Modern Cryptography

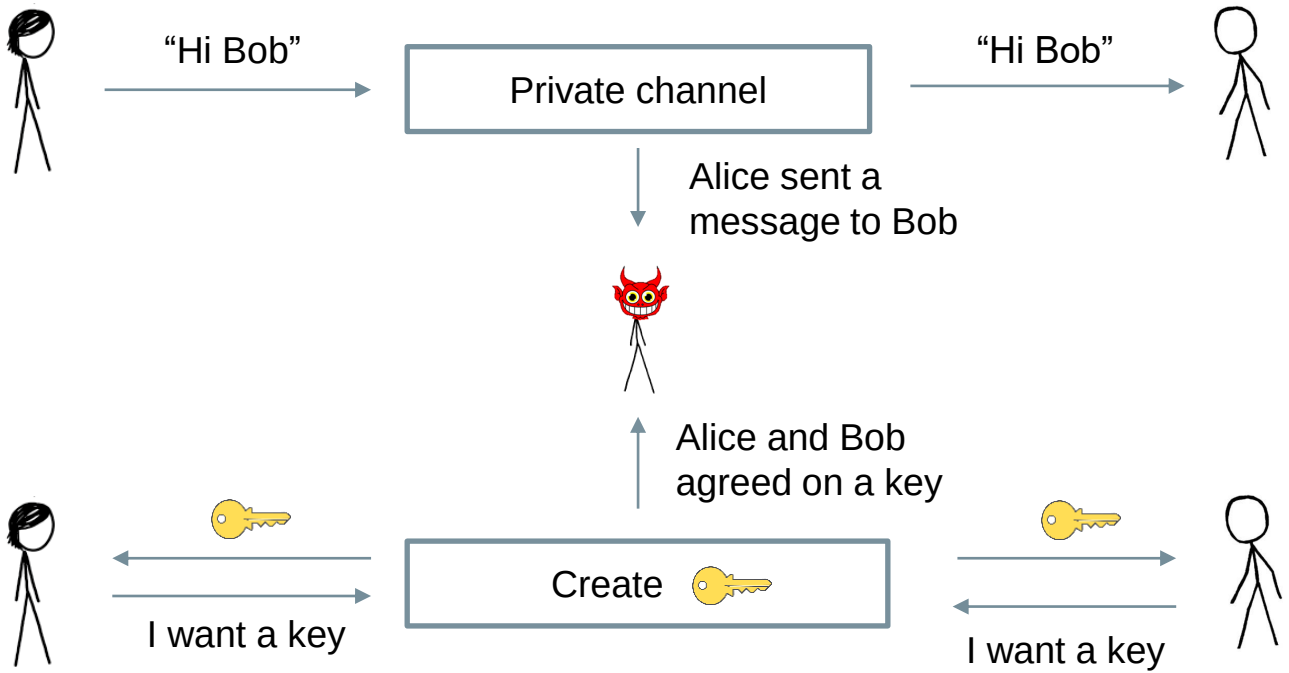
Through the Lens of Secure Computation

Carsten Baum

Conventional view of Cryptography

- Alice sends a message to Bob so Eve can’t listen
- Alice and Bob agree on a key over a public channel and only they know the key

Conventional cryptography - in boxes



1982: Andy Yao has an idea

What if the box can compute on inputs?

Protocols for Secure Computations
(extended abstract)

Andrew C. Yao

University of California
Berkeley, California 94720

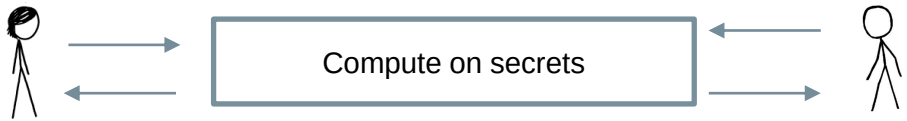
1. INTRODUCTION.

Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?

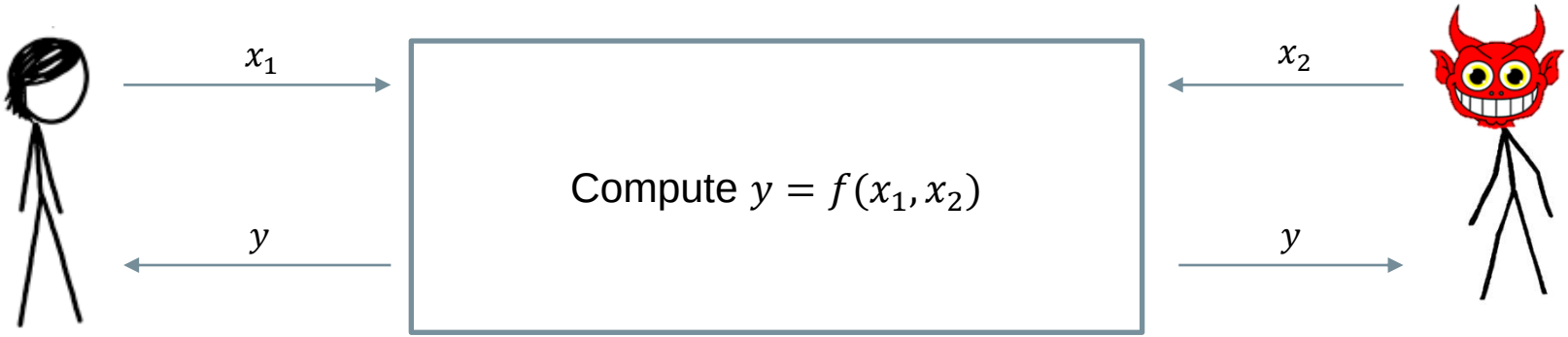
This is a special case of the following general problem. Suppose m people wish to compute the value of a function $f(x_1, x_2, x_3, ..., x_m)$, which is an integer-valued function of m integer variables x_i of bounded range. Assume initially person P_i knows the value of x_i and no other x's. Is it possible for them to compute the value of f, by communicating among themselves, without unduly giving away any information about the values of their own

desirable to have a unified framework where all these applications can be related, and where common proof techniques can be developed for proving the security of protocols. More fundamentally, such a framework is essential if we are ever to understand the intrinsic power and limitation of one-way functions. For example, without a precise model it would be hard to answer a question such as "Is it possible for three mutually suspecting parties to interactively generate a bit with bias $1/e$?".

