COLUMBIA UNIVERSITY

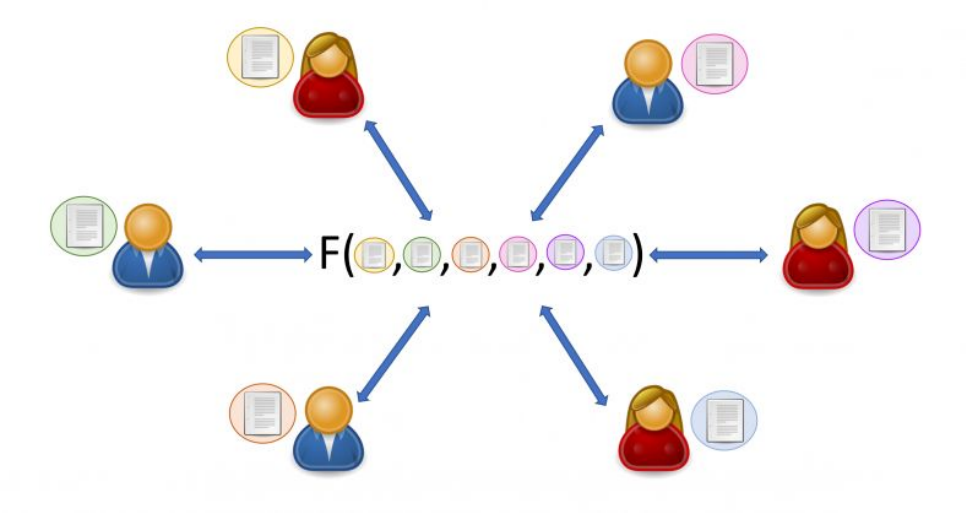
PPMLAC - High Performance Chipset Architecture for Secure Multi-Party Computation

COMS 6424: Hardware Security

Monday, April 1, 2024

What is Multiparty Computation (MPC)

The secure multiparty computation may be defined as the problem of ‘n’ players to compute jointly on an agreed function securely on the inputs **without revealing them**.



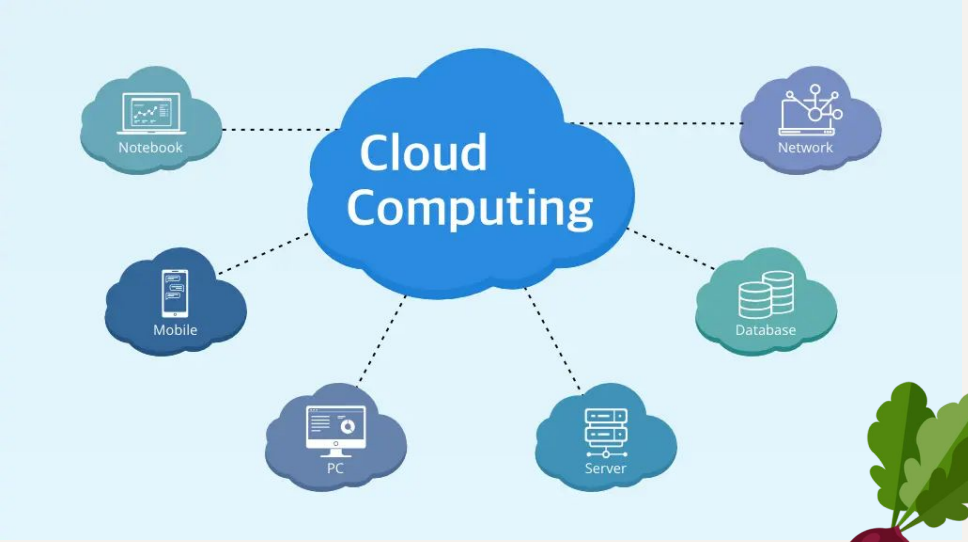
Multiple Parties Sharing data for performing a single Function F.

Why MPC ?

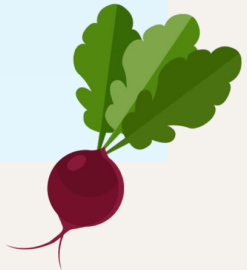
Growth of Cloud Computing

Data Protection

Training Machine Learning



Cloud Computing Infrastructure



History of MPC and PPMLAC

COLUMBIA UNIVERSITY

1970
Mental Poker

1982
Millionaire's Problem (2PC)

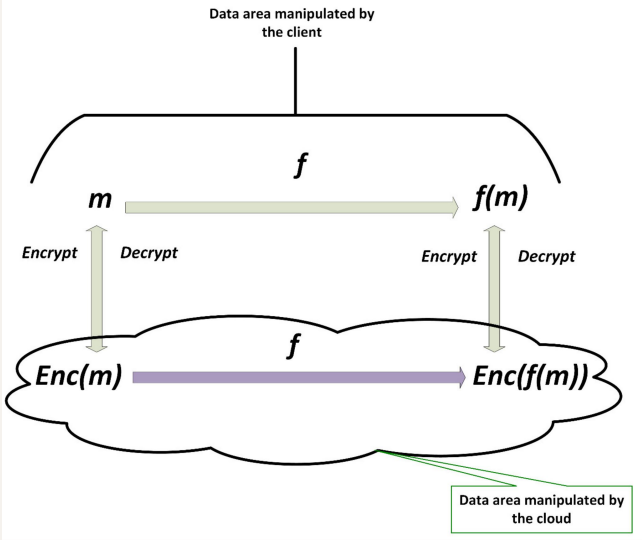
1996
Adaptive Secure MPC

2009
Fully-Homomorphic Encryption

2022
PPMLAC

Homomorphic Encryption

Homomorphic encryption is a form of encryption that allows computations to be performed on encrypted data **without first having to decrypt it.**



Reference : EL-YAHYAOUI, A.; ECH-CHERIF EL KETTANI, M.D. A Verifiable Fully Homomorphic Encryption Scheme for Cloud Computing Security. Technologies 2019, 7, 21. https://doi.org/10.3390/technologies7010021

Additive Secret Sharing (MPC Scheme)

Columbia University

Let us assume a n - party system, and take a node value x

$[x]_0,\ldots\ldots,[x]_i$ such that $x = \sum_{i=0}^{n-1} [x]_i$

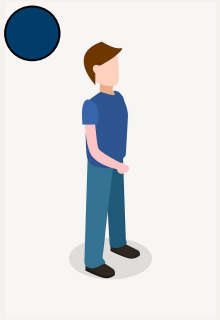
$z = x + y$

Each party -> $[z]_i = [x]_i + [y]_i$, which becomes the secret share of z.

Say 3 employees want to compute their average salary, without revealing their individual salaries to each other ?

Let's see an Example !

COLUMBIA UNIVERSITY

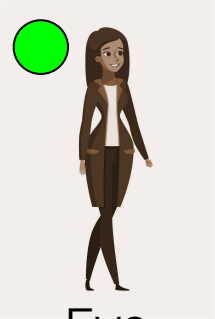


Bob

200 $

Alice

100 $



Eve

300 $

Alice	Bob	Eve
50 $	30 $	20 $
+	+	+
-80 $	100 $	180 $
+	+	+
0 $	350 $	-50 $
=	=	=
-30$	+ 480$	+ 150$

600 $

Beaver Triple

COLUMBIA UNIVERSITY

- **No**, What we want to compute: z = x * y
- So they generate a one-time table of triples (a, b, c), where, a and b are random numbers and c = a * b -> Beaver Triple

A = 5	B = 4	C = 20
A[0] = 3	B[0] = 3	C[0] = 12
A[1] = 2	B[1] = 1	C[1] = 8

Beaver Algorithm

Columbia University

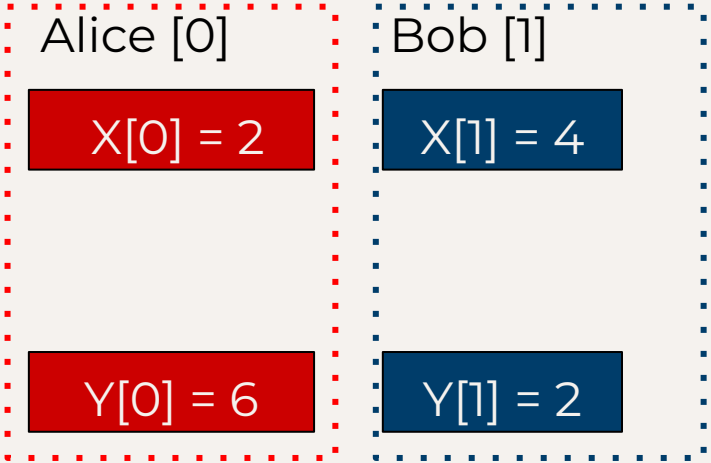
Alice [0]

