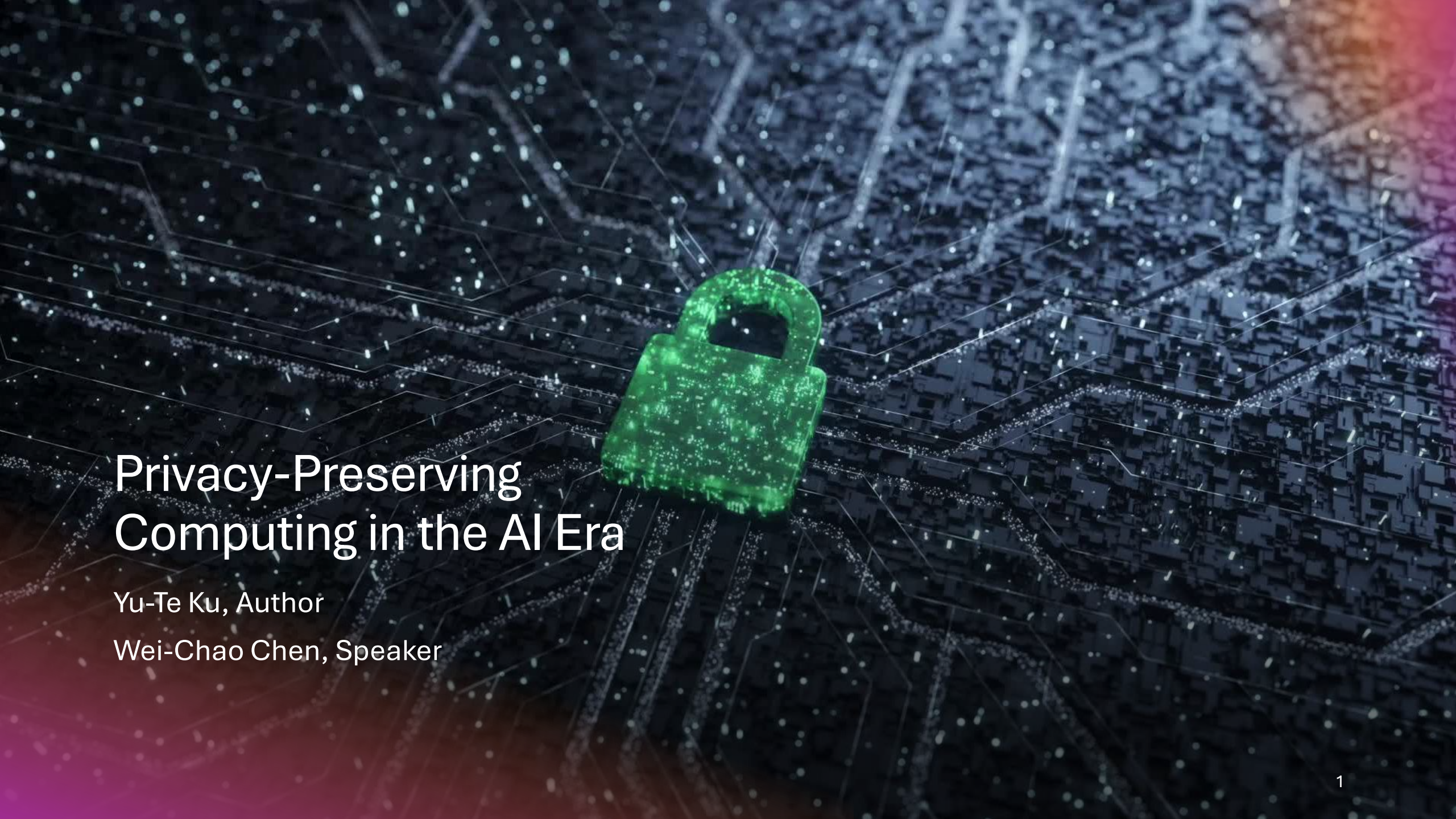


Privacy-Preserving Computing in the AI Era

Yu-Te Ku, Author

Wei-Chao Chen, Speaker

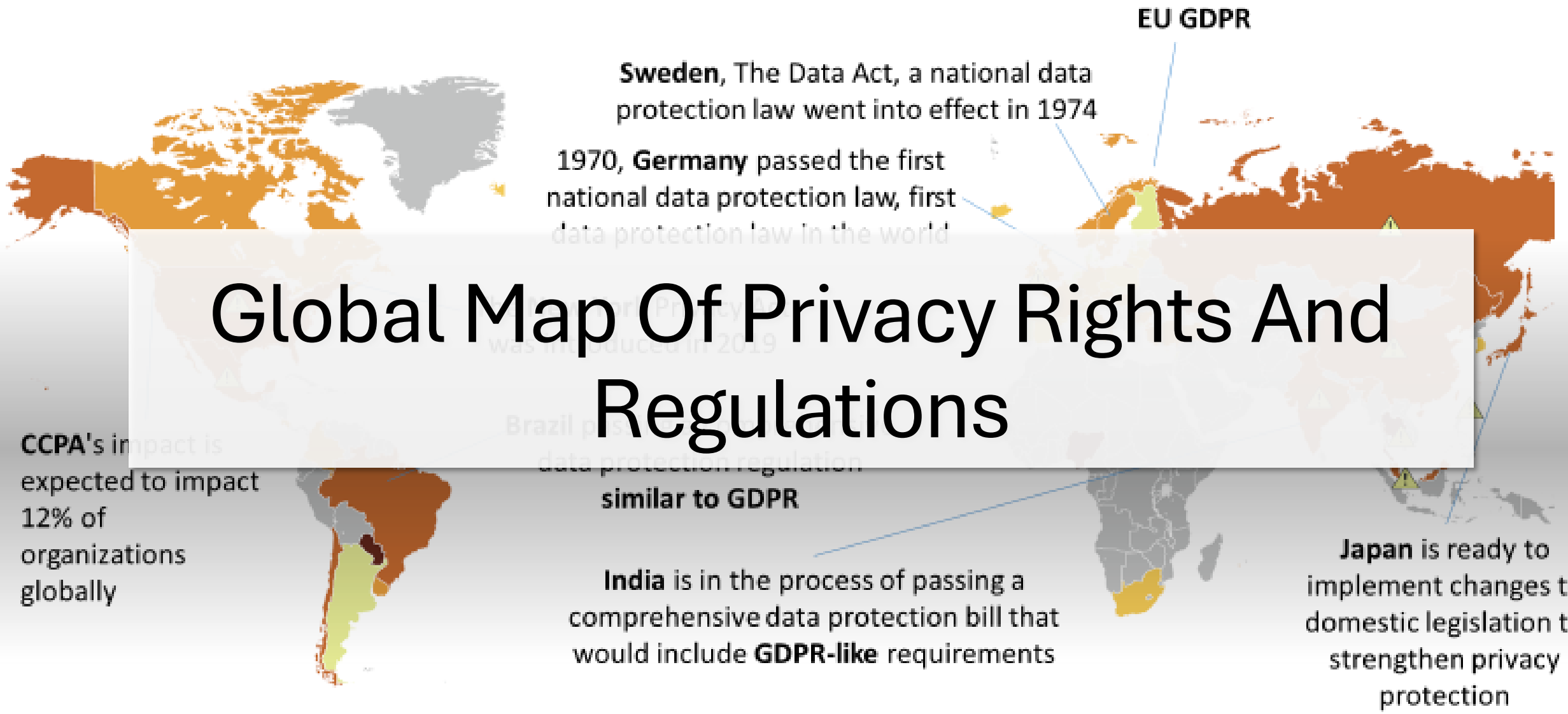
Introduction to Data Privacy

- Data privacy refers to the proper handling, processing, storage, and usage of personal information.
- **Importance:**
  - Protects individual rights and freedoms. Builds trust between consumers and organizations.
  - Essential for compliance with laws and regulations.

The Current State of Data Privacy

- Overview of Current Regulations:
    1. GDPR (General Data Protection Regulation): EU regulation that sets strict guidelines for data collection and processing.
    2. CCPA (California Consumer Privacy Act): Grants California residents more control over their personal information.
    3. Other Regulations: HIPAA (health data), COPPA (children's data), and various state laws in the U.S.

Global Map Of Privacy Rights And Regulations



Source: Forrester, PwC

Future Trends in Data Analytics

Artificial Intelligence: AI algorithms process vast amounts of data but raise ethical concerns regarding privacy.

Big Data: Increasing data volumes necessitate advanced analytics, posing risks of misuse and breaches.

Internet of Things (IoT): Connected devices collect sensitive data, increasing the vulnerability to breaches.

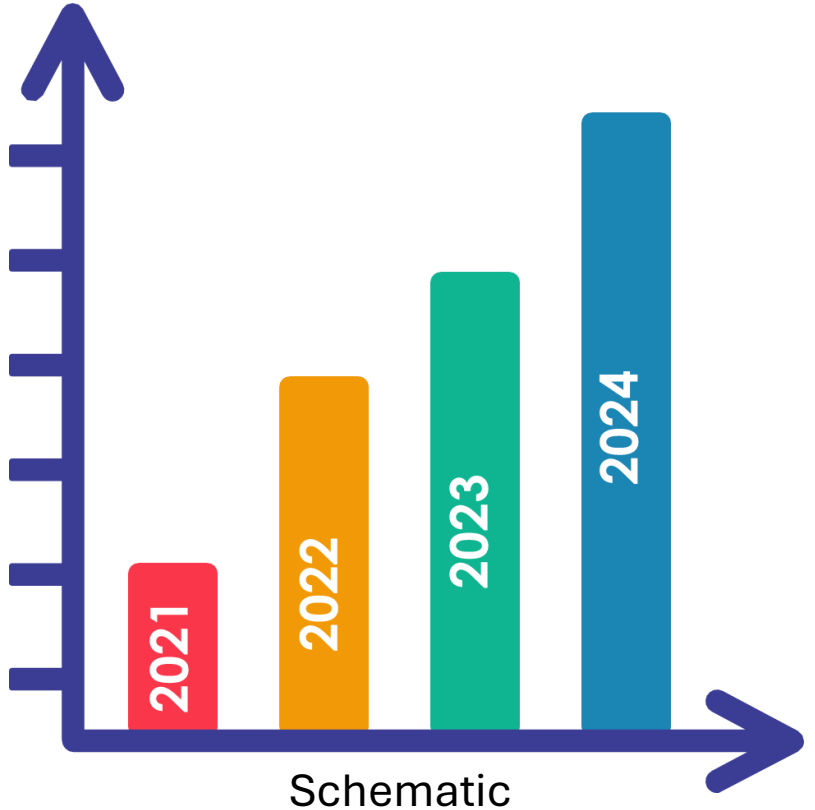
Cloud Computing: Shifts data storage to the cloud, complicating security and compliance.