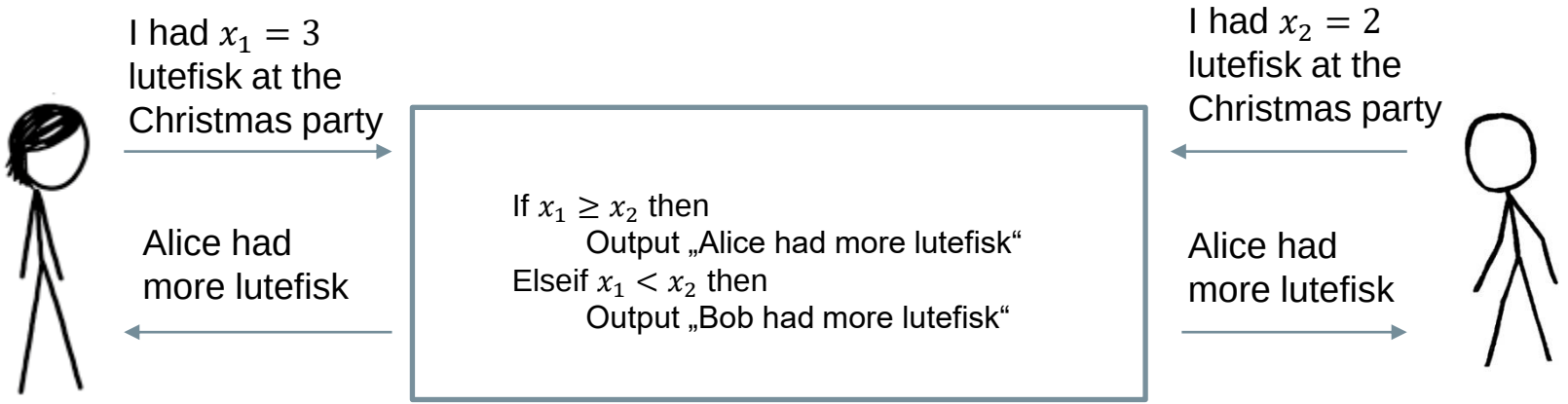
In response to this need, we propose to adopt the following view. Two parties Alice and Bob, in possession of private variables i and j respectively, wish to communicate so that Alice can evaluate a function f(i, j), and Bob a function g(i, j). There may be some eavesdroppers or saboteurs on the communication line.



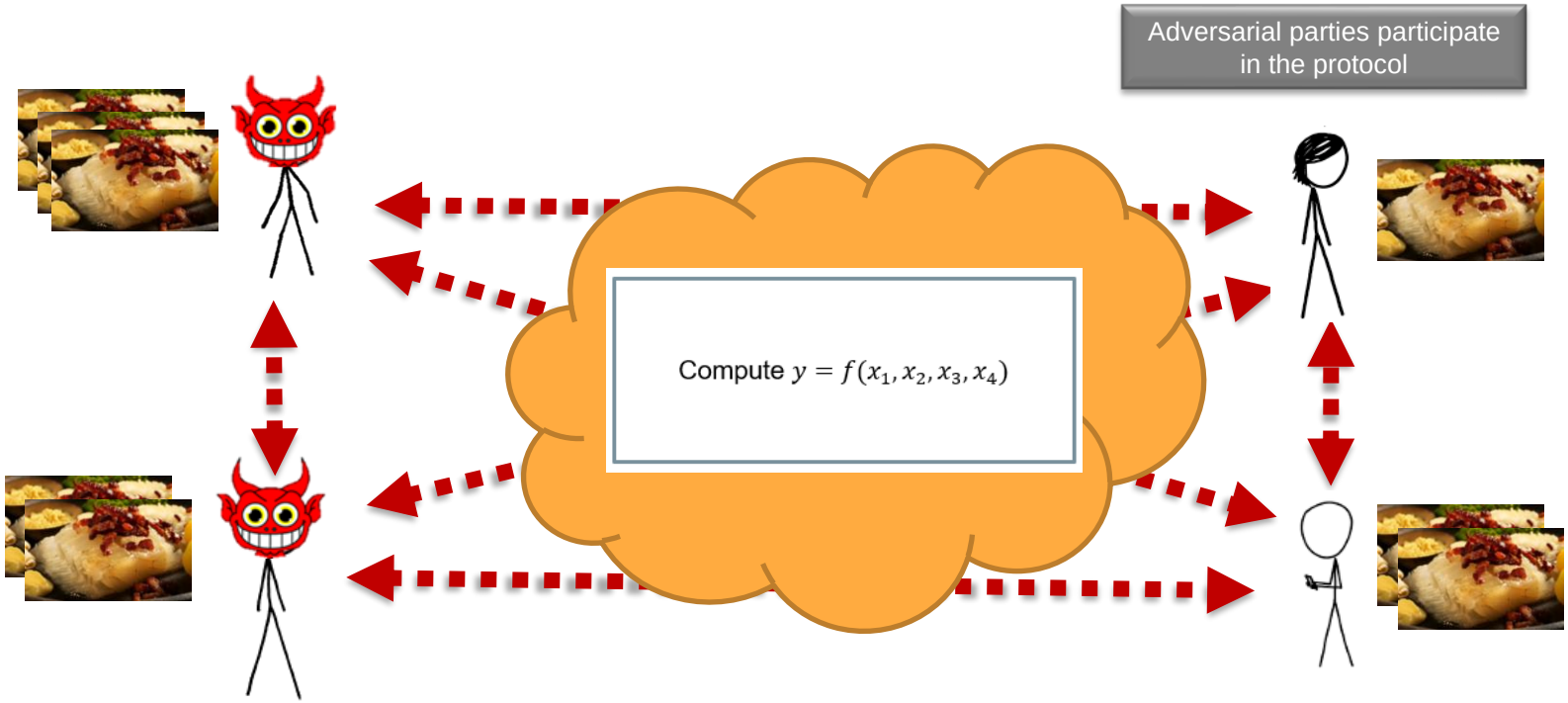
Secure Computation (MPC)



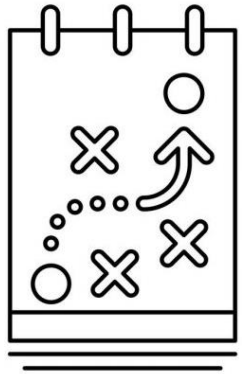
Secure Computation (MPC)