6

⊠

Bob [1]

8



A = 5	B = 4	C = 20
A[0] = 3	B[0] = 3	C[0] = 12
A[1] = 2	B[1] = 1	C[1] = 8

Lets Compute

D[i] = X[i] - A[i]

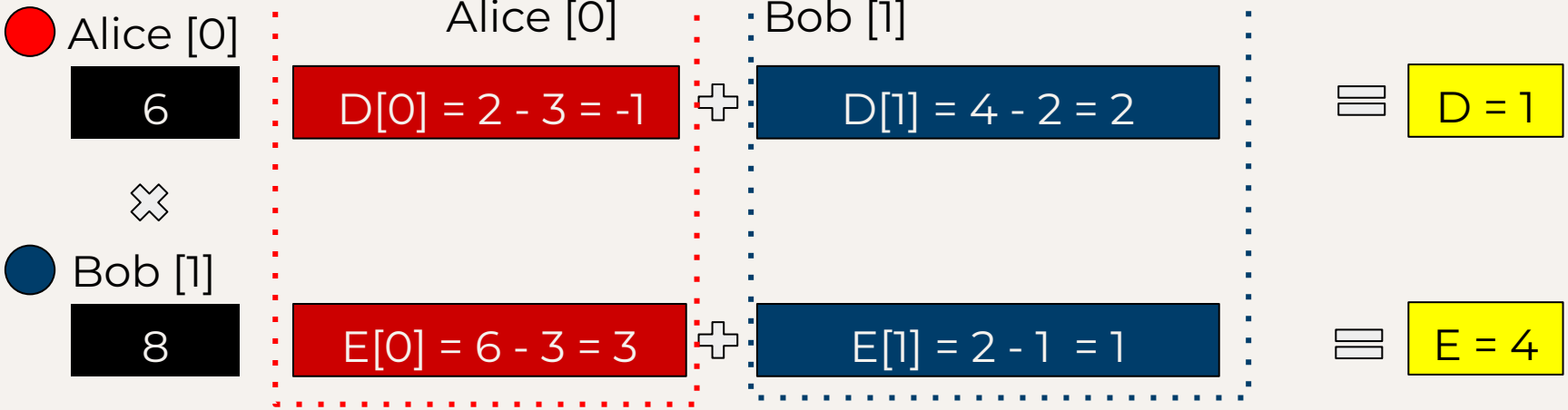
E[i] = Y[i] - B[i]

Beaver Triple

D[i] = X[i] - A[i]

E[i] = Y[i] - B[i]

COLUMBIA UNIVERSITY



X[0] = 2

X[1] = 4

Y[0] = 6

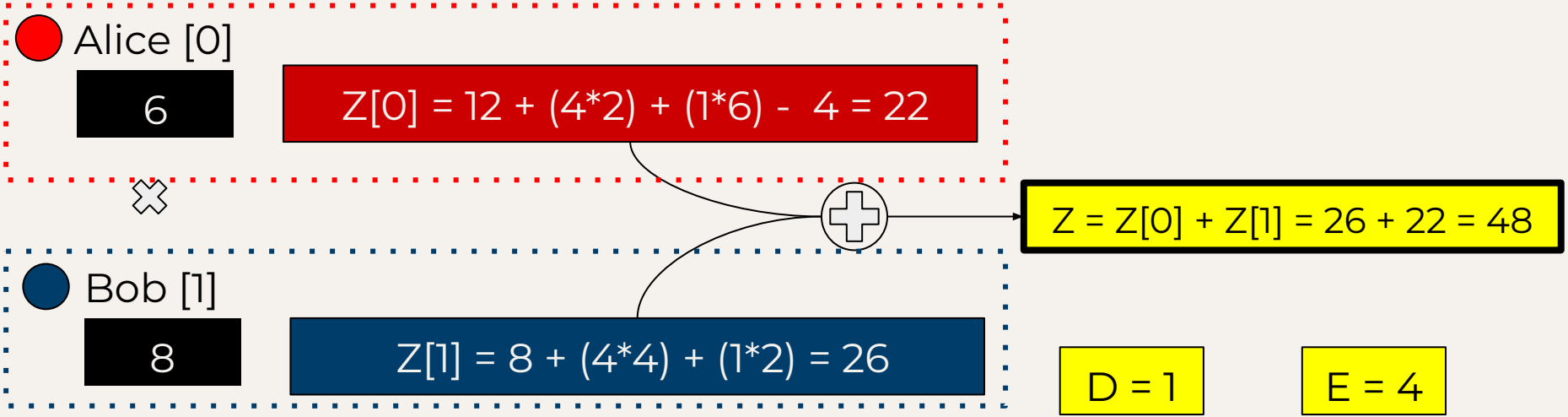
Y[1] = 2

A = 5	B = 4	C = 20
3	3	12
2	1	8

Beaver Triple

$$[z]_i = [x * y]_i = \begin{cases} [c]_i + e * [x]_i + d * [y]_i - e * d, & i = 0 \\ [c]_i + e * [x]_i + d * [y]_i, & i > 0 \end{cases}$$

COLUMBIA UNIVERSITY



A = 5	B = 4	C = 20
A[0] = 3	B[0] = 3	C[0] = 12
A[1] = 2	B[1] = 1	C[1] = 8

D = 1

E = 4

X[0] = 2

X[1] = 4

Y[0] = 6

Y[1] = 2

Beaver Triple Generation

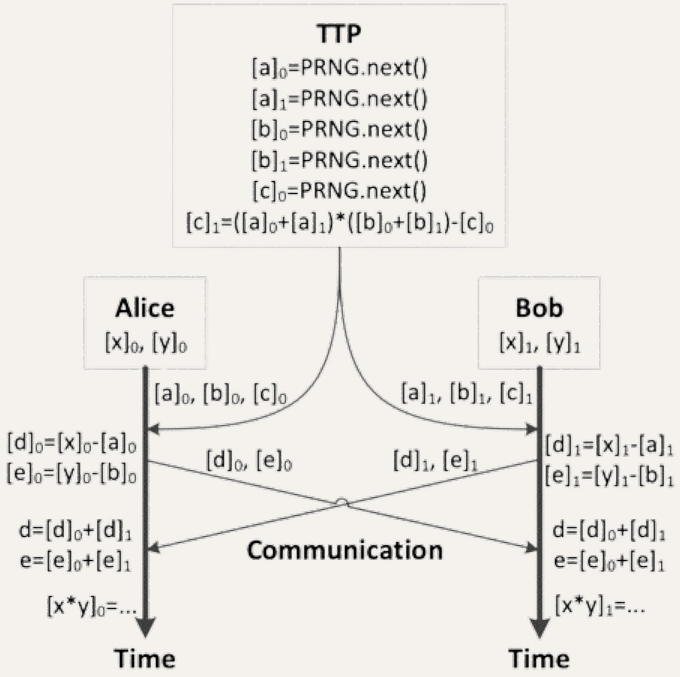
COLUMBIA UNIVERSITY

- We will have to generate a table for every multiplication : /
- And we will have to also share this table with all parties
- All current table generation techniques have latency 0.1ms to 1ms
- So what does this paper suggests?