Key Data Privacy Challenges

1. Data Breaches: Increasingly sophisticated cyberattacks expose sensitive data.
2. User Consent: Difficulty in obtaining informed consent due to complex privacy policies.
3. Anonymization: Challenges in effectively anonymizing data to protect individual identities.
4. Data Sovereignty: Legal issues regarding where data is stored and processed.
5. Transparency: Lack of clarity in how data is used and shared by organizations.

Data Breaches and Security Threats

- **Statistics:**
  - Data Breaches: Over 4 billion records were exposed in 2020 alone (source: Identity Theft Resource Center).
  - Costs: The average cost of a data breach is estimated at $3.86 million (source: IBM).
- **Impact:**
  - Loss of consumer trust.
  - Legal penalties and regulatory fines. Financial losses and damage to reputation.

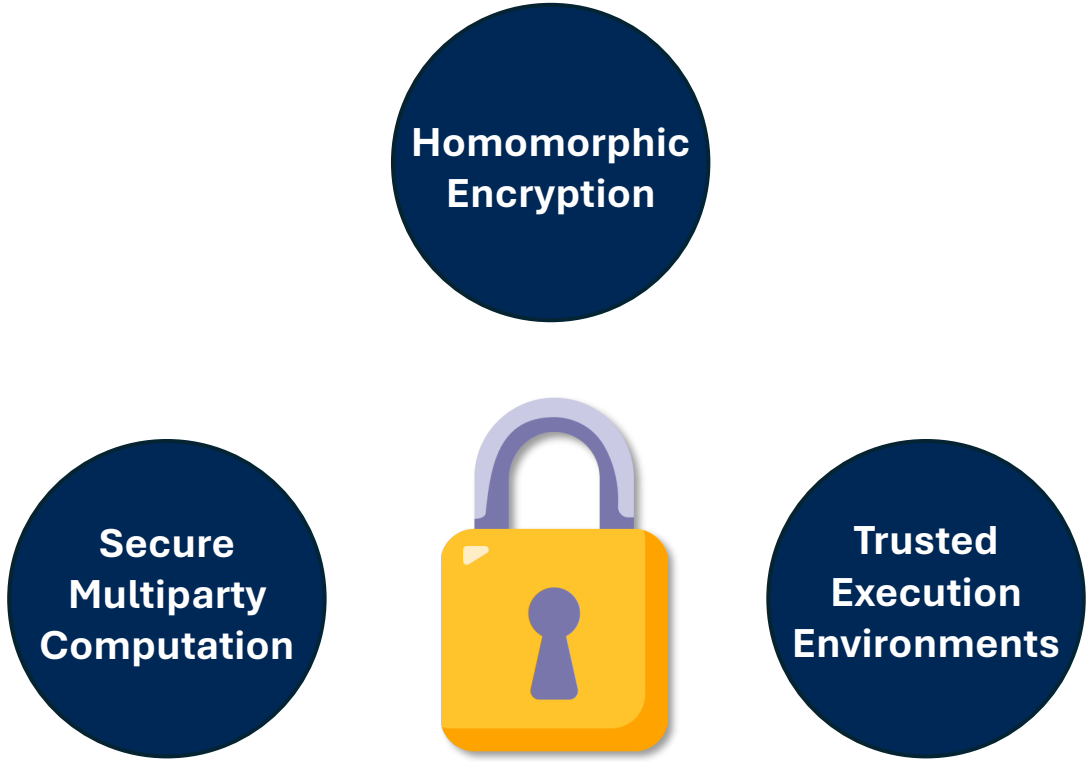


Schematic

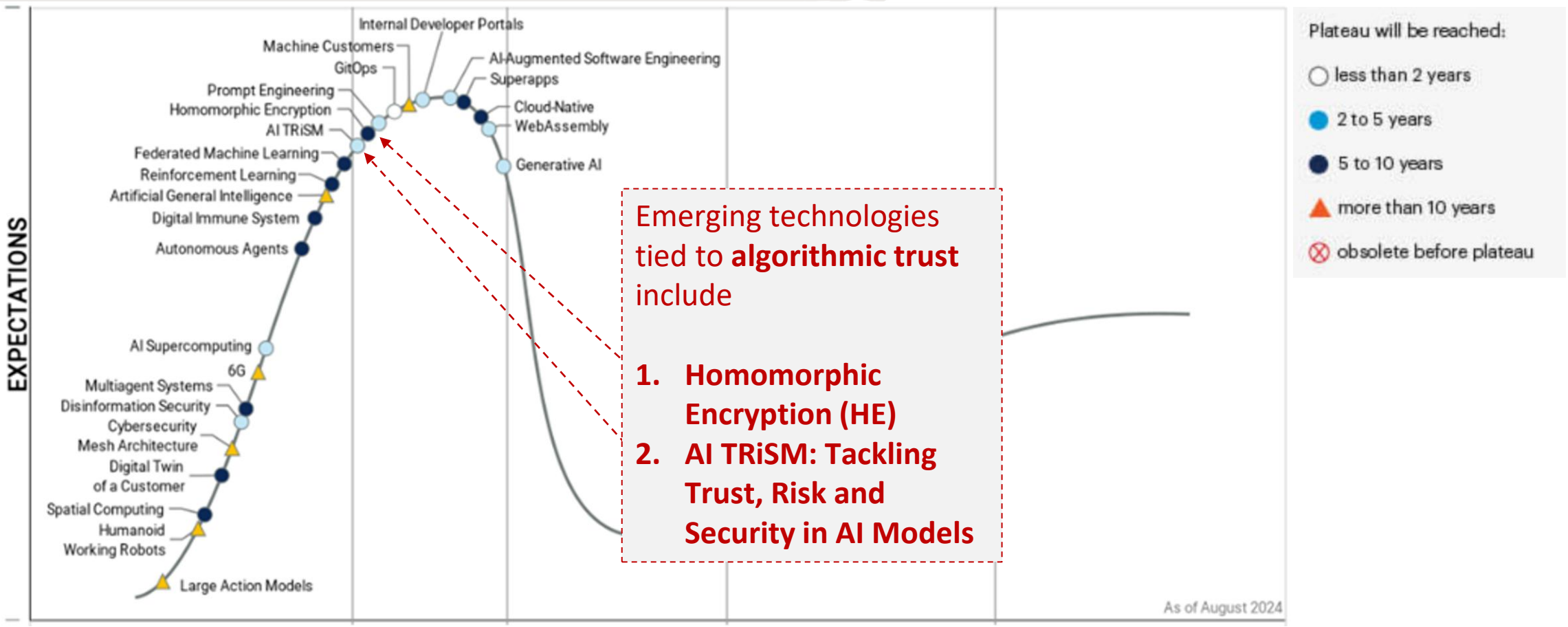
The Role of Technology in Privacy

- **Encryption:** Protects data by converting it into a code; essential for secure storage and transmission.
- **Blockchain:** Provides decentralized data management, enhancing transparency and security.
- **AI in Privacy:** Uses machine learning to detect anomalies and potential breaches in real time.
- **Privacy-Preserving Computation:** Techniques that allow data analysis without exposing raw data.

Three Important Privacy-Preserving Computation Techniques



Gartner Hype Cycle for Emerging Technologies, 2024



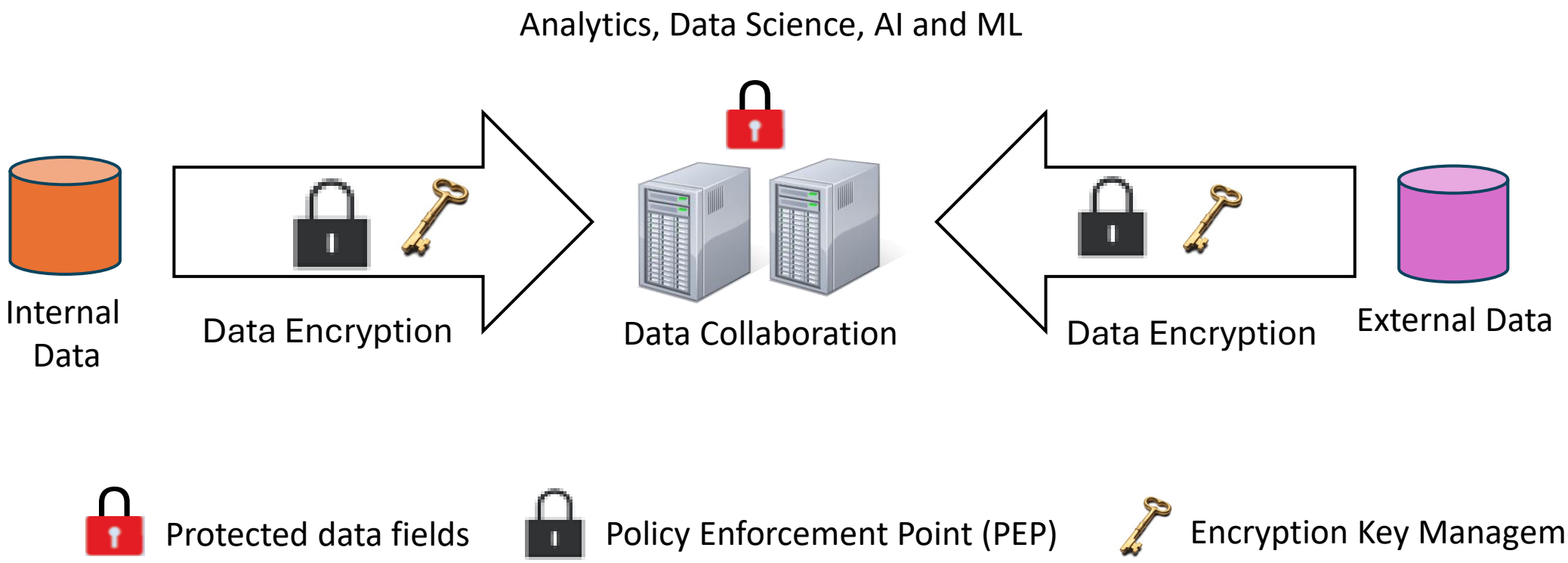
Emerging technologies tied to **algorithmic trust** include

1. **Homomorphic Encryption (HE)**
2. **AI TRiSM: Tackling Trust, Risk and Security in AI Models**

HE vs MPC vs TEE

	HE	MPC	TEE
Performance	Compute-bound	Network-bound	Close to plaintext
Privacy	Encryption	Encryption / Non-collusion	Trusted Hardware
Non-interactive	✓	✗	✓
Cryptographic Security	✓	✓	✗ e.g. Side-Channel Attacks

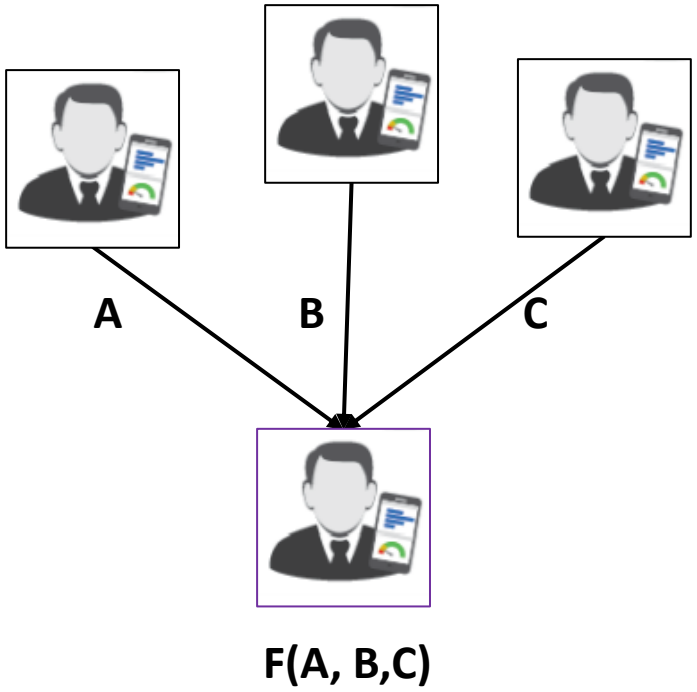
Increased need for data analytics drives requirements.