Magic Boxes don’t exist



Plan

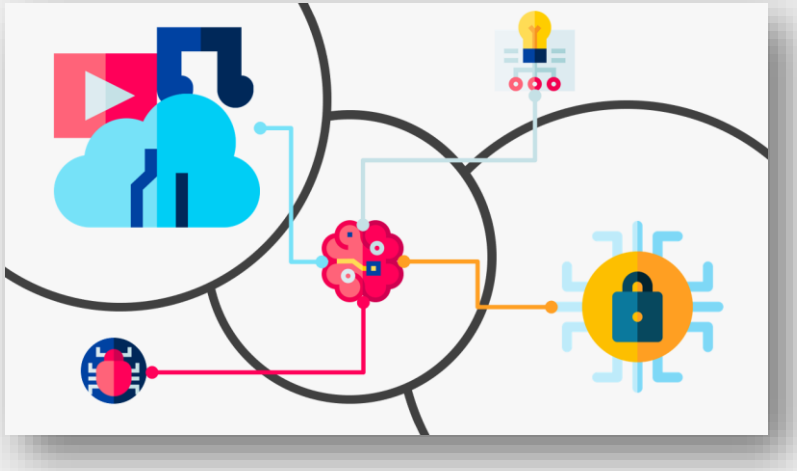


1. Secure Computation & Machine Learning
2. Zero-Knowledge Proofs & Signatures
3. Zero-Knowledge Proofs & Software Bugs

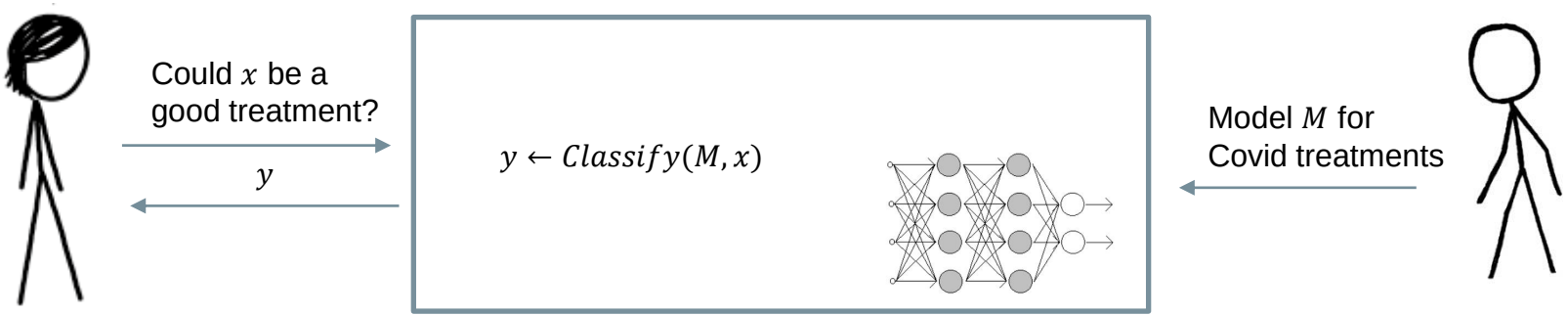
What this talk is about	**What this talk is NOT about**
Conceptual ideas State of the Art	Concrete protocols

MPC for ML

What cool kids do these days



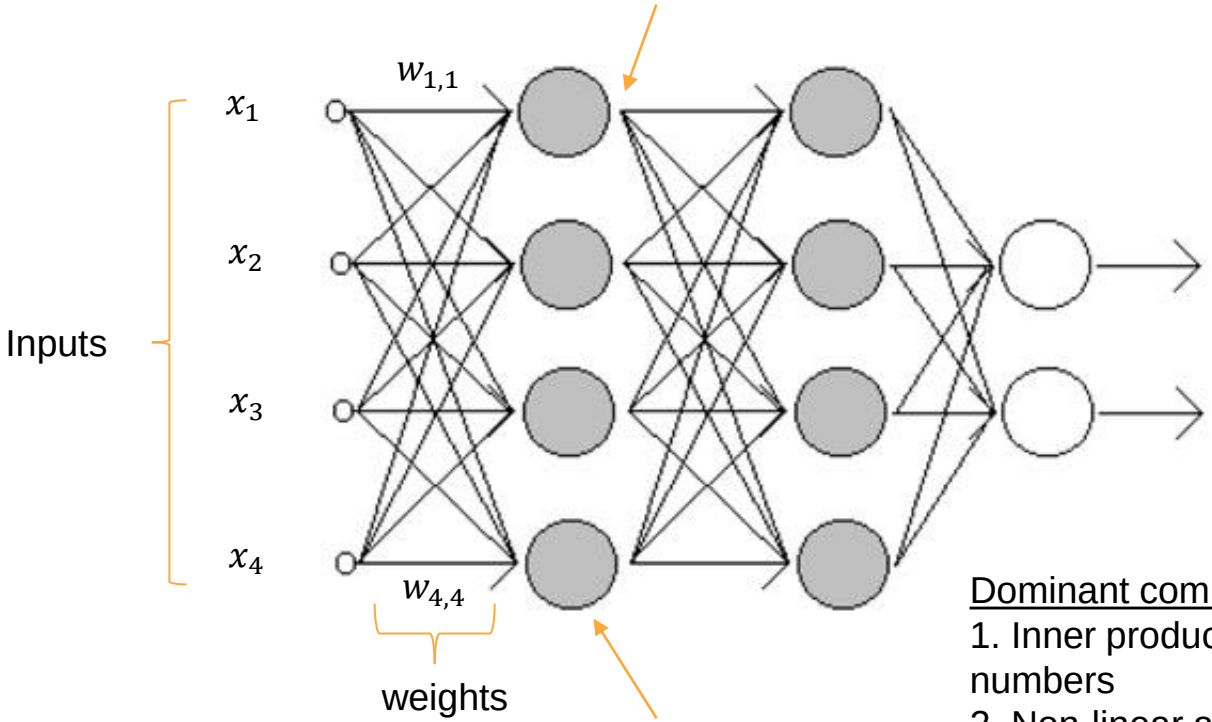
Goal 1: Secure inference*



*for Deep Neural Nets

Evaluating a DNN

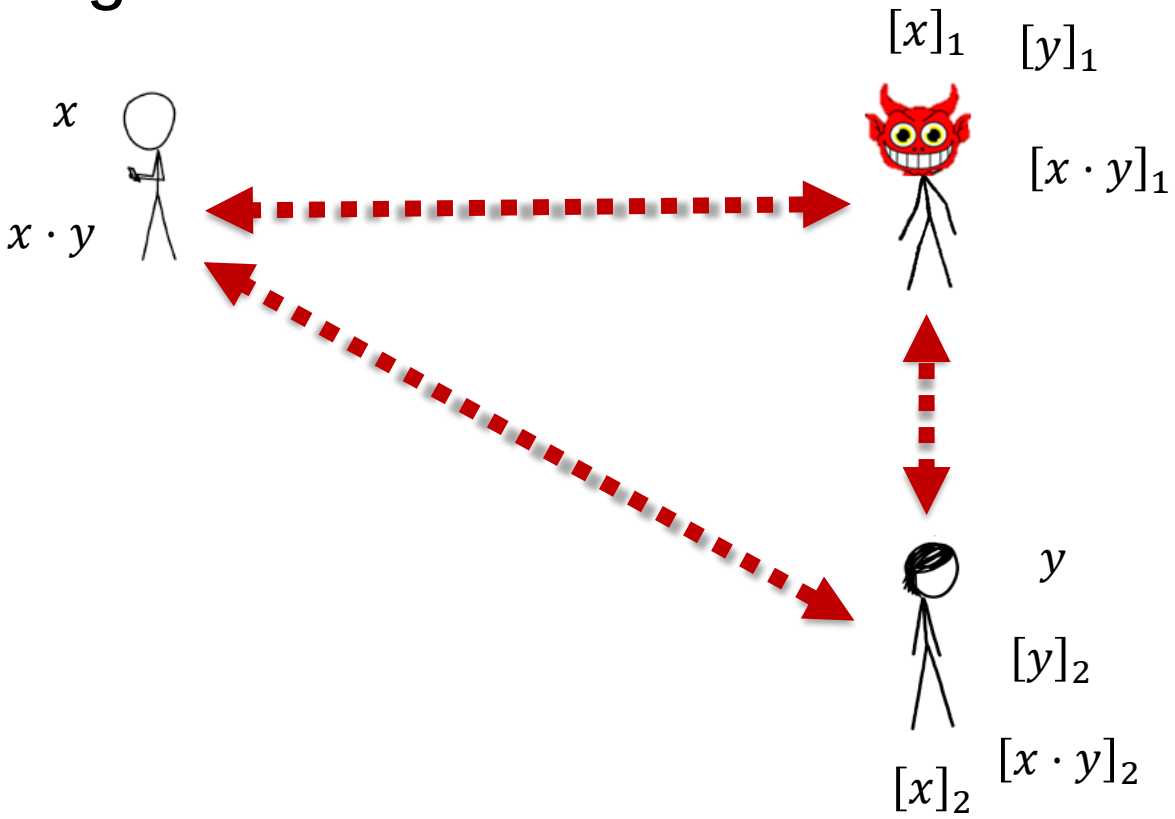
$$x_1' = f(b_1 + \sum_i x_i \cdot w_{1,i})$$