Beaver Triple Generation

- TTP - Trusted Third Party
- TTP -> Trust is based on this party's credibility and neutrality

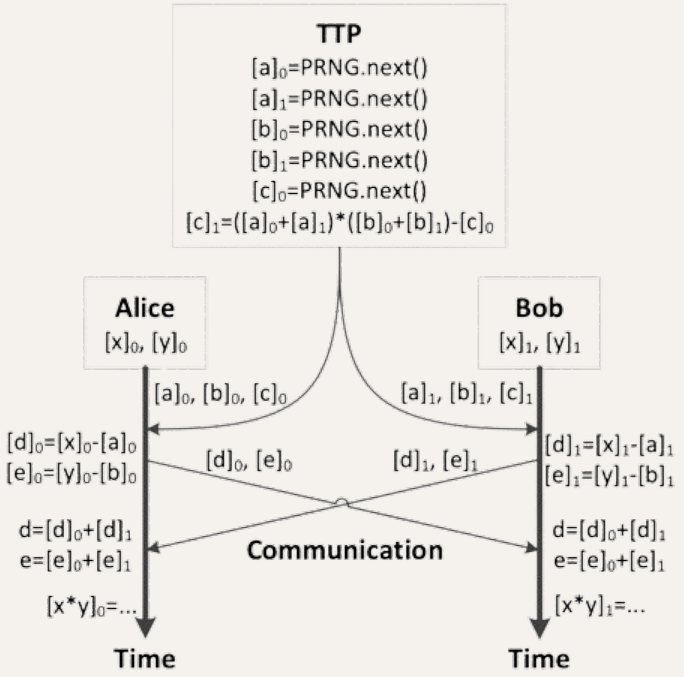
However, PPMLAC wants to remove the need of TTP. How did they do that?



existing TTP widely used in practice

Bottlenecks of MPC

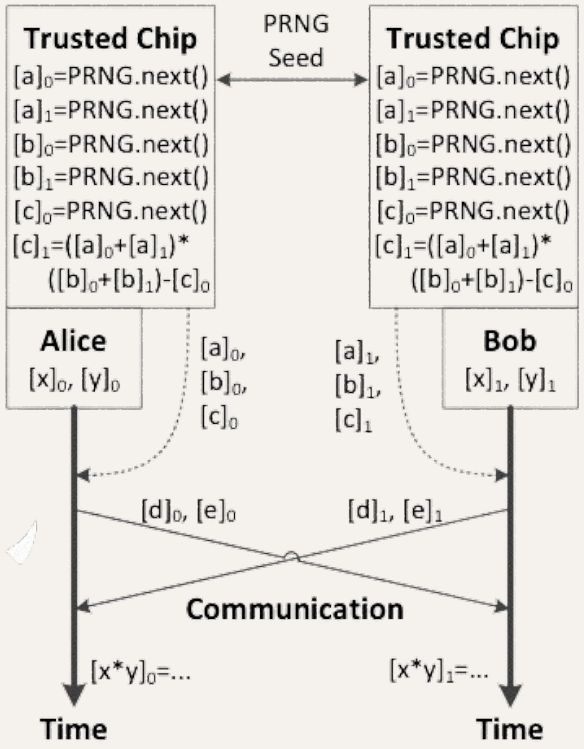
MPC needs the generation and distribution of Beaver triples, this results in communication overheads, ultimately hampering the performance and efficiency of MPC protocols.



(a) existing TTP widely used in practice

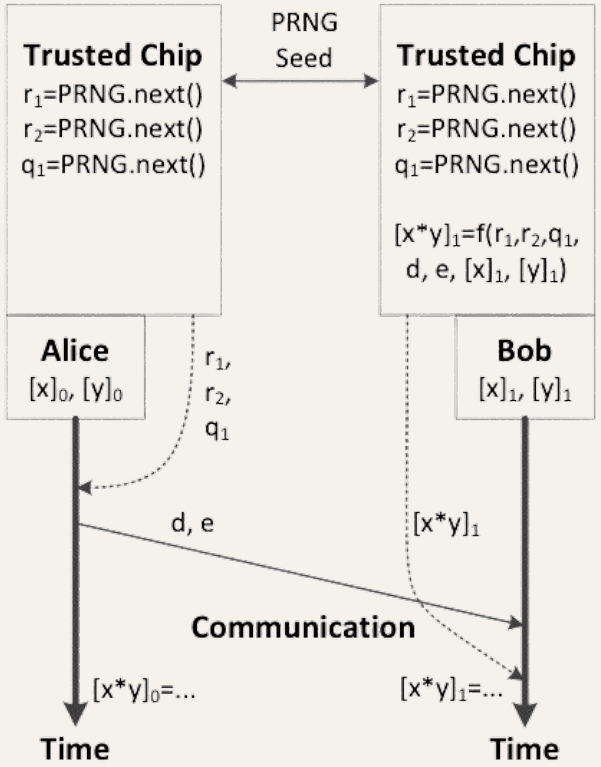
Innovation of PPMLAC

Each party has their own trusted chip that is given the same Seed and it provides the required random numbers to its local party and discards the rest.



Innovation of PPMLAC

We can choose a trusted chip to do all the multiplications after all other parties have send their secret shares to it. Thus, resulting in a one-way communication. All Trusted Chips need to be synchronized for this.



Overall

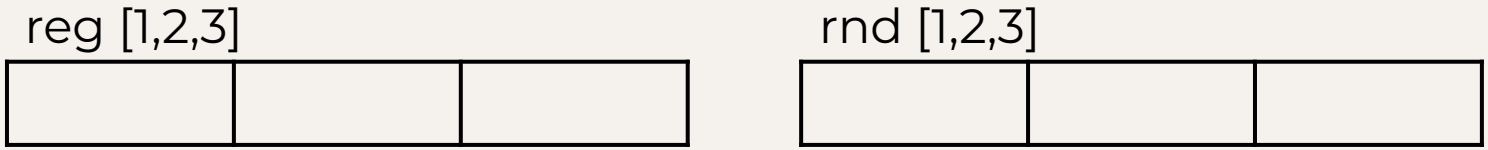
COLUMBIA UNIVERSITY

1. **Seed-Based Random Numbers** are generated locally.
2. **Centralized Multiplication** resulting in reduced communication overhead.
3. **Cache for Efficiency**: Trusted chip has a cache to reuse previous multiplicands.
4. **Robust Architecture**: The Trusted chip is built using minimum number of standard logic units like registers, multipliers and random number generators, thus making architecture robust to side channel attacks.

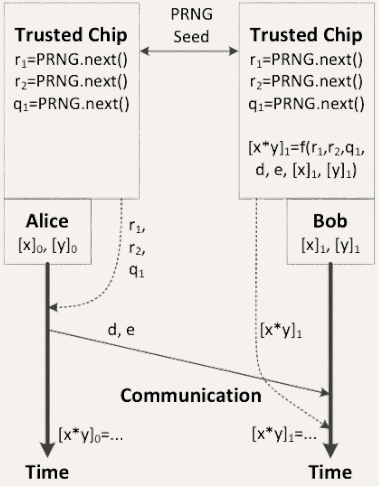
Assumptions made

COLUMBIA UNIVERSITY

- Only two parties: Alice & Bob (For ease of explanation)
- Both have a trusted chip
- Each chip have 3 registers and 3 random registers, each of 64-bit
- The on-chip CSPRNG is secure
- CSPRNG can generate numbers deterministically based on a seed



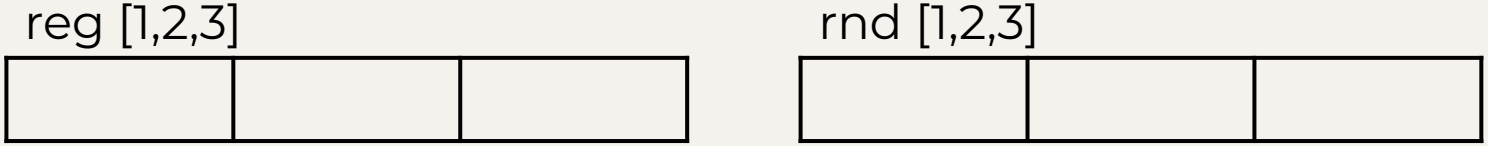
PPMLAC's Protocol 1



- Bob never communicates with Alice
- So if we prove Bob can't learn anything about Alice's secret share, the system is secure.