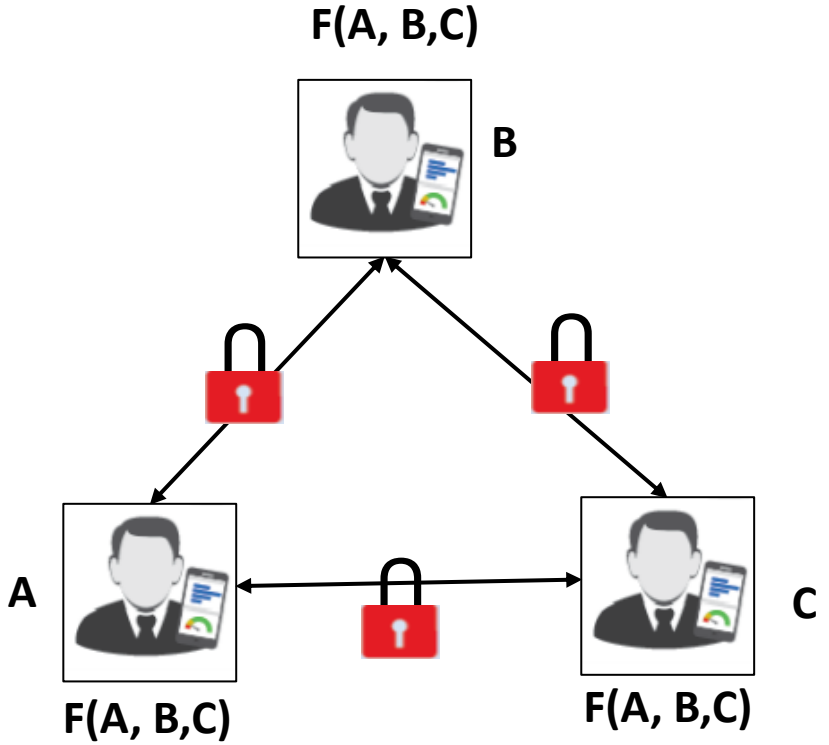
Secure Multi-Party Computation (SMPC)

Private multi-party machine learning with SMPC

Using SMPC, different parties send encrypted messages to each other, and obtain the model **F(A,B,C)** they wanted to compute without revealing their own private input, and without the need for a trusted central authority.



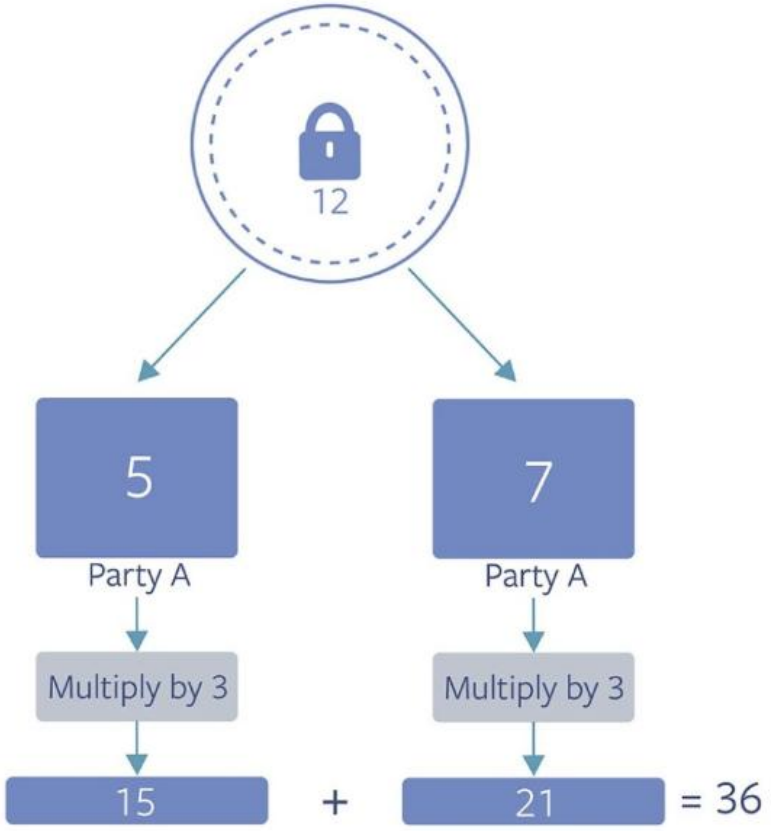
The data of A, B, and C was leaked to the central server.



Secure Multi-Party Computation

Protected data fields

Example of Secure Multi-party Computation: two-party secret addition #1



Secure data point.
(Not Shared with Party A or Party B.)

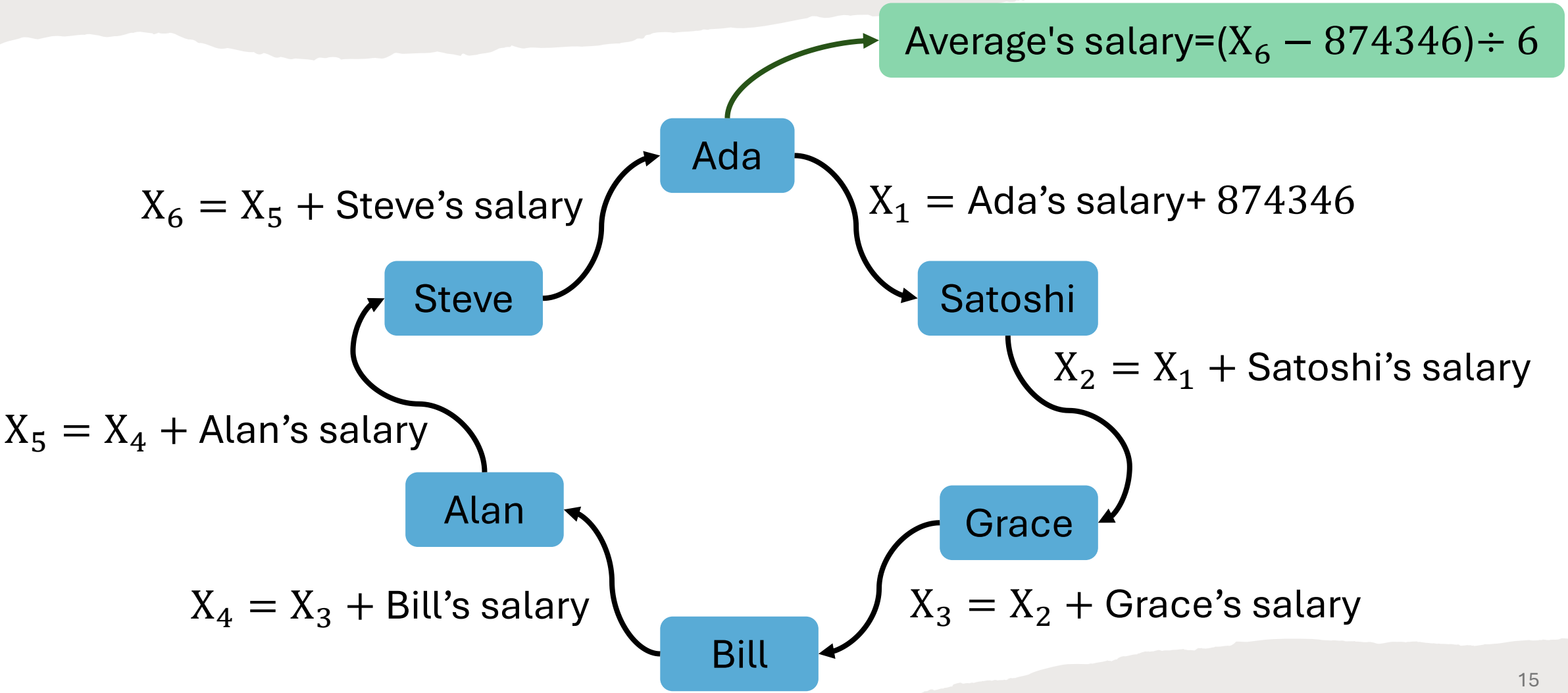
Party A and Party B are each given a number, but neither can use it to learn the secure data point (12).

Party A and Party B can each perform calculations on their number. The results can be combined to perform the calculation (12 x 3).

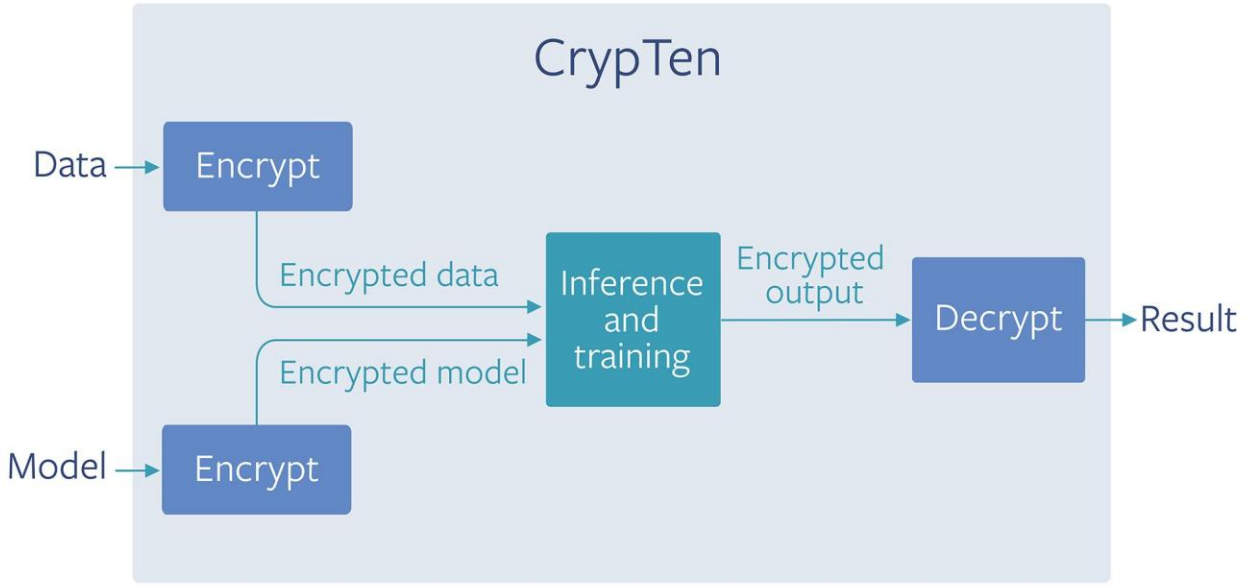
https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/

14

Example of Secure Multi-party Computation: Average Salary #1



Facebook's SMPC Project (CrypTen)




```
x = torch.tensor([1, 2, 3]
y = torch.tensor([4, 5, 6])
z = x + y
```

```
x_enc = crypten.cryptensor([1, 2, 3])
y_enc = crypten.cryptensor([4, 5, 6])
z_enc = x_enc + y_enc
```

https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/

CrypTen's Design Principles

- CrypTen interoperates via ONNX to import PyTorch models for secure computation, runs on PyTorch tensor ops, and uses Gloo/NCCL for high-performance multi-party communication.