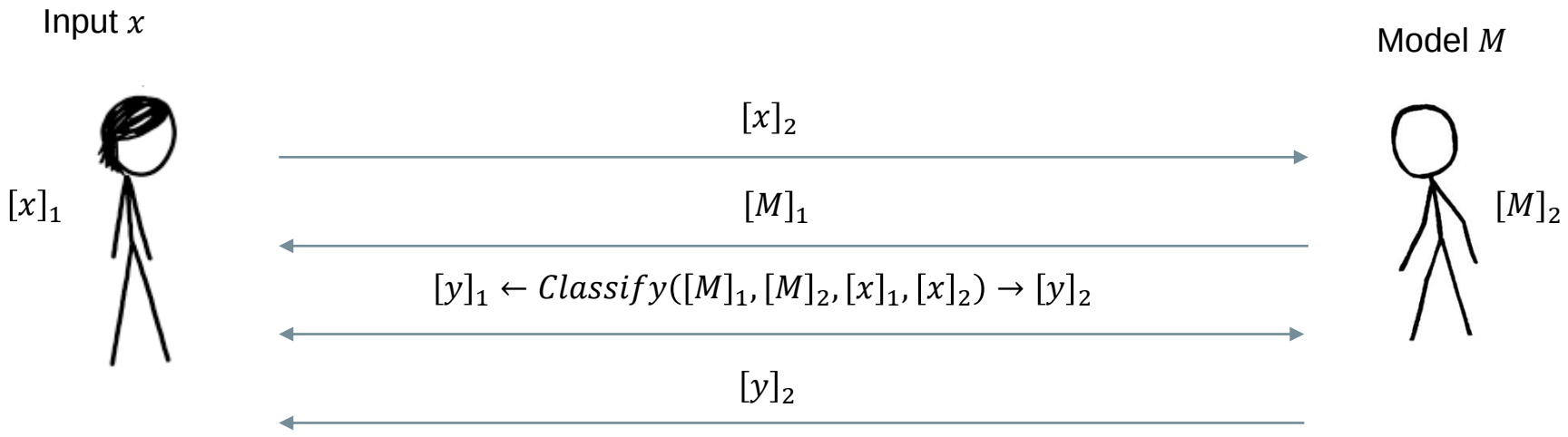
Dominant computations:

1. Inner product with float-point numbers
2. Non-linear activation (ReLu, Sigmoid,…)

Secret Sharing



Secure Inference



$y = Classify(M, x)$

1. How to do float arithmetic?
2. How expensive are inner products?
3. What is the cost of computing activation?

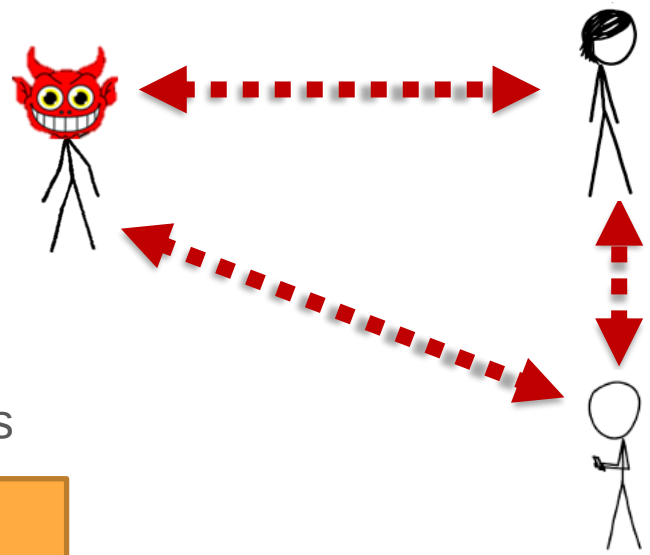
Realizing the Secure Inference box

1. Secret-Sharing not efficient for floats. Use fixed-point arithmetic over Z_{2^n} or modulo large primes

2. Inner products are cheap for certain secret sharing

3. Use quantized network architecture

4. Approximate activation function as polynomials



Potential loss in accuracy!

What can we say about efficiency? (SecureQ8)

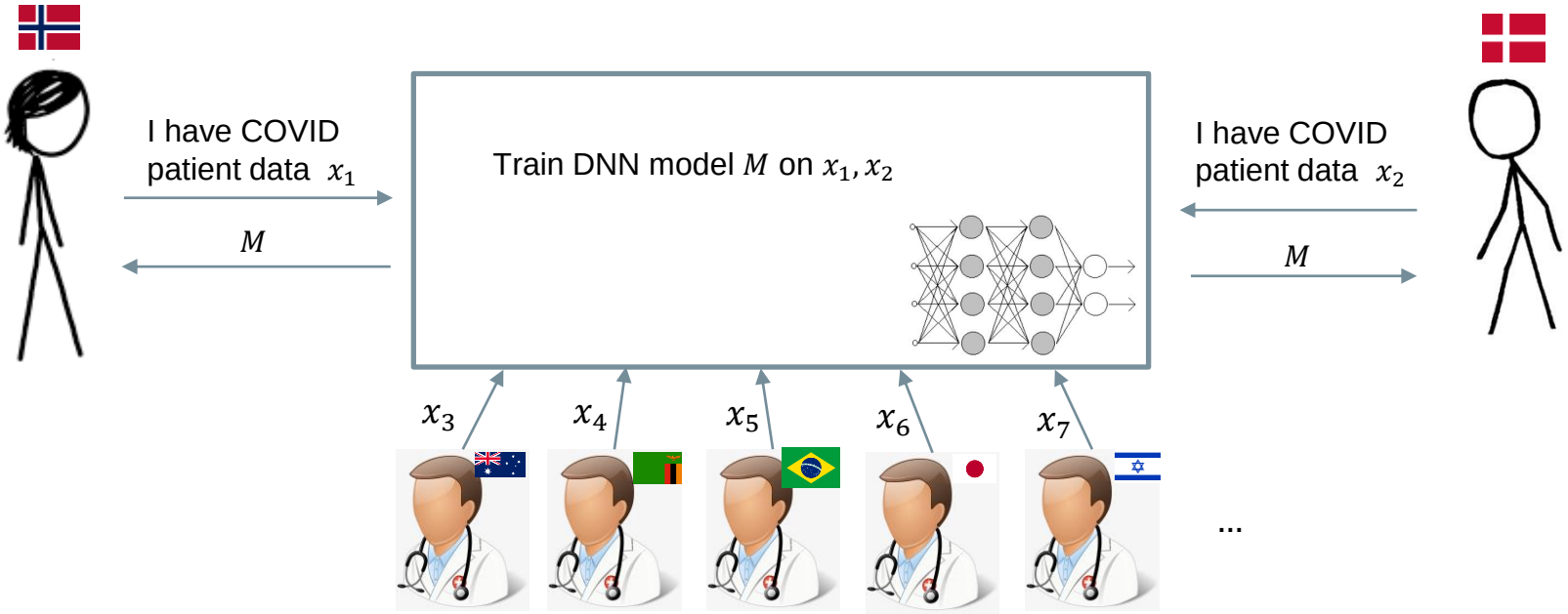
		Passive Security			Active Security		
		Dishonest Maj.		Honest Maj.	Dishonest Maj.		Honest Maj.
	# parties	$\mathbb{Z}_{2^k}$	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{Z}_{2^k}$	$\mathbb{F}_p$	$\mathbb{F}_p$
Time (s)	3	401.1	320.9	5.5	2456.8	2255.0	24.8
	4	799.6	597.4	7.4	3632.8	3063.7	36.0
	5	1332.9	959.5	15.8	4814.3	3921.0	54.7
Comm. (GB)	3	594.8	114.8	7.3	3513.8	515.2	31.8
	4	1183.0	232.2	8.2	5266.5	766.4	35.8
	5	1965.1	389.2	20.7	7018.7	1016.7	68.6

