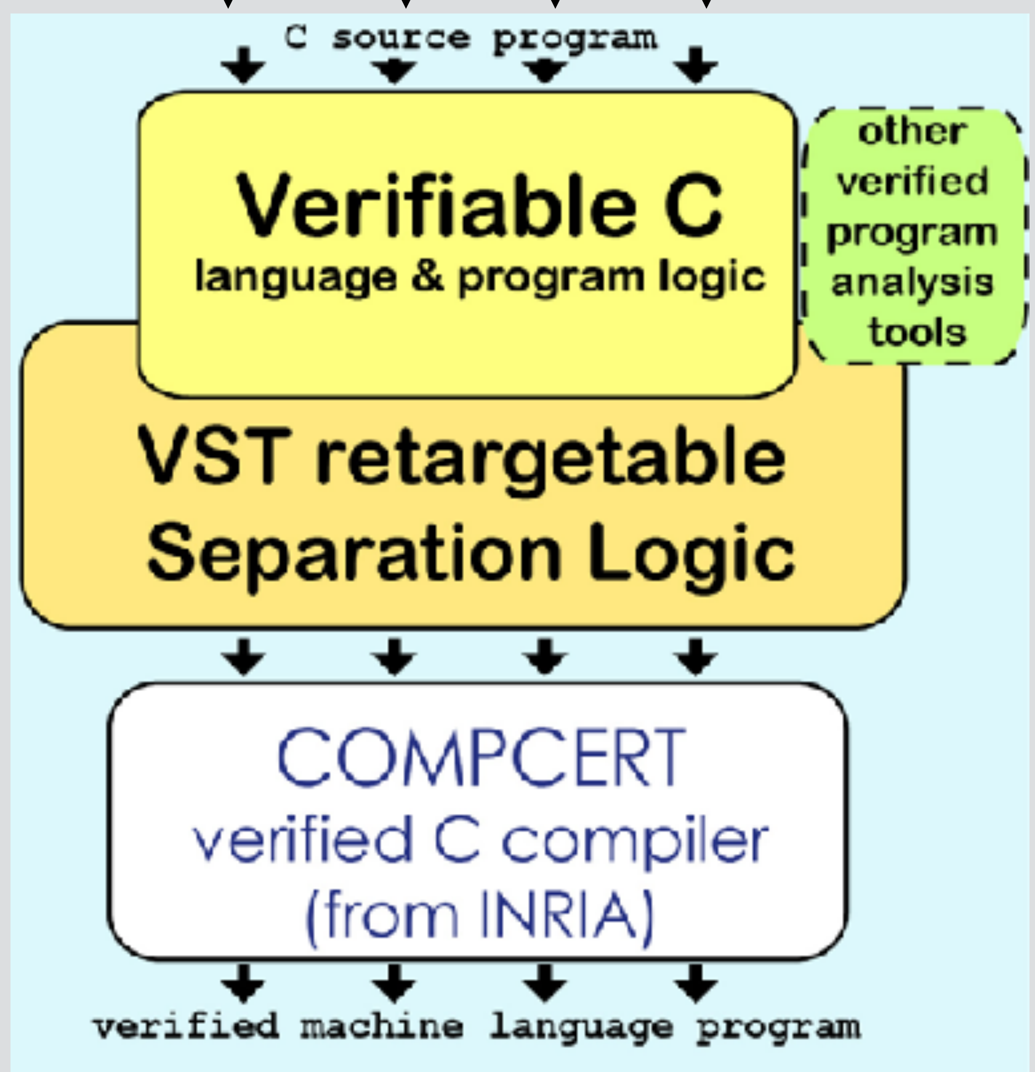
- We focus on algorithms for the heavy hitter problem: practically relevant and a availability of several different algorithms,
- We are implementing the TreeHist algorithm by Bassily&al'17 which provides a good accuracy and is efficient.
- The privacy guarantee is obtained through a simple randomized response mechanism,
- It makes non trivial transformations both on the client and server side.

Our approach

Formal Logic based on coupling

Foundational Cryptography Framework

Petcher&Morrisett'15



Coq
proof assistant

Appel&al

Recently used for HMAC for OpenSSL, (part of)TLS.

Expected Outcomes

- Many months of work!
- Increasing the confidence on the correctness of the mechanism implementation,
- Development of techniques for proving correct basic mechanisms from the local model.

Thanks