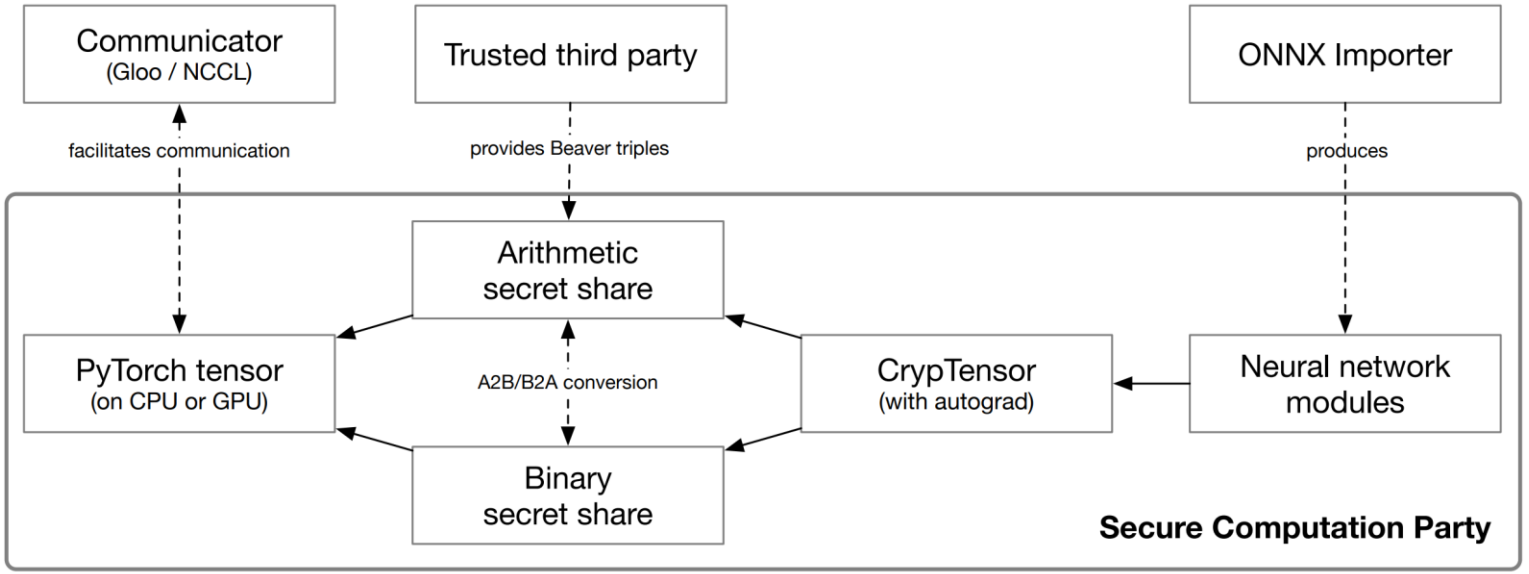


Figure 1: High-level overview of the design of CrypTen. See text in Section 4 for details.

CrypTen: Secure Multi-Party Computation Meets Machine Learning (NeurIPS 2021)

Example of secret-sharing tensors, revealing tensors, and private addition in CrypTen.

```
1
2  import sys
3  import os
4  import crypten
5  import torch
6  from crypten.config import cfg
7
8
9  rank = sys.argv[1]
10 os.environ["RANK"] = str(rank)
11 os.environ["WORLD_SIZE"] = str(2)
12 os.environ["MASTER_ADDR"] = '127.0.0.1'
13 os.environ["MASTER_PORT"] = "30007"
14 os.environ["RENDEZVOUS"] = "env://"
15 os.environ["GLOO_SOCKET_IFNAME"] = 'ens81np0'
```

```
17 crypten.init()
18 cfg.communicator.verbose = True
19 commInit = crypten.communicator.get().get_communication_stats()
20
21
22 x = torch.tensor([1.0, 2.0, 3.0])
23 x_enc = crypten.cryptensor(x,src=1) # encrypt
24 print("x_enc:",x_enc)
25 x_dec = x_enc.get_plain_text() # decrypt
26 y = torch.tensor([2.0, 3.0, 4.0])
27 y_enc = crypten.cryptensor(y,src=0)
28 print("y_enc:",y_enc)
29 sum_xy = x_enc + y_enc # add encrypted tensors
30 sum_xy_dec = sum_xy.get_plain_text() # decrypt sum
31 print("sum:",sum_xy_dec)
```

範例程式

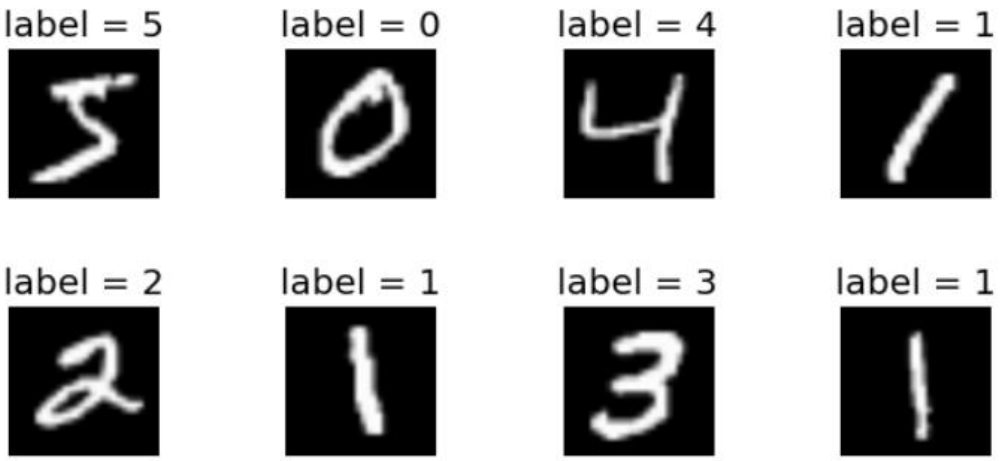
Example of secret-sharing tensors, revealing tensors, and private addition in CrypTen.

```
(CrypTen-env) davidgu@inventec-2:~/MPC_Experiments$ python mpc_vector_add.py 0
x_enc: MPCTensor(
        _tensor=tensor([ 4188819018133653156, -2628303871101976127, -71980814803400215691])
        plain_text=HIDDEN
        ptype=ptype.arithmetic
)
y_enc: MPCTensor(
        _tensor=tensor([ -6203488445605288601,  4484940860805597721, -63503082777172242151])
        plain_text=HIDDEN
        ptype=ptype.arithmetic
)
sum: tensor([3., 5., 7.])
```

```
(CrypTen-env) davidgu@inventec-2:~/MPC_Experiments$ python mpc_vector_add.py 1
x_enc: MPCTensor(
        _tensor=tensor([-4188819018133587620,  2628303871102107199,  71980814803402181771])
        plain_text=HIDDEN
        ptype=ptype.arithmetic
)
y_enc: MPCTensor(
        _tensor=tensor([ 6203488445606599321, -4484940860803631641, 63503082777174863591])
        plain_text=HIDDEN
        ptype=ptype.arithmetic
)
sum: tensor([3., 5., 7.])
```

Example MNIST Training in CrypTen

```
 3 import os, sys, time
 4 import torch, crypten
 5 import torch.nn.functional as F
 6 from torchvision import datasets, transforms
 7
 8 import crypten
 9 import crypten.nn as crynn
10 import crypten.optim as optim
11 from crypten.config import cfg
```




```
53 rank_cli = int(sys.argv[1])
54 os.environ.setdefault("RANK", str(rank_cli))
55 os.environ.setdefault("WORLD_SIZE", "2")
56 os.environ.setdefault("MASTER_ADDR", 'xxx.xxx.xxx.xxx')
57 os.environ.setdefault("MASTER_PORT", "30011")
58 os.environ.setdefault("RENDEZVOUS", "env://")
59 os.environ.setdefault("GLOO_SOCKET_IFNAME", os.environ.get("GLOO_SOCKET_IFNAME", 'ensxxx'))
```

範例程式

Example MNIST Training in CrypTen.

```
115     # 加密資料（src=0 表示資料由第 0 方提供）
116     t0_enc = time.time()
117     Xtr_enc     = crypten.cryptensor(Xtr,     src=0)
118     ytr_oh_enc = crypten.cryptensor(ytr_oh, src=0)
119     Xte_enc     = crypten.cryptensor(Xte,     src=0)
120     yte_oh_enc = crypten.cryptensor(yte_oh, src=0)
121     t_enc = time.time() - t0_enc
122
123     # 模型
124     model_enc = crynn.Sequential(
125         crynn.Linear(sample_dim, hidden_dim),
126         crynn.ReLU(),
127         crynn.Linear(hidden_dim, num_classes),
128     ).encrypt(src=1)
129
130     criterion = crynn.CrossEntropyLoss()
131     optimizer = optim.SGD(model_enc.parameters(), lr=lr, momentum=momentum)
132
```

範例程式

Example MNIST Training in CrypTen.

```
(CrypTen-env) davidgu@inventec-2:~/MPC_Experiments/P0/MNIST$ python mnist_crypten_mpc_env.py 0
[Gloo] Rank 0 is connected to 1 peer ranks. Expected number of connected peer ranks is : 1
[Gloo] Rank 0 is connected to 1 peer ranks. Expected number of connected peer ranks is : 1
[Gloo] Rank 0 is connected to 1 peer ranks. Expected number of connected peer ranks is : 1
[Gloo] Rank 0 is connected to 1 peer ranks. Expected number of connected peer ranks is : 1
# Setup
- data_load: 0.117284 s
- encrypt  : 0.396756 s

## Epoch 01
- total : 126.012641 s
- compute≈: 58.226328 s (46.21%)
- comm   : 67.786313 s (53.79%)
- avg loss: 0.7595

## Epoch 02
- total : 126.216517 s
- compute≈: 58.149225 s (46.07%)
- comm   : 68.067292 s (53.93%)
- avg loss: 0.3133

# Eval
- eval_total: 15.577256 s
- accuracy  : 90.76% on 5000 samples
```

範例程式

Example MNIST Training in CrypTen.