Table 4. Time and communication per party for computing V1 0.25_128 with probabilistic truncation.

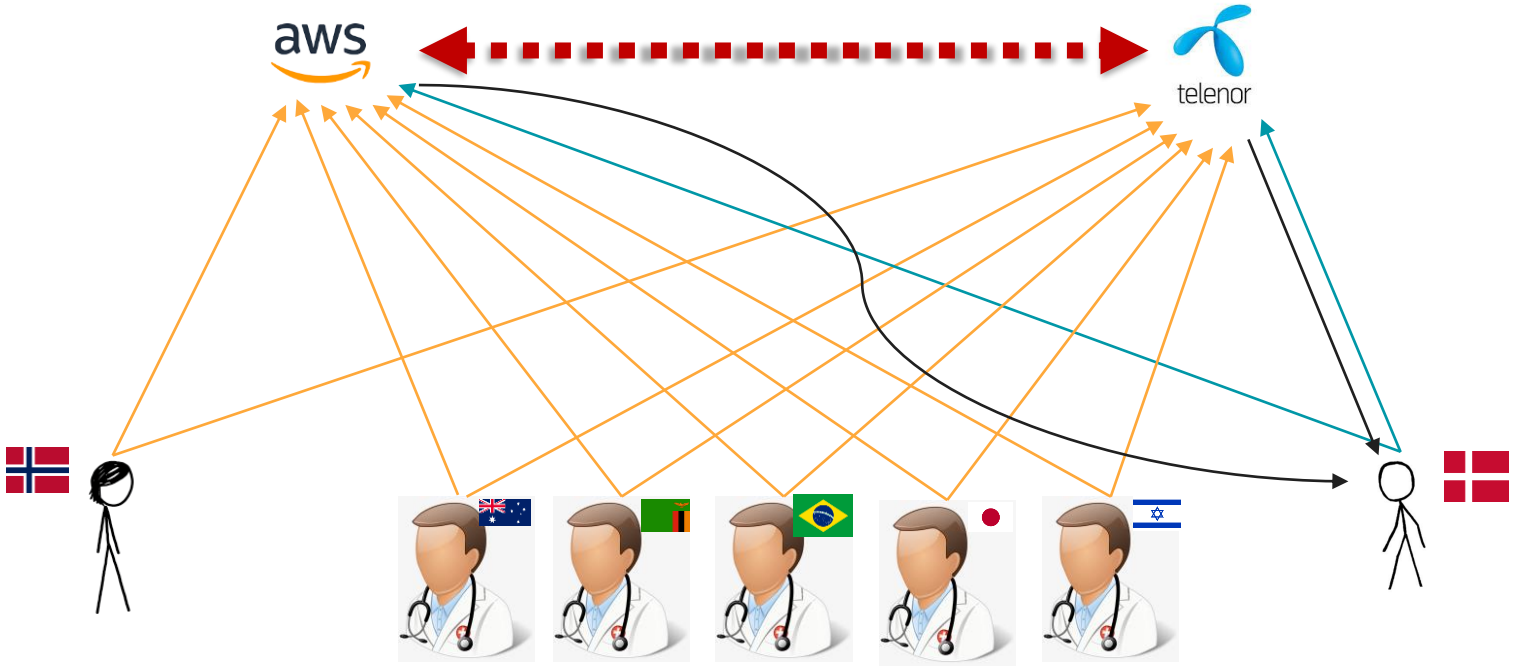
Platform: c5.9xlarge AWS machines with 36 cores, 72gb RAM, a 10gpbs link and sub-millisecond latency
Smallest architecture of MobileNet for ImageNet
Huge cost for dishonest majority (2 out of 3 corrupted)

Comparison:
on 2 cores of a Cortex A72, this takes <200ms in plain (Raspberry Pi 4)

Goal 2: Secure training (of Neural Nets)



Client-Server model



Example: MNIST

Recent work, using LeNet & 16 epochs on MNIST dataset (10 classes, 28x28 binary images)

Training done in MP-SPDZ, accuracy comparable to non-private training, 3 party, 1 corrupted, semi-honest