Trusted Chip's ISA

COLUMBIA UNIVERSITY



ld $reg, addr$ Load a word from memory to operand register
st $reg, addr$ Store a word from operand register to memory
setRnd rnd Let the CSPRNG generate a new random number r_j (from stream r); set random register rnd to be r_j
outRnd $memAddr$ Let CSPRNG generate a new random number r_j; copy r_j to memory memAddr **outRnd is disabled for Bob.**
outQnd $memAddr$ Let CSPRNG generate a new random number q_k; copy q_k to memory memAddr. **outQnd is disabled for Bob.**
mul $reg_1, rnd_1, reg_2, rnd_2, reg_3$ Let the CSPRNG generate a new random number q_k; set reg_3 to $(reg_1 + rnd_1) * (reg_2 + rnd_2) - q_k$

?

PPMLAC's Protocol 1

Initial Conditions

Alice [0]

6

Temporary Registers

a b d e

Memory

X[0] = 2

Y[0] = 6

q[0] = Invalid

Bob [1]

8

Registers

r[1] = Invalid

r[2] = Invalid

r[3] = Invalid

Temporary Registers

d e u v

Memory

X[1] = 4

Y[1] = 2

Z[1] = Invalid

Random Register

rnd[1] = Invalid

rnd[2] = Invalid

rnd[3] = Invalid

COLUMBIA UNIVERSITY

PPMLAC's Protocol 1

Alice [0]

6

Temporary Registers

a	b	d	e
3	2		

Memory

X[0] = 2
Y[0] = 6
q[0] = Invalid

```
outRnd <a>;
outRnd <b>;
```

Bob [1]

8

Registers

r[1] = Invalid
r[2] = Invalid
r[3] = Invalid

Temporary Registers

d	e	u	v

Memory

X[1] = 4
Y[1] = 2
Z[1] = Invalid

Random Register

rnd[1] = Invalid
rnd[2] = Invalid
rnd[3] = Invalid

COLUMBIA UNIVERSITY

PPMLAC's Protocol 1

Alice [0]

6

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2
Y[0] = 6
q[0] = Invalid

CPU

```
d <- x[0] - a;
e <- y[0] - b;
```

Bob [1]

8

Registers

r[1] = Invalid
r[2] = Invalid
r[3] = Invalid

Temporary Registers

d	e	u	v

Memory

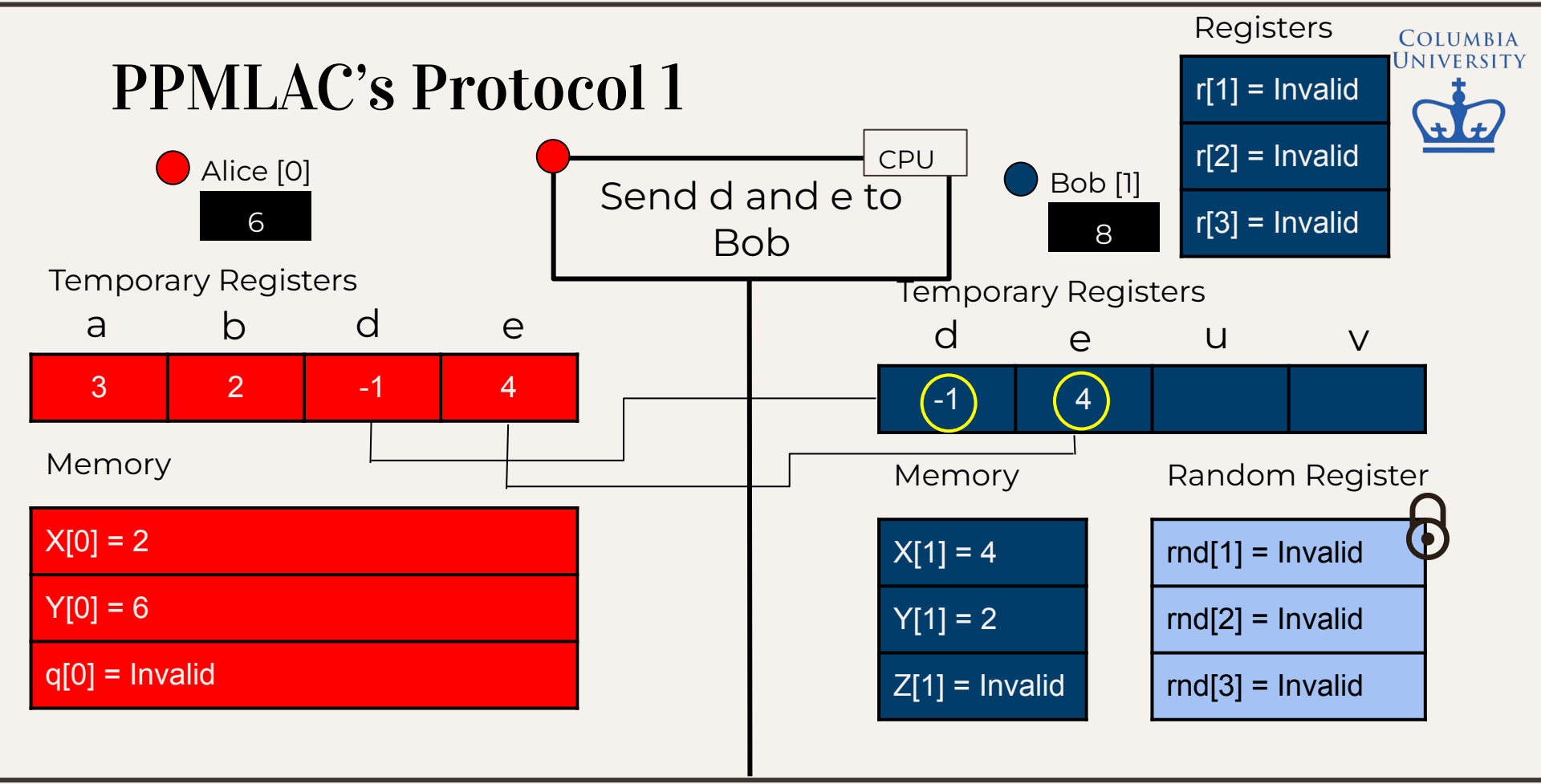
X[1] = 4
Y[1] = 2
Z[1] = Invalid

Random Register

rnd[1] = Invalid
rnd[2] = Invalid
rnd[3] = Invalid

COLUMBIA UNIVERSITY

PPMLAC's Protocol 1



PPMLAC's Protocol 1

Alice [0]

6

outQnd<q[0]>;

Bob [1]

8

Registers

r[1] = Invalid
r[2] = Invalid
r[3] = Invalid

COLUMBIA UNIVERSITY

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2
Y[0] = 6
q[0] = 5

Temporary Registers

d	e	u	v
-1	4		

Memory

X[1] = 4
Y[1] = 2
Z[1] = Invalid

Random Register

rnd[1] = Invalid
rnd[2] = Invalid
rnd[3] = Invalid

PPMLAC's Protocol 1

Alice [0]

6

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2
Y[0] = 6
q[0] = 5

```
setRnd<rnd1>;
setRnd<rnd2>;
setRnd<rnd3>;
```

Bob [1]