Network Bandwidth		1 Gb/s	100 Mb/s
Setup	Data load (s)	0.113	0.144
	Encrypt (s)	0.387	0.604
Epoch 01	Total (s)	78.089	127.412
	Compute (s)	50.8127 (65.07%)	59.284 (46.53%)
	Comm (s)	27.276 (34.93%)	68.127 (53.47%)
	avg loss	0.7601	0.7590
Epoch 02	Total (s)	77.533	127.553
	Compute (s)	50.323	58.284
	Comm (s)	27.21	69.268
	avg loss	0.3132	0.3133
Eval	eval_total (s)	2.993	15.57
	accuracy	90.80% (5000 samples)	90.80% (5000 samples)

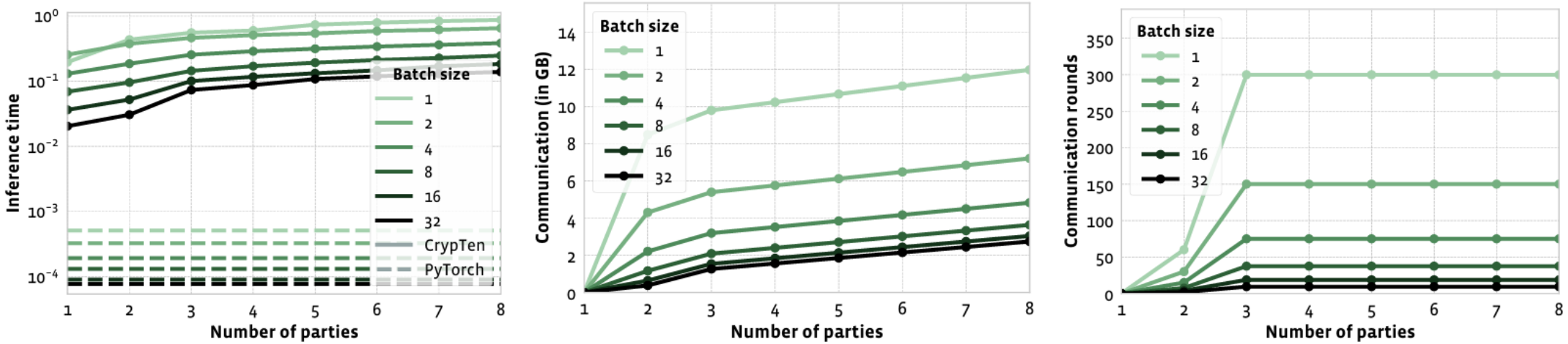
Example: Private Inference on GPU Using a Secret-Shared ResNet-18 Model in CrypTen

```
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms

# download and set up ImageNet dataset:
transform = transforms.ToTensor()
dataset = datasets.ImageNet(
    imagenet_folder, transform=transform,
)

# secret share pre-trained ResNet-18 on GPU:
model = models.resnet18(pretrained=True)
model_enc = crypten.nn.from_pytorch(
    model, dataset[0],
).encrypt().cuda()

# perform inference on secret-shared images:
for image in dataset:
    image_enc = crypten.cryptensor(image).cuda()
    output_enc = model_enc(image_enc)
    output = output_enc.get_plain_text()
```

https://github.com/facebookresearch/CrypTen

Example using neural networks in CrypTen.

- Dataset (concise): Yelp reviews sentiment classification.
- Model (concise): 32-dim word embeddings → linear layer (binary output).



Benchmarks for inference with text-sentiment classification model on GPUs in CRypten and PyTorch.

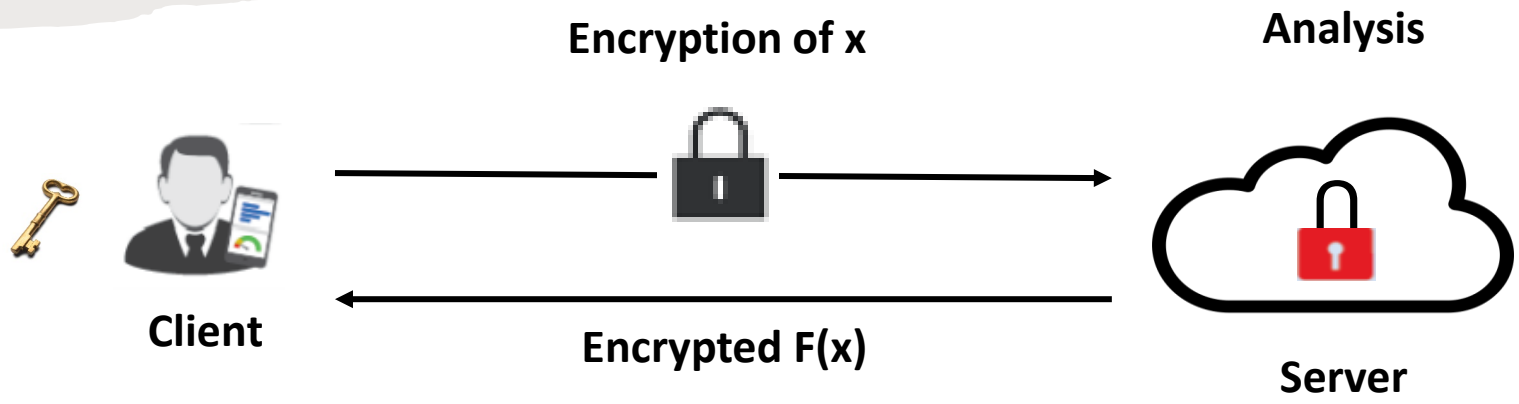
- Left: Average wall-clock time per sample (in seconds).
- Middle: Number of bytes communicated per sample, per party (in GB).
- Right: Number of communication rounds per sample.

CrypTen: Secure Multi-Party Computation Meets Machine Learning (NeurIPS 2021)

Homomorphic Encryption (HE)

HE depicted in a client-server model

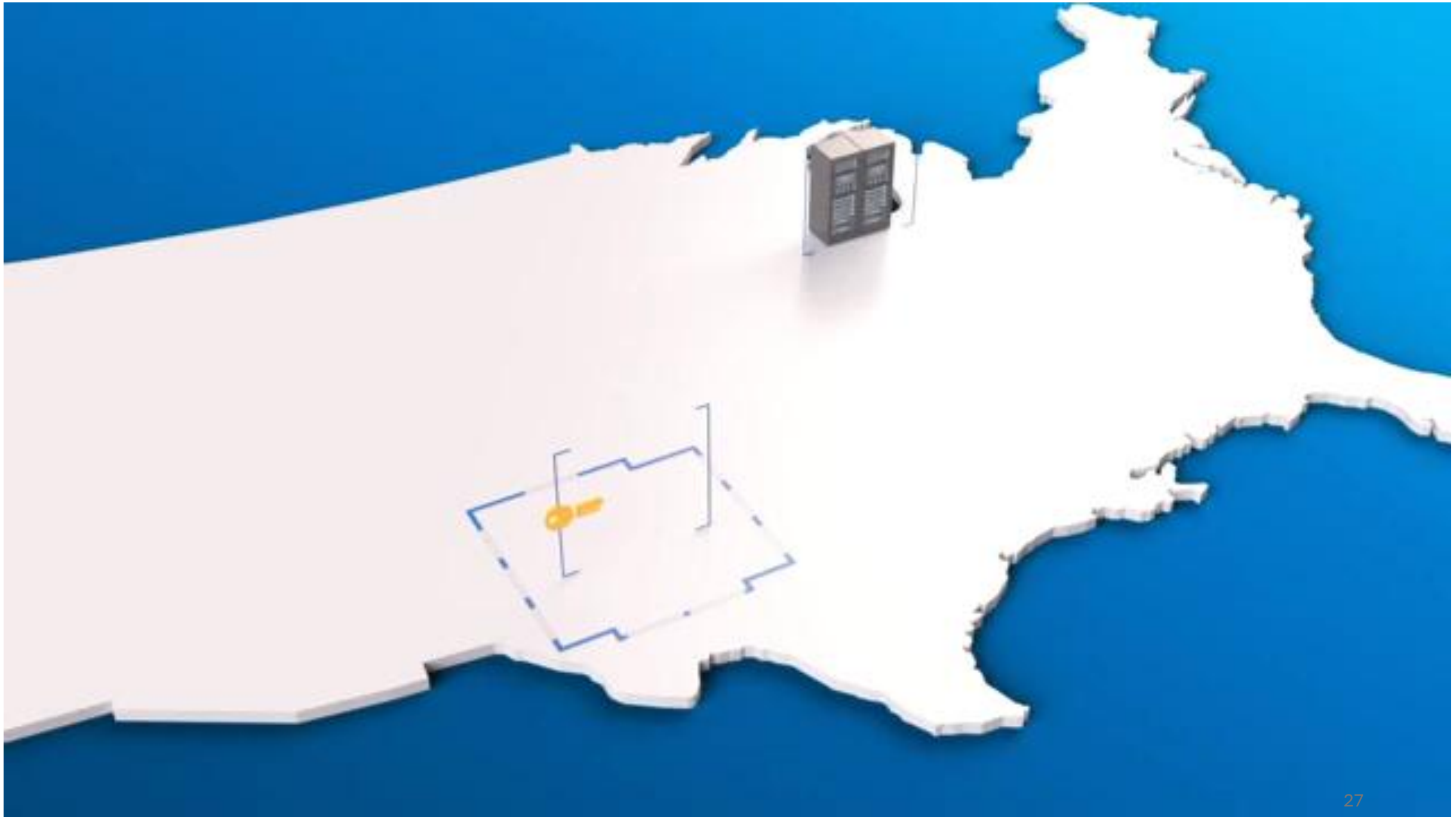
- The client sends encrypted data to a server, where a **specific analysis is performed on the encrypted** data, without decrypting that data.
- The encrypted result is then sent to **the client, who can decrypt it to obtain the result** of the analysis they wished to outsource.