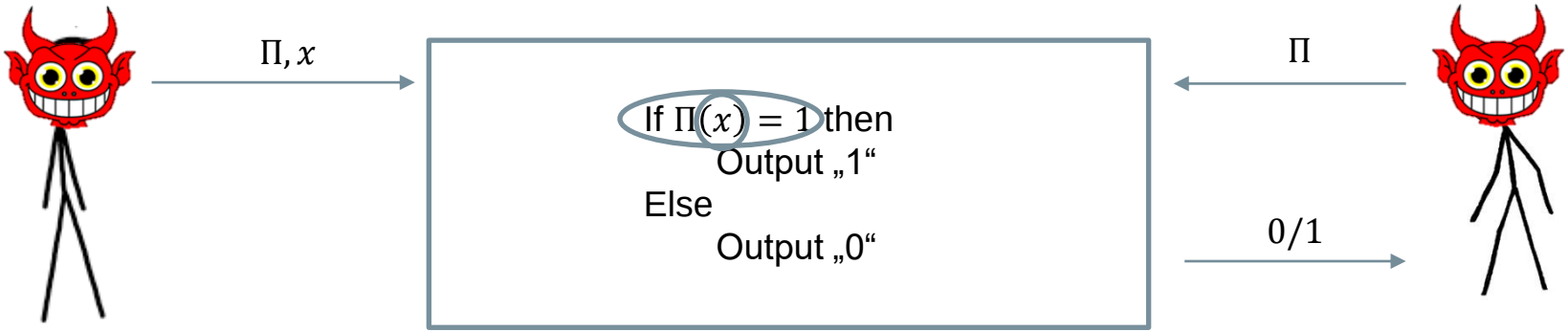
Training per epoch:
340s. 16 epochs

Expected result in plain:
>100x faster, plus speed-up using GPUs

ZK & Signatures

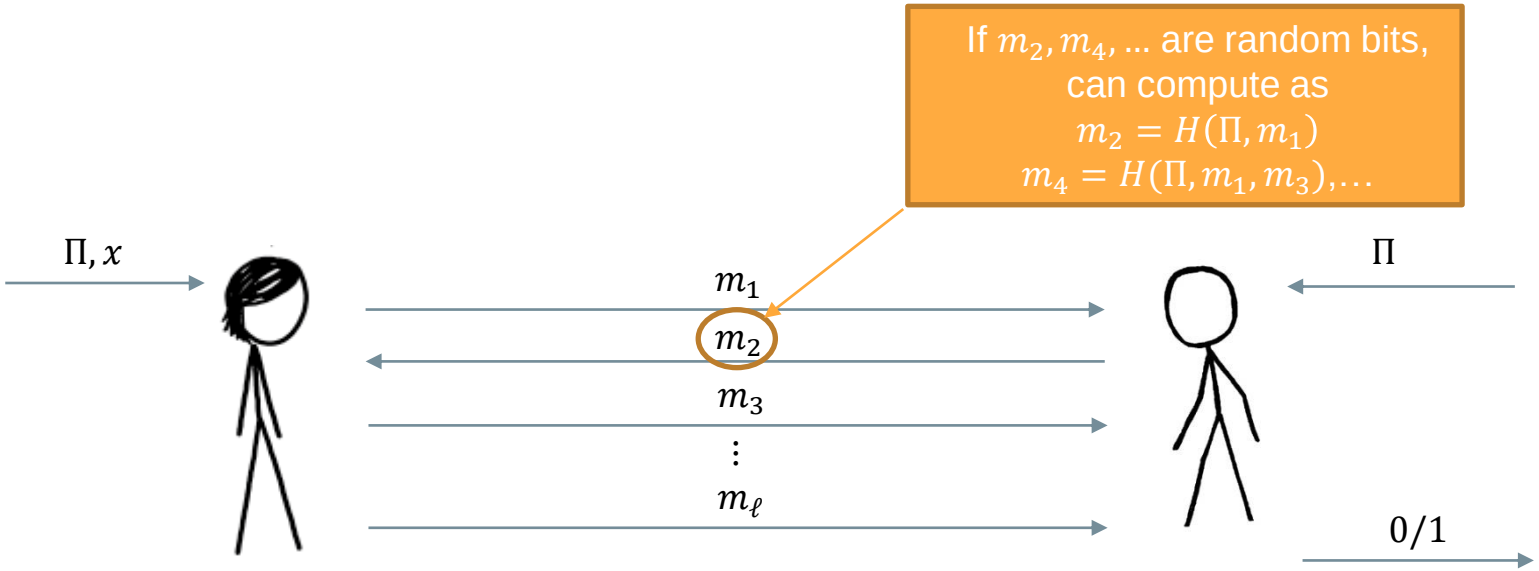
Signatures without public-key assumptions

Zero-Knowledge Proofs



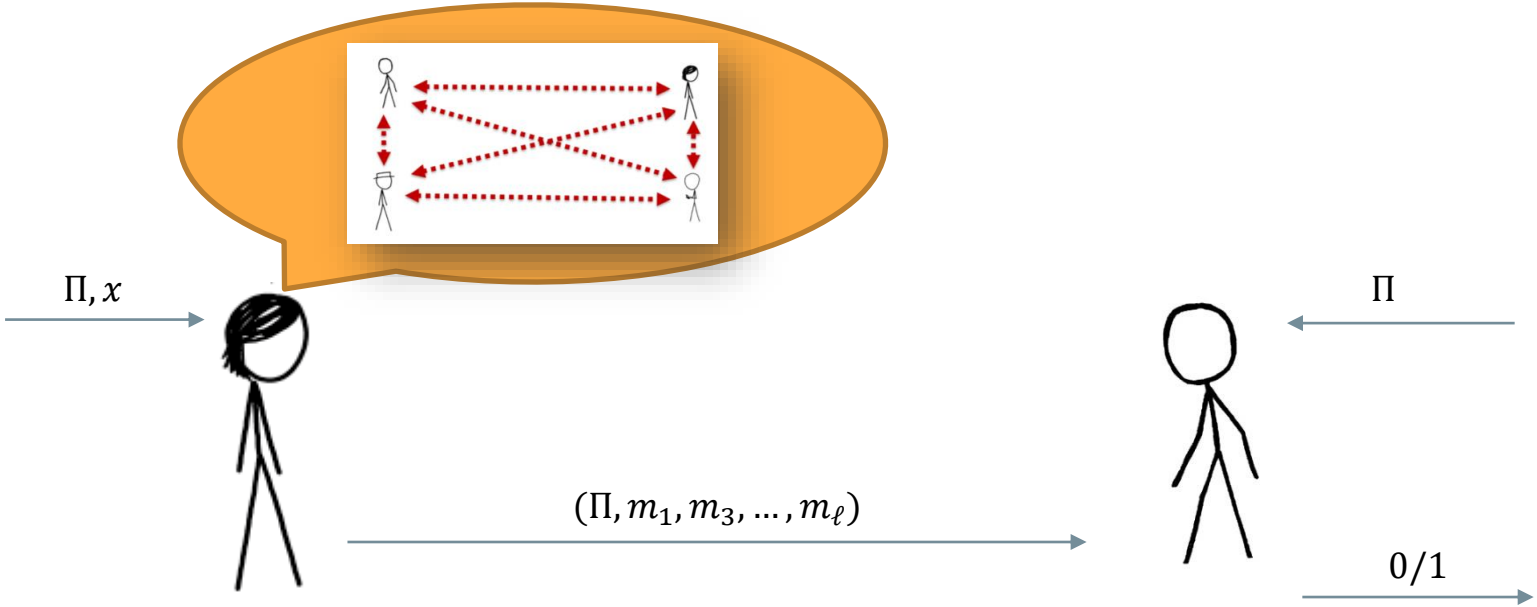
1. Completeness
2. Knowledge Soundness
3. Zero-Knowledge

Zero-Knowledge = Special form of MPC



$(\Pi, m_1, m_3, \ldots, m_\ell)$ is a non-interactive proof of Π!

Build ZK by simulating MPC…?



Many protocols available: ZKBoo, KKW, BN, Limbo, Ligero, STARKs,…

Interesting criteria: *Prover runtime* vs. *Proof size* vs. *Verifier runtime*

Signatures from ZK proofs

Let $Enc_k(\cdot)$ be encryption of block cipher under key k

Define: $sk = (k), vk = (x, Enc_k(x))$

How to sign:

1. Prove knowledge of k for vk in ZK by showing
   $y = Enc_k(x)$ for known $vk = (x, y)$ (ID scheme)
2. Also hash message m when computing verifier msgs

Verify signature: verify ZK proof & message!