8

Registers

r[1] = Invalid
r[2] = Invalid
r[3] = Invalid

Temporary Registers

d	e	u	v
-1	4		

Memory

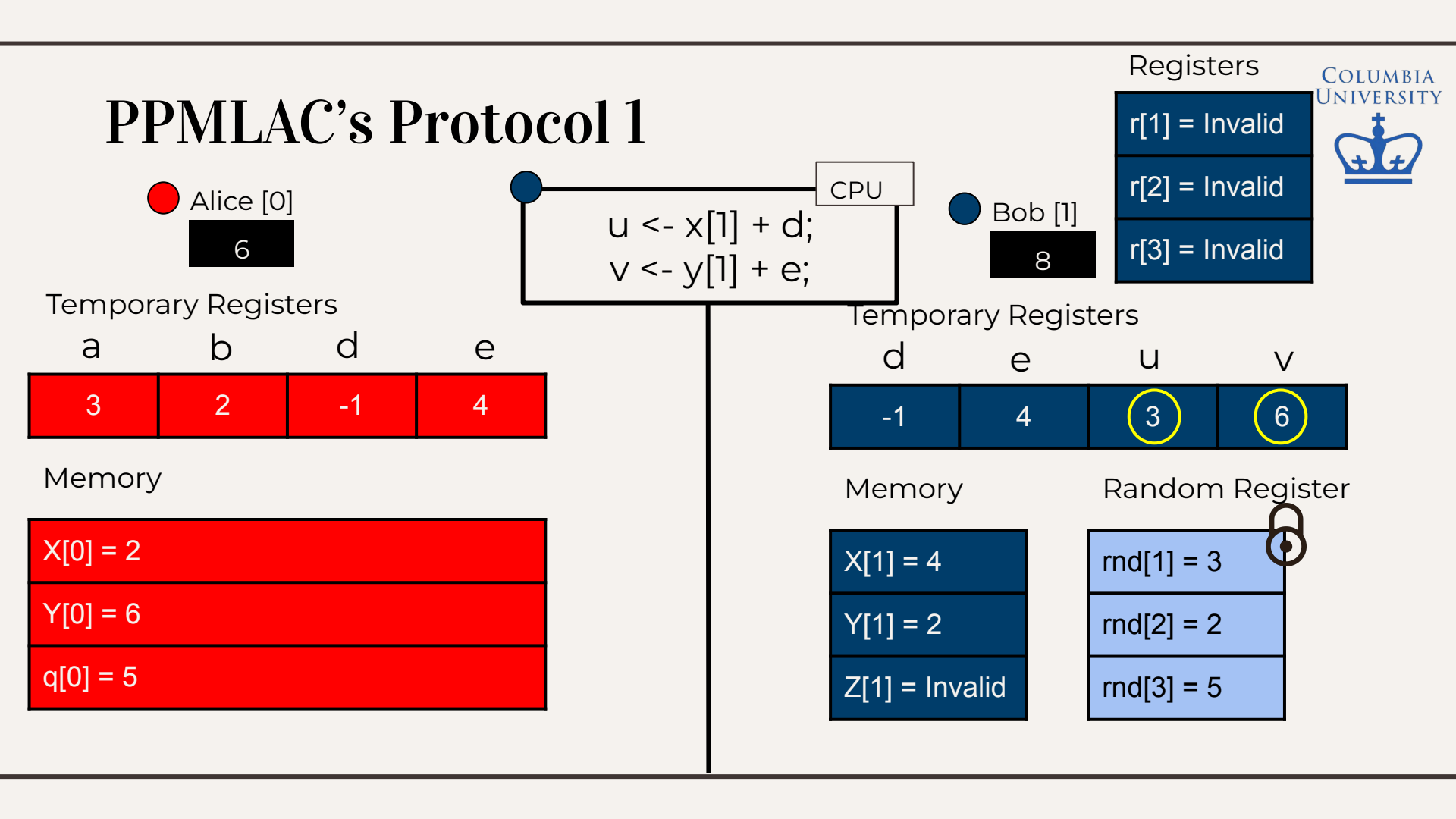
X[1] = 4
Y[1] = 2
Z[1] = Invalid

Random Register

rnd[1] = 3
rnd[2] = 2
rnd[3] = 5

COLUMBIA UNIVERSITY

PPMLAC's Protocol 1



PPMLAC's Protocol 1

Alice [0]

6

Ld reg1, <u>;
Ld reg2, <v>;

Bob [1]

8

Registers

r[1] = 3

r[2] = 6

r[3] = Invalid

COLUMBIA UNIVERSITY

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2

Y[0] = 6

q[0] = 5

Temporary Registers

d	e	u	v
-1	4	3	6

Memory

X[1] = 4

Y[1] = 2

Z[1] = Invalid

Random Register

rnd[1] = 3

rnd[2] = 2

rnd[3] = 5

PPMLAC's Protocol 1

Alice [0]

6

Mul ;

//r[3] = (r[1] + rnd[1]) * (r[2] + rnd[2]) - rnd[3] ;

Bob [1]

8

Registers

r[1] = 3

r[2] = 6

r[3] = 53

Z

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2

Y[0] = 6

q[0] = 5

Temporary Registers

d	e	u	v
-1	4	3	6

Memory

X[1] = 4

Y[1] = 6

Z[1] = Invalid

Random Register

rnd[1] = 3

rnd[2] = 2

rnd[3] = 5

COLUMBIA UNIVERSITY

PPMLAC's Protocol 1

Alice [0]

6

Temporary Registers

a	b	d	e
3	2	-1	4

Memory

X[0] = 2

Y[0] = 6

q[0] = 5

Is this secure if Bob deviates from the protocol ?

Registers

r[1] = 3

r[2] = 6

r[3] = 53

Bob [1]

8

Z

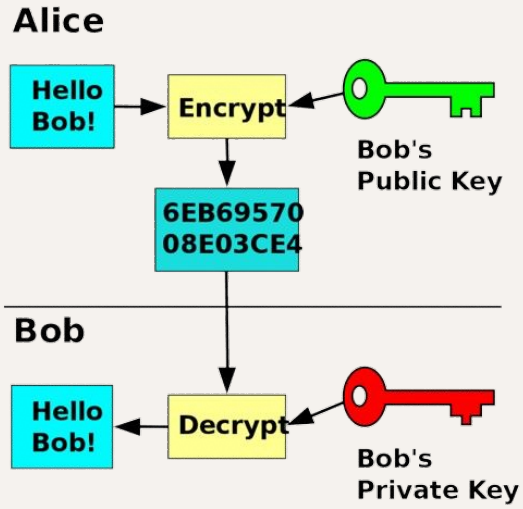
Output = 48

COLUMBIA UNIVERSITY

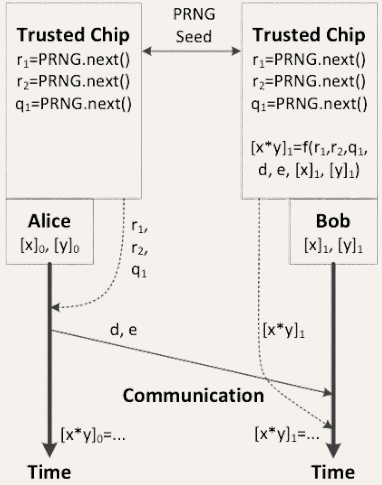
Protocol 1 : Security Issues

What if Bob gets access to the seed? 🚫

Hence, they use Asymmetric Cryptography



But now Bob can perform a replay attack, how?



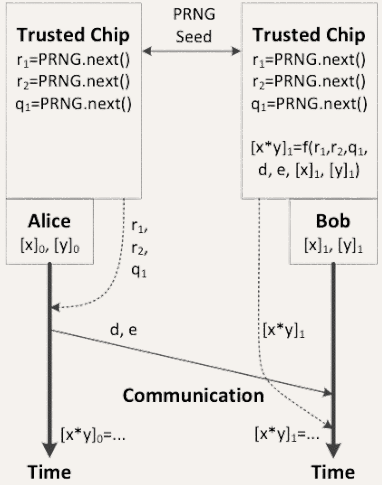
Protocol 1 : Security Issues

To break this,
seed, $s = m + TR$

m chosen by Alice,

unpredictable truly random number TR generated by Bob's trusted chip each time.

So now we will need a additional TRNG in the chip!



PPMLAC's Protocol 2

Alice [0]

6

Initial Conditions

Bob [1]

8

Temporary Registers

m x x' y

Temporary Registers

x y

Memory

PK[0] = Public Key

HK[0] = Hidden Key

-

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

Alice [0]

6

trueRnd <m>;

Bob [1]

8

Temporary Registers

m x x' y

m

Memory

PK[0] = Public Key

HK[0] = Hidden Key

-

Temporary Registers

x y

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

Alice [0]

6

Bob sends PK[1] to Alice

Bob [1]

8

Temporary Registers

m x x' y

m

Memory

PK[0] = Public Key

HK[0] = Hidden Key

PK[1] = Public Key

Temporary Registers

x y

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

Alice [0]

6

Alice performs remote attestation on PK[1]

Bob [1]

8

Temporary Registers

m x x' y

m

Memory

PK[0] = Public Key

HK[0] = Hidden Key

PK[1] = Public Key

Temporary Registers

x y

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

Alice [0]

6

x = Encrypt (PK[1], {m, PK[0]});
Send x to Bob ;

Bob [1]

8

Temporary Registers

m x x' y

m | x | | |

Memory

PK[0] = Public Key

HK[0] = Hidden Key

PK[1] = Public Key

Temporary Registers

x y

x | |

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

COLUMBIA UNIVERSITY

Alice [0]