Encryption Key Management

Policy Enforcement Point (PEP)

Protected data fields



Well-Known HE Schemes

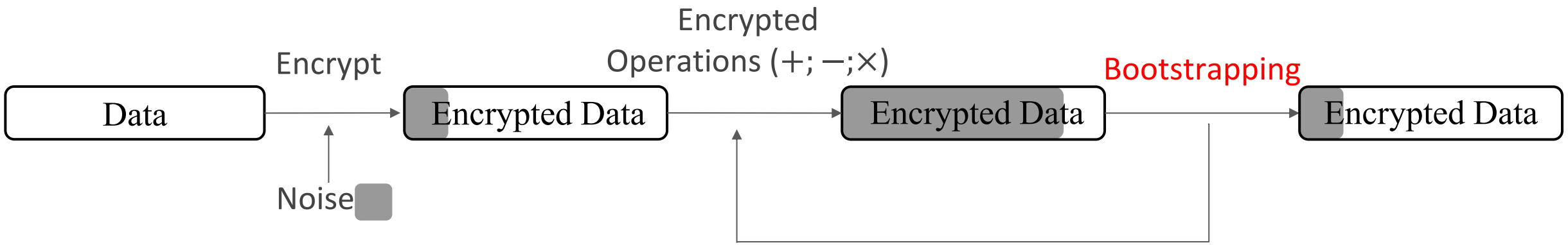
- Modular (Exact) Integer Arithmetic: **BGV / BFV**
  - Plaintext data represented as **vectors modulo a plaintext modulus "p"**
  - Computations expressed as **vectors arithmetic mod p**
- Binary FHE : **FHEW/ TFHE**
  - Plaintext represented as **integers/Boolean values**
  - Supports evaluation of arbitrary functions using Look-Up Tables (LUTs)
- Approximate Number Arithmetic: **CKKS**
  - Plaintext data represented as **vectors of real numbers**
  - Compute model similar to **floating-point arithmetic** but dealing with fixed-point numbers

Well-Known HE Schemes

- **FHEW/ TFHE:** Encrypt bits and perform logical AND, OR, XOR operations on the ciphertexts.
  - ✓ 0 AND 1 → 0, 0 OR 1 → 1, 1 XOR 1 → 0
- **BGV / BFV:** Encrypt 8-bit unsigned integers (between 0 and 255) and perform addition and multiplication modulo 256.
  - ✓ 128 + 128 → 0, 2 * 129 → 2
- **CKKS:** Encrypt fixed-point numbers and perform addition and multiplication with the result rounded to a fixed precision, for instance, two digits after the decimal point.
  - ✓ 12.42 + 1.34 → 13.76, 2.23 + 5.19 → 11.57
- Different homomorphic encryption schemes support different plaintext types and different operations on them.

Fully Homomorphic Encryption (FHE)

- Small random noise is added into the encrypted data to ensure security in homomorphic encryption. However, this noise will accumulate during operations.
- Bootstrapping is an operation to clean up the accumulated error in ciphertext.
- Schemes that support bootstrapping are referred to as Fully Homomorphic Encryption (FHE) .



Bootstrapping: ~10,000× Slower Than Encrypted (+/−/×)

Bootstrapping is bottleneck!

Well-Known HE Lbraries

Library/Scheme	BGV	BFV	CKKS	CKKS Bootstr.	FHEW	TFHE
FHEW					✓	
HEAAN			✓	✓		
HELib	✓		✓			
Lattigo	✓	✓	✓	✓		
OpenFHE (PALISADE**)**	✓	✓	✓	✓	✓	✓
SEAL	✓	✓	✓			
TenSeal		✓	✓			
TFHE-rs						✓
TFHE-lib						✓

Selecting Security Parameters

- The ciphertext dimension (degree of polynomial) should be chosen according to the security tables published at Homomorphic Encryption Standardization

distribution	n	security level	logq	uSVP	dec	dual
(-1, 1)	1024	128	27	131.6	160.2	138.7
		192	19	193.0	259.5	207.7
		256	14	265.6	406.4	293.8
	2048	128	54	129.7	144.4	134.2
		192	37	197.5	233.0	207.8
		256	29	259.1	321.7	273.5
	4096	128	109	128.1	134.9	129.9
		192	75	194.7	212.2	198.5
		256	58	260.4	292.6	270.1

Source: https://openfheorg.github.io/aaai-2024-lab-materials/

TenSEAL

- TenSEAL is a library for doing homomorphic encryption **operations on tensors**, built on top of Microsoft SEAL. It provides ease of use through a Python API, while preserving efficiency by implementing most of its operations using C++.
- Features
  - 🔑 Encryption/Decryption of vectors of integers using BFV
  - 🔑 Encryption/Decryption of vectors of real numbers using CKKS
  - 🔥 Element-wise addition, subtraction and multiplication of **encrypted-encrypted vectors** and **encrypted-plain vectors**
  - 🌀 Dot product and vector-matrix multiplication
- Install (Linux, MacOS, Windows)
  - $ pip install tenseal

A Simple BFV Example in TenSEAL

```python
import tenseal as ts
import time

context = ts.context(
    ts.SCHEME_TYPE.BFV,
    poly_modulus_degree=8192,
    plain_modulus=786433,
)
context.generate_galois_keys()

# 向量資料（整數）
v1 = [0, 1, 2, 3, 4]
v2 = [4, 3, 2, 1, 0]

start = time.perf_counter()
# 加密向量（BFV）
enc_v1 = ts.bfv_vector(context, v1)
enc_v2 = ts.bfv_vector(context, v2)
end = time.perf_counter()
print("Encryption Time (s):",end-start)
```

```python
start = time.perf_counter()
# ---- 逐元素加法
res = enc_v1 + enc_v2
end = time.perf_counter()
print("enc_v1 + enc_v2 time (s):",end-start)
print("enc_v1 + enc_v2=",res.decrypt())   # =>


start = time.perf_counter()
# ---- 內積（兩種寫法）
# 寫法1：逐元素乘法後做 slot sum
dot_enc = (enc_v1 * enc_v2).sum()  # enc_v1.dot
end = time.perf_counter()
print("enc_v1.dot(enc_v2) time (s):",end-start)
print("enc_v1.dot(enc_v2)=",dot_enc.decrypt())
```

```
(CrypTen-env) davidgu@inventec-2:~$ python BFV.py
Encryption Time (s): 0.012799451127648354
enc_v1 + enc_v2 time (s): 0.0010970258153975 01
enc_v1 + enc_v2= [4, 4, 4, 4, 4]
enc_v1.dot(enc_v2) time (s): 0.049883141182363 03
enc_v1.dot(enc_v2)= [10]
```

範例程式

A Simple CKKS Example in TenSEAL

```python
import tenseal as ts
import time
# Setup TenSEAL context
context = ts.context(
            ts.SCHEME_TYPE.CKKS,
            poly_modulus_degree=8192,
            coeff_mod_bit_sizes=[60, 40, 40, 60]
          )
context.generate_galois_keys()
context.global_scale = 2**40

v1 = [0, 1, 2, 3, 4]
v2 = [4, 3, 2, 1, 0]

start = time.perf_counter()
# encrypted vectors
enc_v1 = ts.ckks_vector(context, v1)
enc_v2 = ts.ckks_vector(context, v2)
end = time.perf_counter()
print("Encryption Time (s):",end-start)
```

```python
start = time.perf_counter()
result = enc_v1 + enc_v2
end = time.perf_counter()
print("enc_v1 + enc_v2 time (s):",end-start)
print("enc_v1 + enc_v2=",result.decrypt())  # ~

start = time.perf_counter()
result = enc_v1.dot(enc_v2)
end = time.perf_counter()
print("enc_v1.dot(enc_v2) time (s):",end-start)
print("enc_v1.dot(enc_v2)=",result.decrypt())
```

```
(CrypTen-env) davidgu@inventec-2:~$ python CKKS.py
Encryption Time (s): 0.01568648498505354
enc_v1 + enc_v2 time (s): 0.0004481417126953602
enc_v1 + enc_v2= [4.00000000206549, 4.000000001071652, 4.000000000099898
, 4.000000002884829, 4.000000000801553]
enc_v1.dot(enc_v2) time (s): 0.015246727969497442
enc_v1.dot(enc_v2)= [10.000001000988302]
```

範例程式

OpenFHE

- OpenFHE is a full-featured library for **fully homomorphic encryption**, offering multiple schemes for integers, real numbers, and boolean gates. It exposes a C++ core with Python bindings for rapid prototyping.
- Features
  - 🧮 BFV / BGV: exact-integer vector encryption & arithmetic (add/mul, rotations, batching/packing)
  - 🔢 CKKS: approximate real-number vectors (add/mul, rescale, rotate, conjugate)
  - 🔌 BinFHE (FHEW/TFHE-style): boolean gates (AND/OR/XOR/NOT), comparisons, programmable bootstrapping
  - 🔁 Bootstrapping & key-switching: depth extension for CKKS/BFV/BGV; automatic in BinFHE gates
- Install (Linux)
  - $ pip install openfhe

A Simple BinFHE (FHEW/TFHE-style) Example in OpenFHE

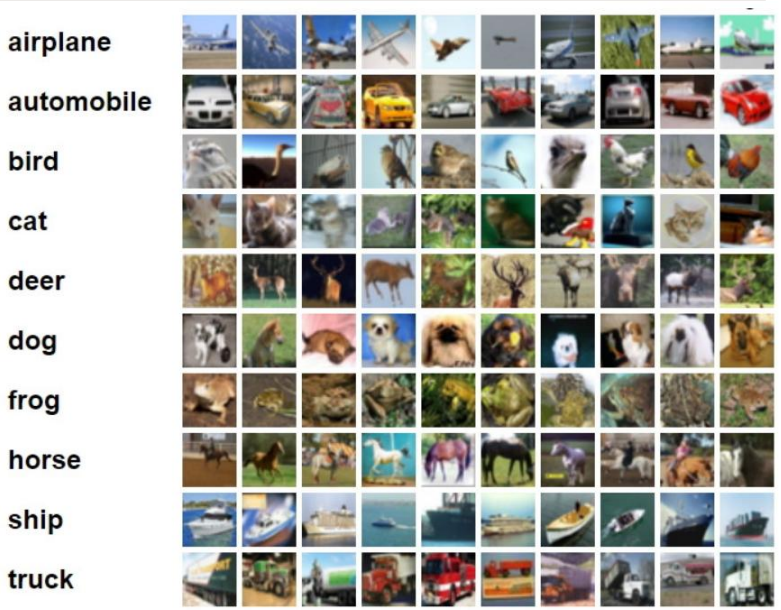
```python
from openfhe import *
import time
# BinFHE context
cc = BinFHEContext()
cc.GenerateBinFHEContext(BINFHE_PARAMSET.STD128, BINFHE_METHOD.GINX)

# Keys
sk = cc.KeyGen()
cc.BTKeyGen(sk)

def dec(ct):
    return cc.Decrypt(sk, ct)

start = time.perf_counter()
# Encrypt independent bits (p=4；0/1 為 Z4 裡的 0/1)
c1a = cc.Encrypt(sk, 1, BINFHE_OUTPUT.BOOTSTRAPPED)
c1b = cc.Encrypt(sk, 1, BINFHE_OUTPUT.BOOTSTRAPPED)
c0a = cc.Encrypt(sk, 0, BINFHE_OUTPUT.BOOTSTRAPPED)
c0b = cc.Encrypt(sk, 0, BINFHE_OUTPUT.BOOTSTRAPPED)
end = time.perf_counter()
print("Encryption Time (s):",end-start)
```

```python
start = time.perf_counter()
# Boolean gates（輸入需獨立，且同一 p）
cand = cc.EvalBinGate(BINGATE.AND, c1a, c1b)
cor  = cc.EvalBinGate(BINGATE.OR,  c1a, c0a)
cxor = cc.EvalBinGate(BINGATE.XOR, c1a, c0a)
cnot = cc.EvalNOT(c1a)
end = time.perf_counter()
print("Compute Time (s):",end-start)


print("AND(1,1) =", dec(cand))
print("OR(1,0)  =", dec(cor))
print("XOR(1,0) =", dec(cxor))
print("NOT(1)   =", dec(cnot))
```

```
(CrypTen-env) davidgu@inventec-2:~$ python FHEW.py
Encryption Time (s): 0.0003447178751230 24
Compute Time (s): 0.2824556641280651
AND(1,1) = 1
OR(1,0)  = 1
XOR(1,0) = 1
NOT(1)   = 0
```