Algorithms

1. ZKB++ proof system & Picnic block cipher (NIST Round 3 alternative candidate)
2. KKW & AES
3. BN & AES (BBQ, Banquet, Limbo)
4. BN & Legendre-Symbol (LegRoast)

	Prover time (ms)	Verifier time (ms)	Signature size (bytes)
Picnic	5.3	4	12466
KKW & AES	28	28	45900
Banquet	9.1	7.5	17456
Limbo	5.7	5.5	17334
LegRoast	6	6	13900
ECDSA	faster	faster	smaller

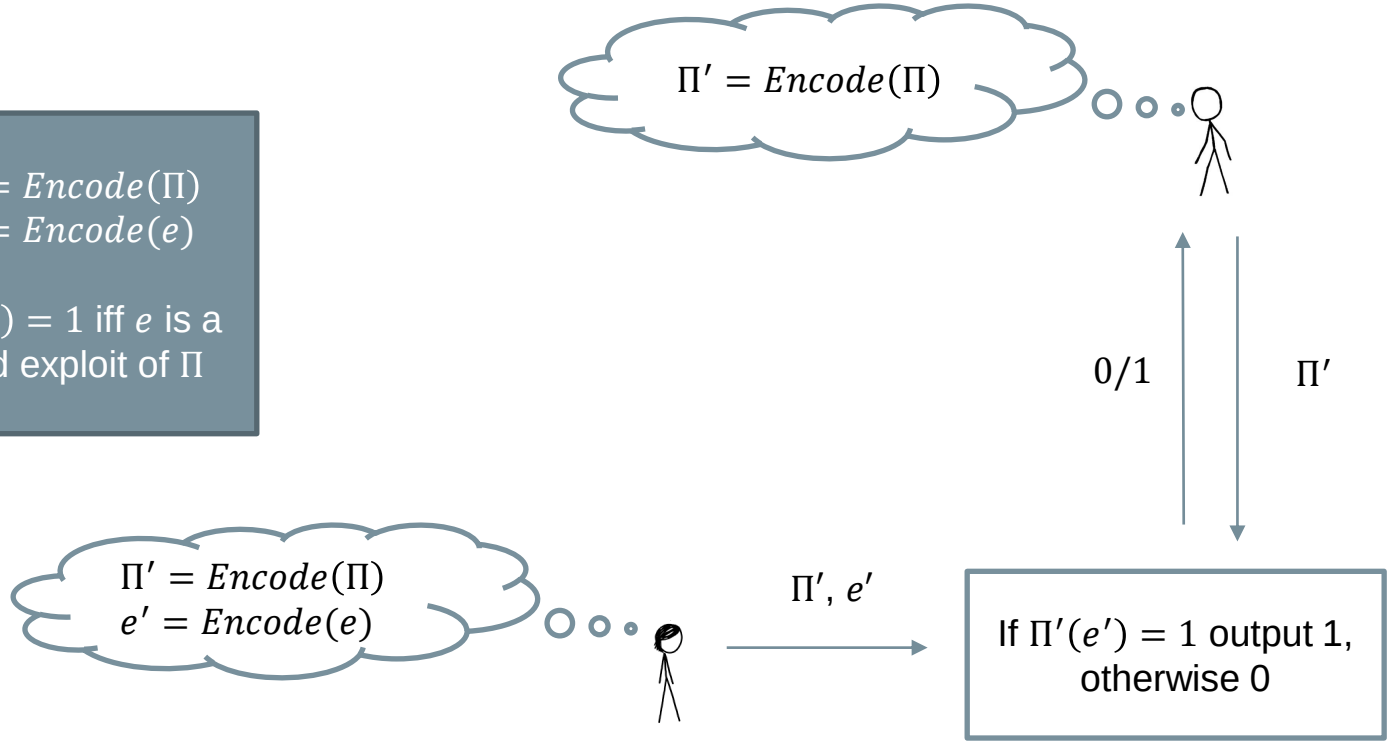
ZK & Exploits

Showing that weaknesses exist without revealing them

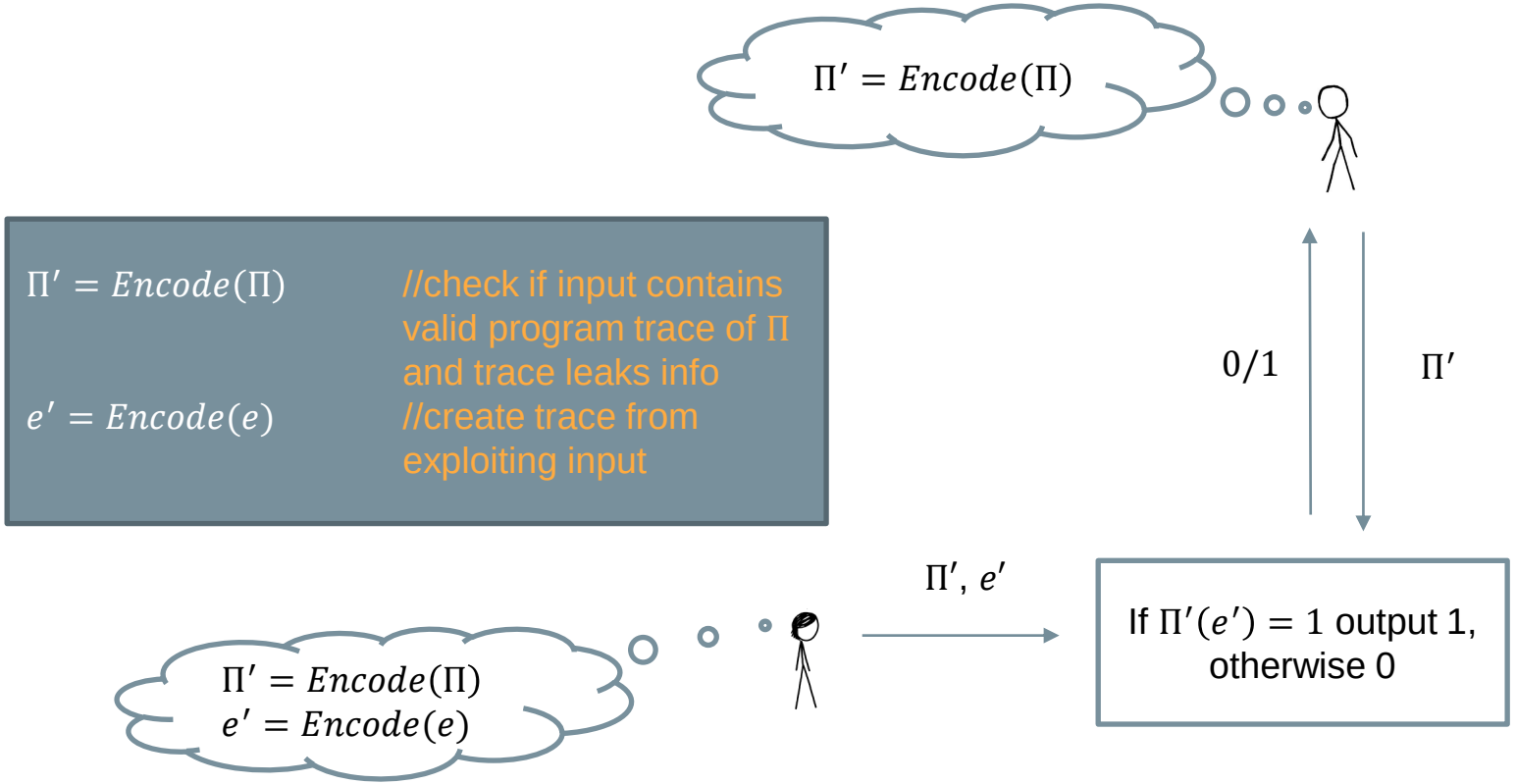
What to use these Zero-Knowledge proofs for?