6

{m, PK[0]} = Decrypt (x);
y = Encrypt(PK[0], TR);
Bob sends y to Alice;

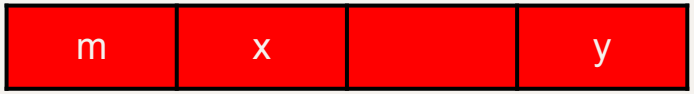
Bob [1]

8

GenB =m + TR

Temporary Registers

m x x' y



Memory

PK[0] = Public Key

HK[0] = Hidden Key

PK[1] = Public Key

Temporary Registers

x y

Where TR is a True-Random Number

Memory

HK[1] = Hidden Key

PK[1] = Public Key

PPMLAC's Protocol 2

Alice [0]

6

initA <m>, <y>;
//TR = Decrypt (HK[0], y)

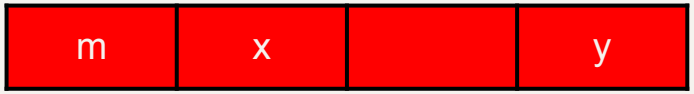
Bob [1]

8

GenB =m + TR

Temporary Registers

m x x' y



Memory

GenA =m + TR

PK[0] = Public Key

HK[0] = Hidden Key

PK[1] = Public Key

Temporary Registers

x y

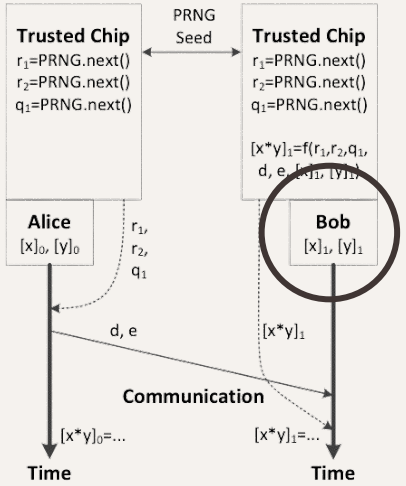
Memory

HK[1] = Hidden Key

PK[1] = Public Key

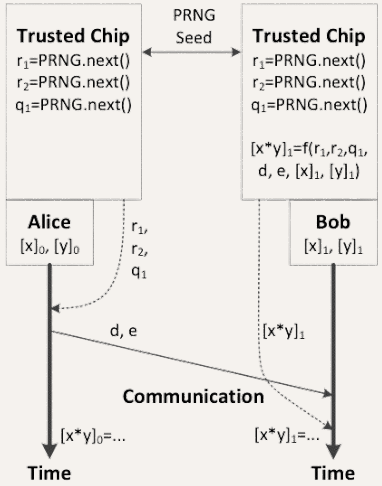
Beyond Two-Parties

- Each Alice and Bob have their own synchronous CSPRNG pair, so different Alices are mutually distrusted.
- Time-Multiplexing CSPRNG in Bob's side by storing states will help in optimizing area.



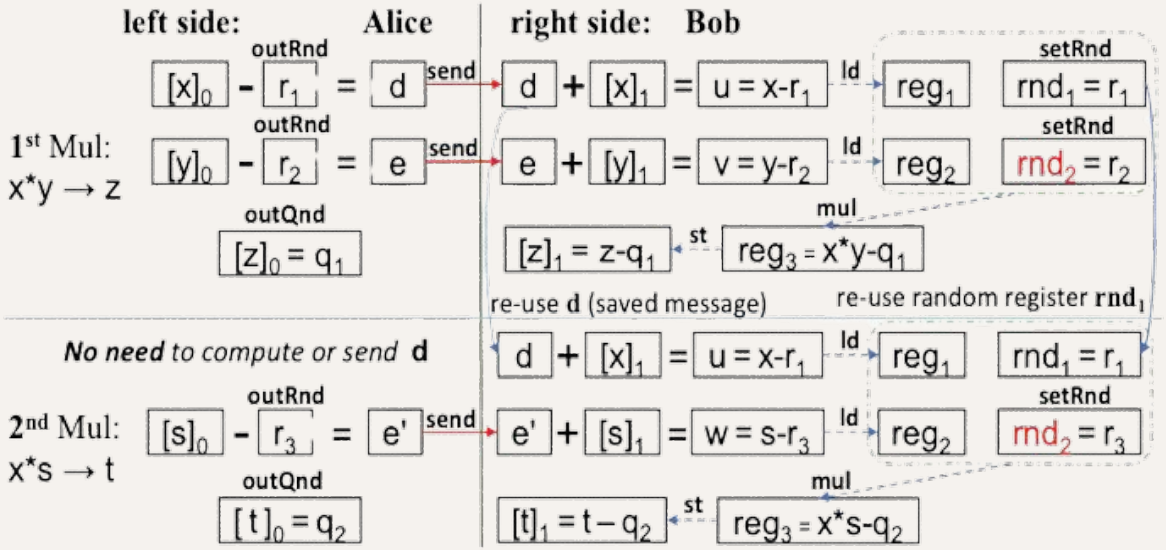
Security Guarantees

- Free from Control Flow and memory side-channels
- Forward Security
  - Secure, if Bob finds HK
- Man-in-the-middle-attack
  - Using Encypt{m,PK_0}



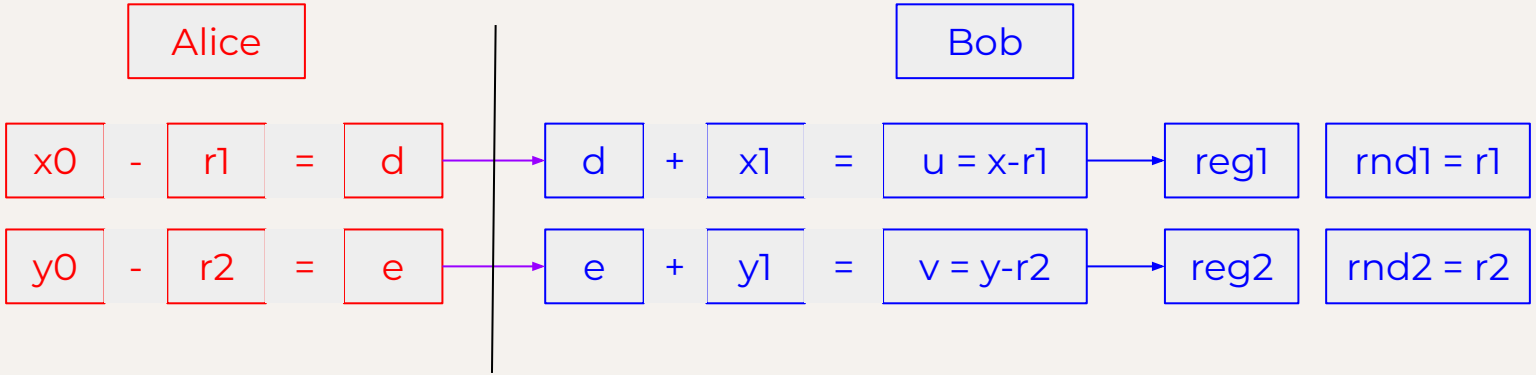
Reducing Multiplications by Caching

PPMLAC uses trusted chip of one of the parties to cache the data sent from other parties so that it can be used in later multiplications.