範例程式

FHE-DNN VS SMPC-DNN



- Model Architecture: Vgg9
- Dataset: Cifar-10
- Batch Size: 1

	Plaintext inference (pytorch)	MPC-DNN with CrypTen	FHE-DNN with FHEW/TFHE
Total Time	0.004 s	1.2092 s	86 s
Experimental Setup	• One machines equipped with Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz (36 cores / 72 threads) with 755GB of memory	• Two machines equipped with Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz (36 cores / 72 threads) with 755GB of memory • Comm. Bandwidth: 31.1 Gbps	• AMD Ryzen Threadripper PRO 5975WX 32-Cores with 252GB of memory and **4 NVIDIA RTX4090 GPU**

What is a TEE? (and how it relates to Confidential Computing)

- TEE (Trusted Execution Environment): a hardware-isolated area in a CPU that runs code and handles data with confidentiality and integrity guarantees.
- Typical capabilities: isolated execution, sealed storage, platform integrity/secure boot, remote attestation.
- Confidential Computing (CC): protects data in use by running computations inside a hardware-based, attested TEE; builds a broader ecosystem of standards and deployments.
- Representative platforms: Arm TrustZone/CCA, Intel SGX/TDX, AMD SEV/SEV-SNP, RISC-V Keystone.

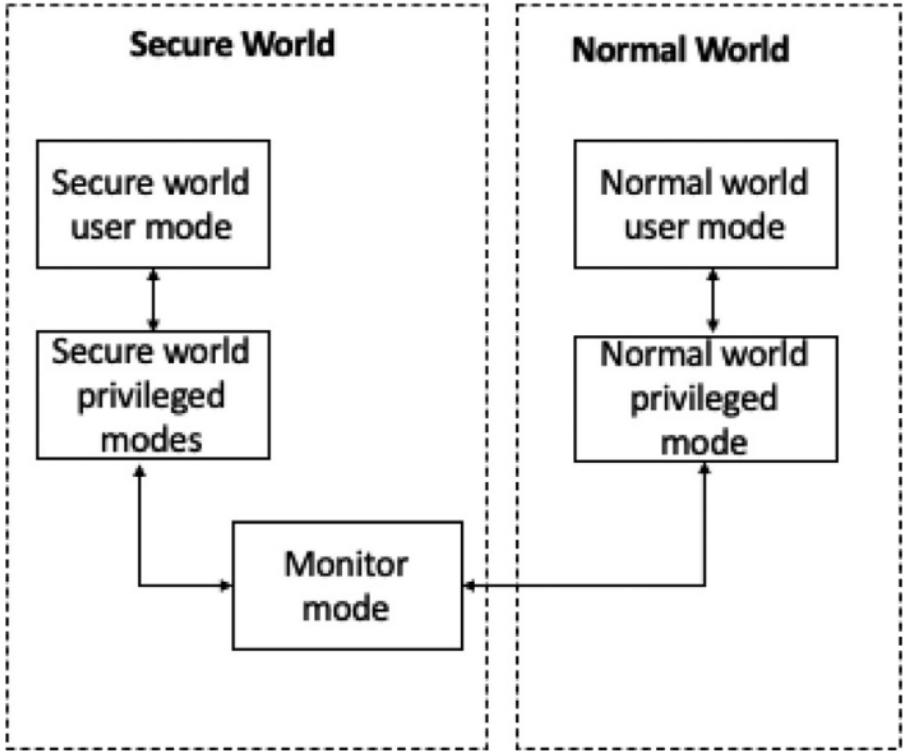


Fig. 1. Relationship between the Secure World and the Normal World.

A survey on the (in)security of trusted execution environments — 2023 — Computers & Security

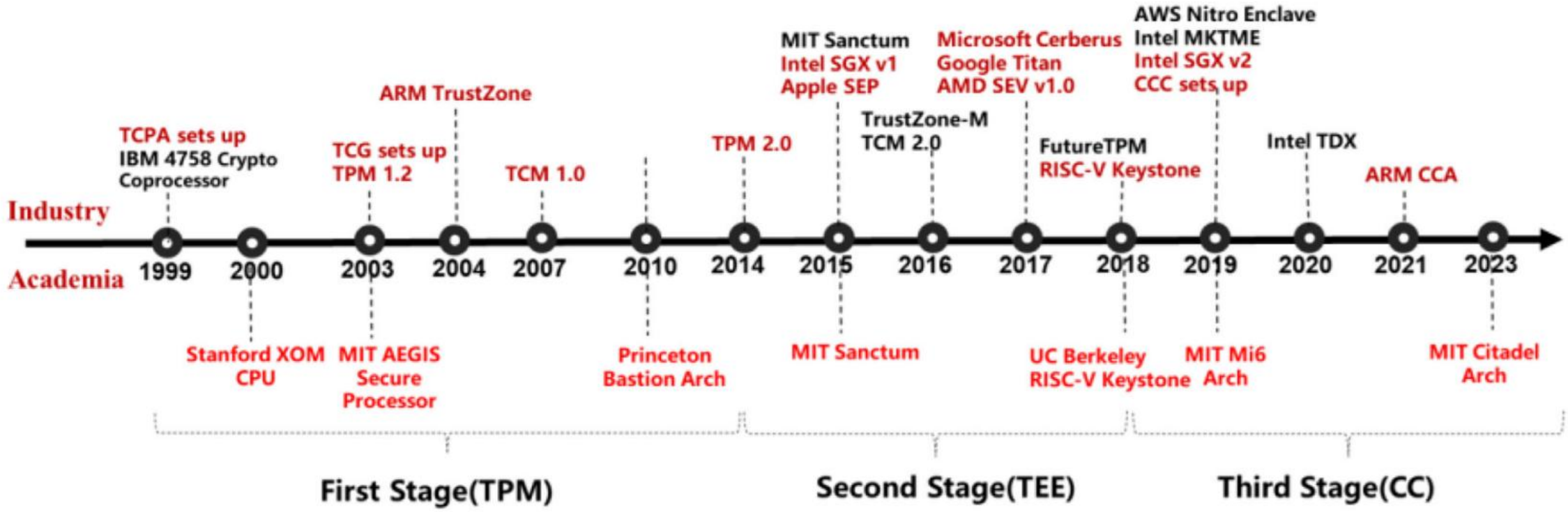


FIGURE 1 The development history of confidential computing.

History & Evolution (TPM → Enclaves → VM-based TEEs)

- Stage 1 (2003–2014): TPM/crypto co-processors establish roots of trust and secure storage.
- Stage 2 (2015–2018): CPU TEEs emerge—Intel SGX (process-level enclave), AMD SEV (VM memory protection), TrustZone-M, early RISC-V designs.
- Stage 3 (2019–present): Cloud-friendly VM-based TEEs become mainstream: SEV-SNP, Intel TDX, Arm CCA; CC grows via industry standards and deployments.
- Why the shift: VM-based TEEs bring compatibility and lift-and-shift for existing workloads; enclaves keep smaller TCB for narrow, high-sensitivity code.

Architecture & Platforms (Process- vs VM-level; TrustZone anatomy)

- Process-level (e.g., SGX): smallest TCB but requires app changes; syscall/page-fault side channels are key engineering concerns.
- VM-level (e.g., TDX/SEV-SNP/CCA): larger TCB but high compatibility with cloud stacks and easier lift-and-shift.
- Arm TrustZone: two realms (Secure World and Normal World) with SMC/monitor mode switches; shared-memory patterns (FAST/YIELD) for inter-world communication.
- Ecosystem spans commercial TEE OSes (Trusty, QSEE, Kinibi) and open-source (OP-TEE); privileged vs non-privileged TEEs.

	Intel SGX	Intel TDX	AMD SEV	ARM TZ	ARM CCA	KeyStone
Arch	X86	X86	X86	ARM	ARM	RISC V
ISA Release Time	Intel 2013	Intel 2021	AMD 2016	ARM 2005	ARM 2022	RISC V 2017
CPU Model	6th Intel CPU and above	Intel 4th Gen Xeon	AMD EPYC	ARMv6-A、ARMv8-M	ARMv9-A	Xilinx Artix-7 SiFive E31, U54
System Security Level	Process	VM	VM	Process/secure OS	VM	Process/VM
Isolation Compartment	Multi-enclave	Multi-VM	Multi-VM	Single TEE	Multi-realm	Multi-Enclave
Data Integrity	Support	Support	Support (Nest Page Table)	Support(Isolation Partition)	Support(Isolation Partition)	Support(PMP)
Data Confidentiality	Support	Support	Support	Not Support	Not Support	Support(PMP)
Code Integrity	Support	Support	Support (Nest Page Table)	Support(Isolation Partition)	Support(Isolation Partition)	Support
Code Confidentiality	Support (Mem Encryption)	Support (Mem Encryption)	Support (Mem Encryption)	Not Support	Not Support	Support(PMP)
Measurement Boot	Support	Support	Support	Support	Support	Support
Remote Attestation	Support	Support	Support	Support (fTPM Extension)	Support	Support
TCB (include CPU/SoC)	Firmware、SGX driver	Firmware、TDX module、Hype/VM	Firmware、Hype、guest OS	Secure OS	SM、RMM	SM、Runtime
Secure Mem Size	Max 128MB(SGX v1) Max 1TB(SGX v2)	Max Available Mem	Max Available Mem	16~64MB	Max Available Mem	Max Available Mem
Hardware Attack	Attack Surface	--	Attack Surface	Attack Surface	--	Attack Surface
Side Channel Attack	Large Attack Surface	--	Attack Surface	Attack Surface	Attack Surface	Partial defense
Software Attack	Attack Surface (OS、APP)	Attack Surface (VMM、OS、Process)	Attack Surface (VMM、OS、Process)	Attack Surface (OS、Process)	Attack Surface (OS、Process)	Attack Surface (OS、Process)
Application Compatibility	Modify App a lot	Do not Modify App	Do not Modify App	Modify App a lot	Modify App a little	Modify App a lot
Application Scenarios	PC/Cloud Server desktop-level	Cloud Server desktop/server/cloud-level	Cloud Server desktop/server/cloud-level	Embedded/Mobile Device mobile/desktop-level	Embedded/Cloud Server mobile/desktop/server-level	Embedded/Mobile Device mobile/desktop-level

FIGURE 2 Comparison among typical technology roadmaps of confidential computing.