$\Pi' = Encode(\Pi)$
$e' = Encode(e)$

$\Pi'(e') = 1$ iff e is a valid exploit of Π



More concretely



How to encode an exploit?

Focus on information leakage:

Label input variables as low/high sensitive, outputs as low/high visible

Two-trace approach:

Create two traces with same low-sensitive inputs, but differing low visible outputs

Two traces = 2x overhead

Information flow:

Label information flow based on program, show that leak *may* occur

Not sound

Program	x's Label
x:=y	High
x:=y+z	High
if (y==0) then x:=1 else x:=0	High

How to encode an exploit?

Focus on information leakage:

Label input variables as low/high sensitive, outputs as low/high visible

Two-trace approach:

Create two traces with same low-sensitive inputs, but differing low visible outputs

Two traces = 2x overhead

Information flow:

Label information flow based on program, show that leak *must* occur

Enough for e.g. Heartbleed

Program	x's Label
x:=y	High
x:=y+z	Low
if (y==0) then x:=1 else x:=0	Low

What do we need for this?

1. Encode programs and exploits in "Zero-Knowledge friendly" form
2. Construct ZK proof system efficient enough to execute proof*

FROMAGER project

galois | QEDit™ | DARPA | KU LEUVEN | Solidum petit in profundis · Universitas Arhusiensis | Columbia University

*also other solutions available, such as the QuickSilver proof system

Meet: The Mac'n'Cheese proof system

Proof system developed at Aarhus University and implemented by Galois, Inc.

Other proof systems: make proof size $o(|\Pi|)$, no matter the cost of prover.

Zero-Knowledge for blockchains

<u>Mac'n'Cheese:</u>

1. Proof runtime & size linear in $|\Pi|$
2. Small constants & highly efficient cryptography
3. Software is available, but not yet production-ready

| galois |

How big are the proofs?

Program	Execution Steps (K)	Num. Mults (M)
grit	**6**	**38.6**
FFmpeg	**76**	**716.0**
OpenSSL-two-trace	**~50,000 (2,600)**	**Mem. Out**
OpenSSL-labels	**~25,000 (1,300)**	**Mem. Out**

Performance with 95 ms ping, 2.25MB/s bandwidth, including preprocessing:

$\mathbb{F}_{2^{61}-1}$: **1.5 µs** per multiplication

$\mathbb{F}_2$: **140 ns** per multiplication

Summary

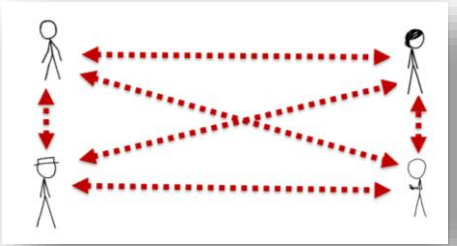
Secure computation allows to compute on data without leaking said data

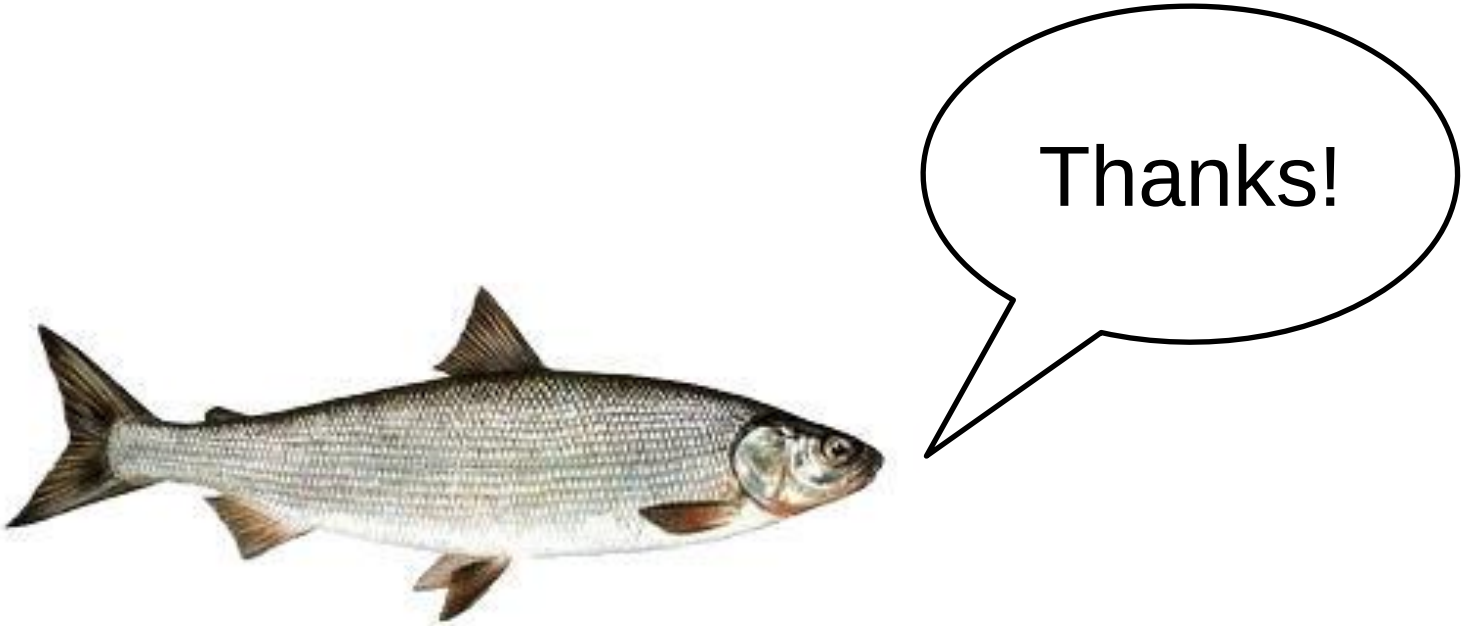
<u>Example applications:</u>

1. Privacy-Preserving Machine Learning
2. Post-quantum Signatures
3. Proofs of Exploits for Software

Applications I did not talk about:

1. Encrypted databases
2. Threshold key storage
3. Private order matching
4. …




Thanks!