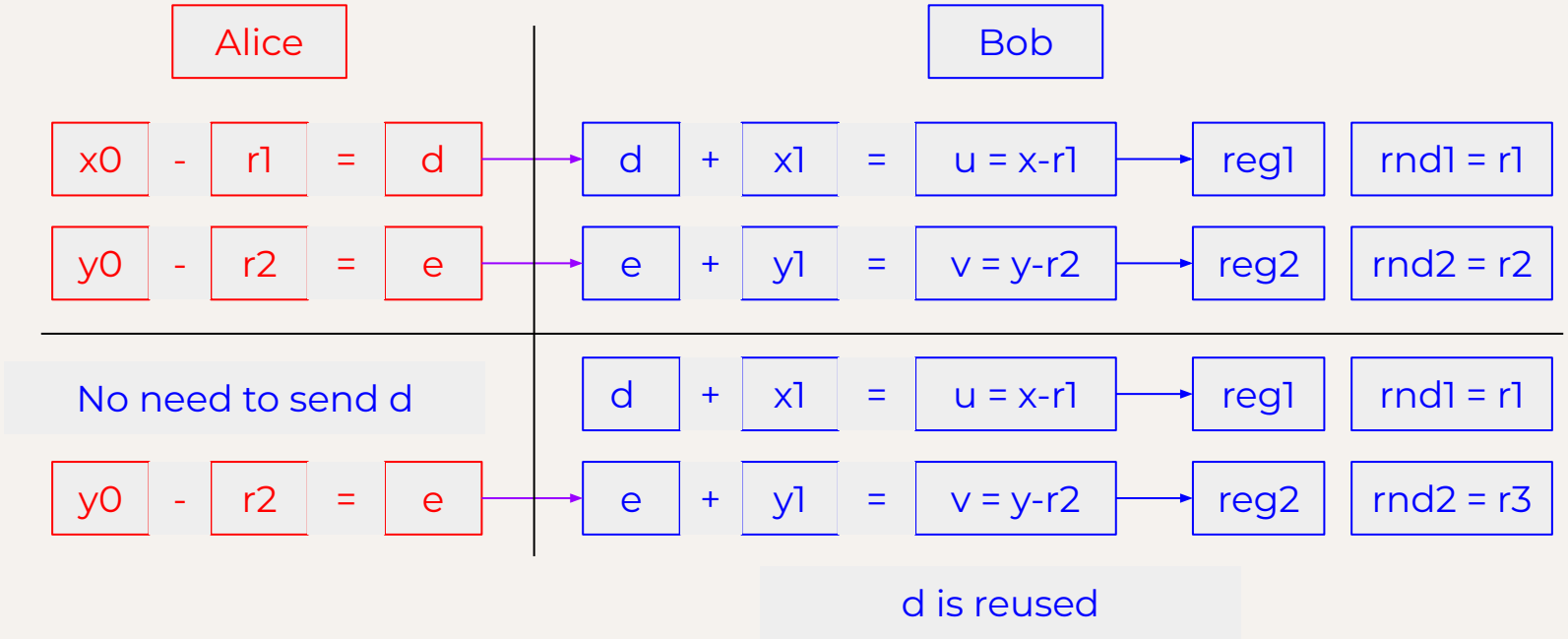
Reducing Multiplications by caching

Both parties compute x*y

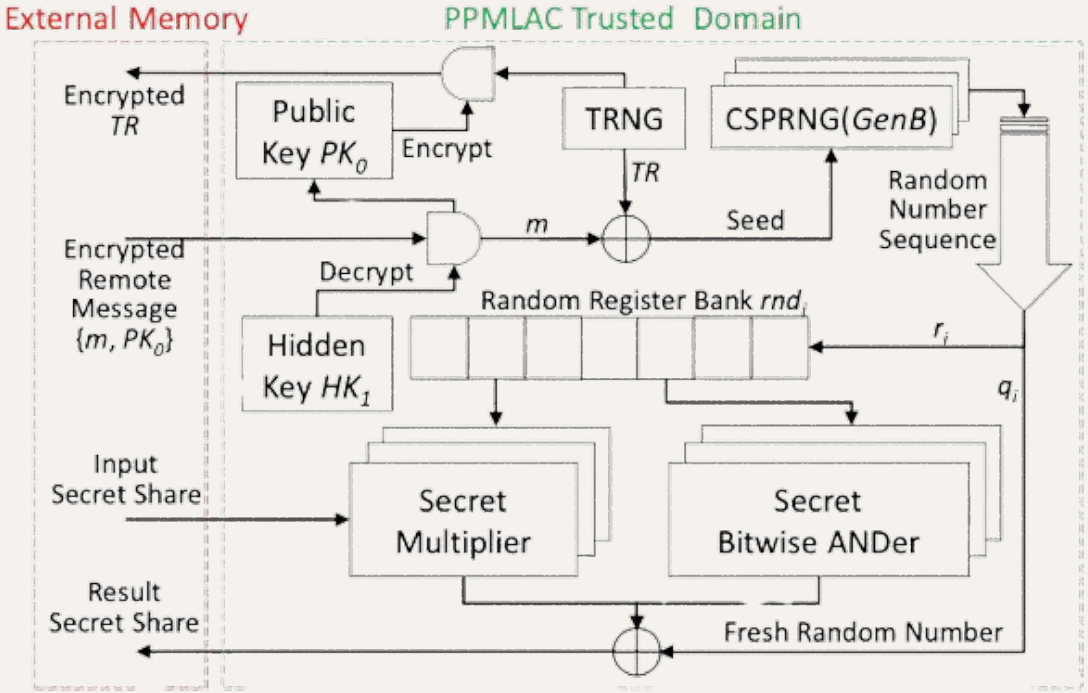


Reducing Multiplications by caching

Now Both parties want to compute x*s



Architecture for Bob's Side



Evaluation Parameters

COLUMBIA UNIVERSITY

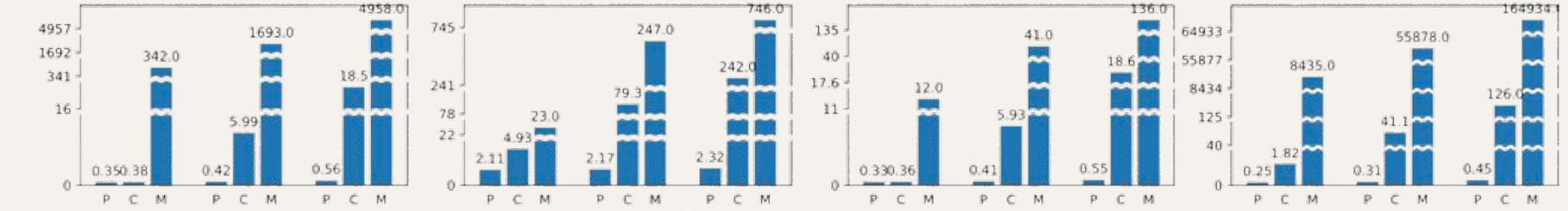
- PPMLAC is compared with two software MPC frameworks: the MP-SPDZ(M) and CrypTen(C).
- PPMLAC is implemented as a discrete accelerator on FPGA with 256 register banks and 100KB of cache. This is developed using HLS and running on AWS F1 instance.
- MP-SPDZ is implemented using 80-core Intel processor and 128GB Ram.
- CrypTen is implemented using V100 GPUs and Intel CPU.

Terms

COLUMBIA UNIVERSITY

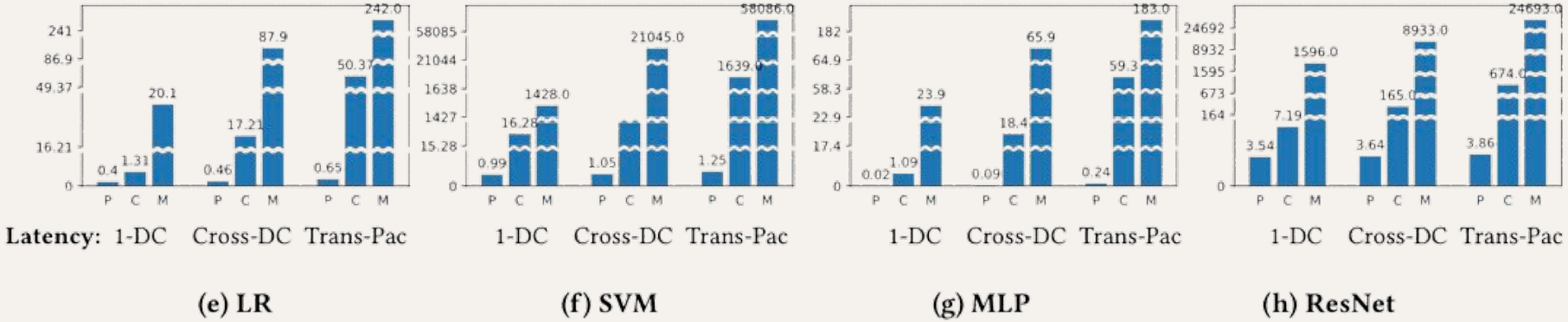
1. **1-DC**: Alice and Bob are 2 nodes in one datacenter with 0.3ms of latency.
1. **Cross-DC**: Alice and Bob have 65ms of latency to emulate Alice and Bob being in different datacenters.
2. **Trans-Pacific**: They have 200ms of latency to emulate connection between East Asia and US