Applications (Cloud, Keys/Identity, Data Collaboration, ML, Mobile/IoT)

- Cloud confidential VMs/containers: isolate tenant memory/state from host OS/hypervisor; establish trust via remote attestation before loading secrets.
- Keys & identity (e.g., Android KeyMaster/Keystore): private keys are generated, stored, and used inside the TEE.
- Data clean rooms / cross-organization analytics: decrypt/process only inside attested TEEs; export aggregated results.
- Confidential ML (training & inference): protect models and inputs in a TEE; combine with DP/MPC when needed.
- Mobile/IoT: payments/wallets, DRM, secure boot, trusted peripherals using TrustZone/CCA.

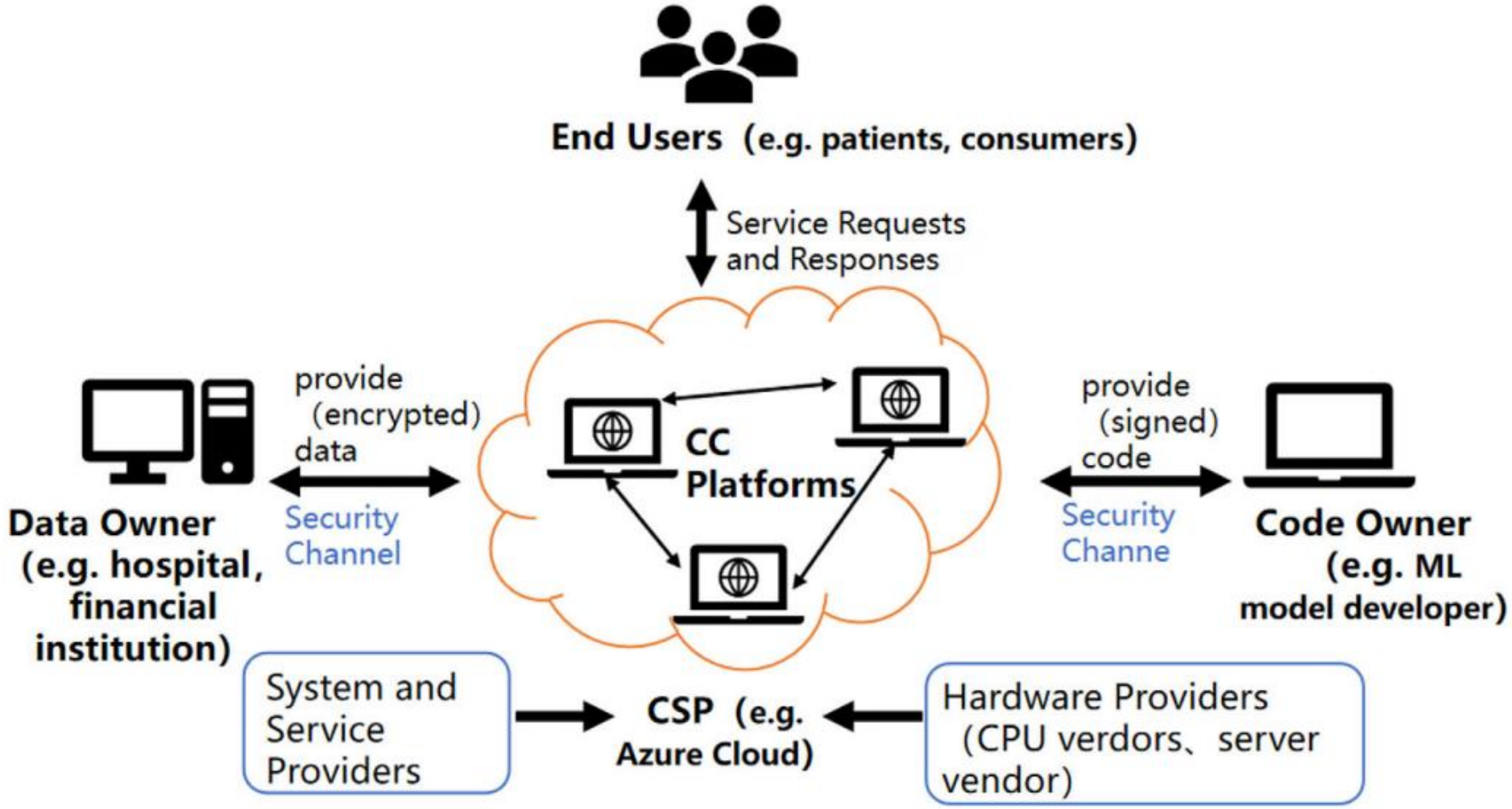


FIGURE 4 Typical application framework of confidential computing.

Security Landscape: Attacks & What Actually Works in Practice

- Attack taxonomy: software/system, side-channel, transient execution, fault injection.
- TrustZone case patterns: userland → driver (QSEECOM) → TA → TZ kernel chains; syscall hijacking; bootloader unlock; ROM extraction; BOOMERANG isolation bypass.
- Countermeasures: minimize TCB; data-oblivious/near-constant-time coding; strict RA policy (measurement/version/microcode; bind session keys; freshness/revocation); sealing with anti-rollback (monotonic counters); minimize SMC/I-O, validate parameters; patch & audit lifecycle.

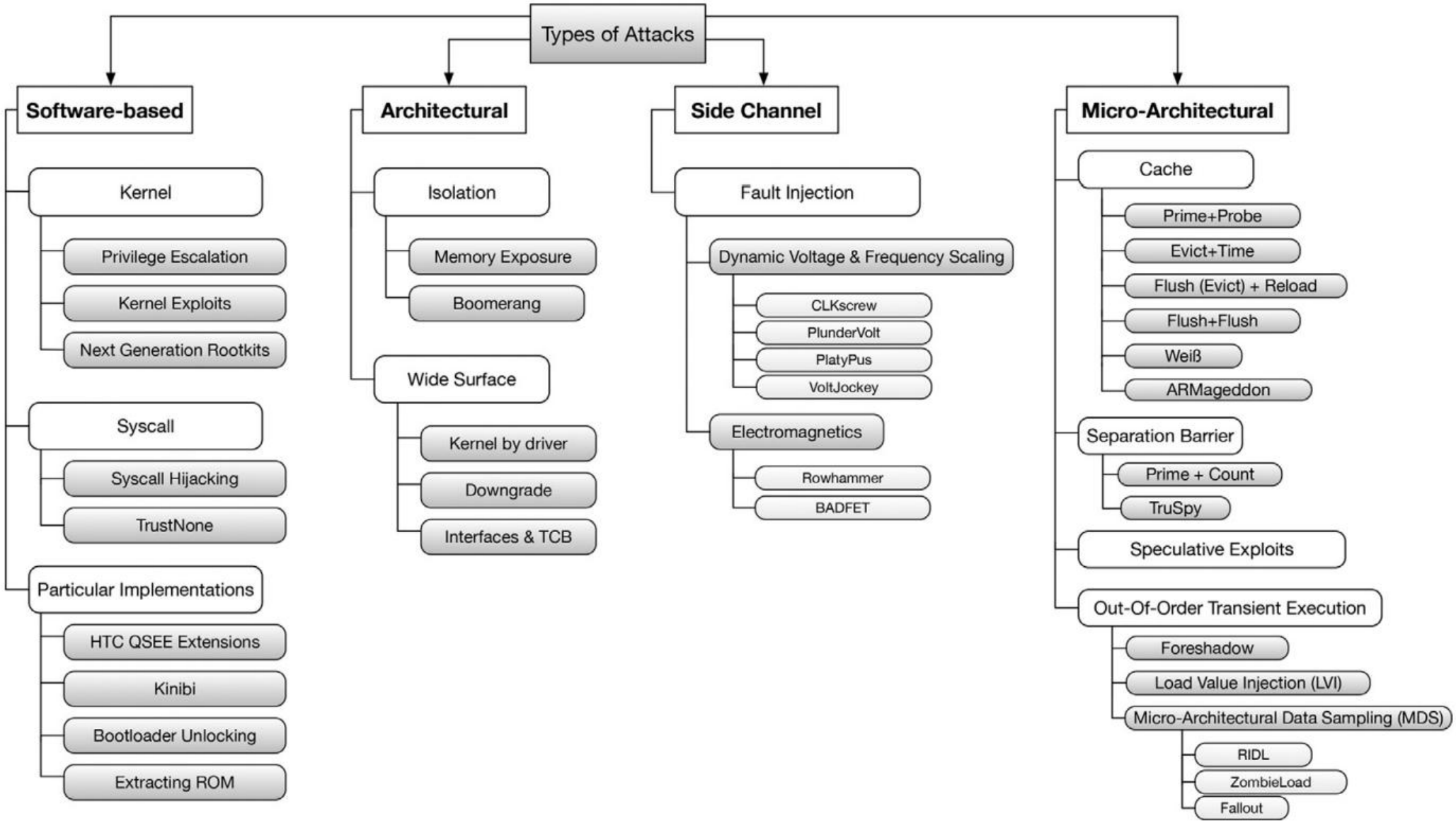


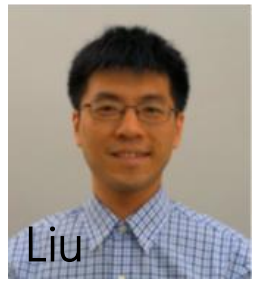
Fig. 5. Taxonomy of Attacks to TEE Implementations.

A survey on the (in)security of trusted execution environments — 2023 — Computers & Security

Our Publication

Collaborators

- Dr. Shih-Hao Hung
- Dr. Tu, I-Ping
- Dr. Wei-Chao Chen
- Dr. Feng-Hao Liu
- Dr. Ming-Ching Chang
- Dr. Chih-Fan Hsu
- PhD Candidate Tu-Te Ku
- Researcher Ming-Chien Ho
- Researcher Tzu-Li Liu
- Researcher Ming-Chien Ho



Papers

- Optimizing Encrypted Neural Networks: Model Design, Quantization and Fine-Tuning Using FHEW/TFHE. PoPETs 2025
- A Comprehensive Evaluation of Encrypted DNN Inference Methods. IEEE ISCAS 2025
- GPU acceleration for FHEW/TFHE bootstrapping. CHES 2025
- Invited paper: Efficient design of FHEW/TFHE bootstrapping implementation with scalable parameters. ICCAD 2024
- An Efficient CKKS-FHEW/TFHE Hybrid Encrypted Inference Framework. ESORICS Workshops 2023