Results



(a) Multiply $*$ (b) Comparison $<$ (c) Bitwise AND (d) Exponential e^x

Results



(e) LR (f) SVM (g) MLP (h) ResNet

Conclusion

COLUMBIA UNIVERSITY

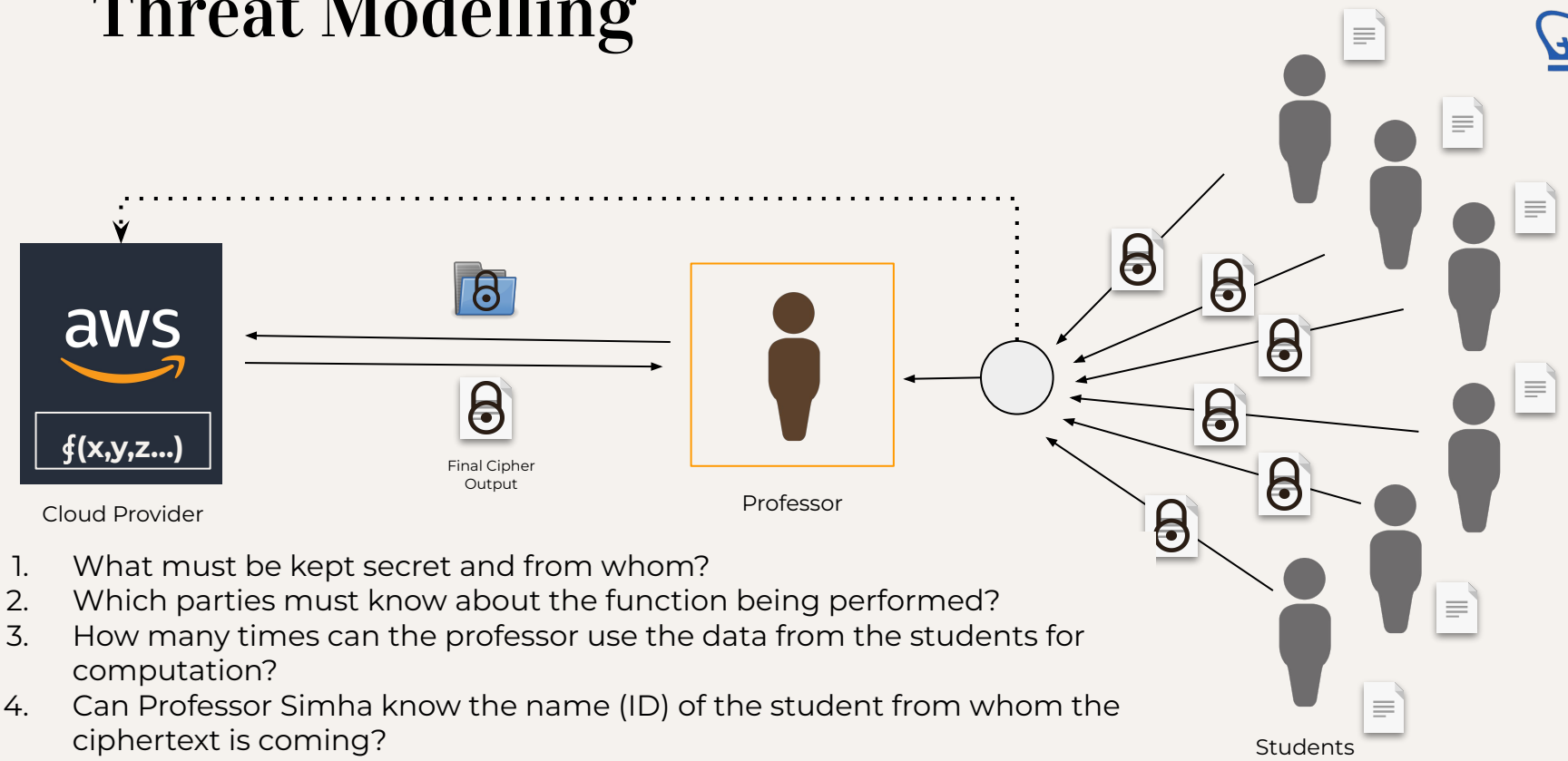
1. PPMLAC combines strong security of MPC with hardware's high performance.
2. Achieves several orders of magnitude speedup over software-based MPC.
3. Robust against side-channel attacks and malicious adversaries.
4. Enables ML models like ResNet to run on an FPGA which was impractical to do before PPMLAC.

COLUMBIA UNIVERSITY

Thanks!

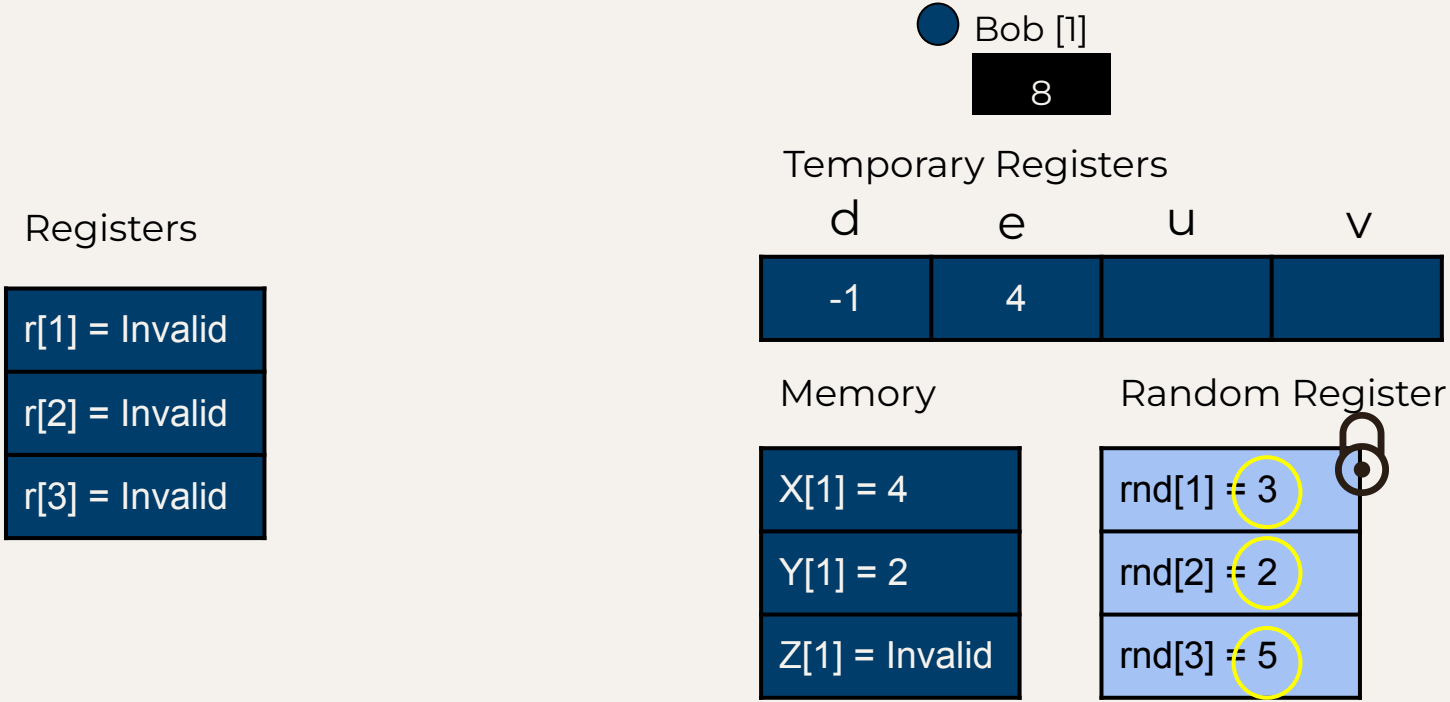
Reference: Zhou, Xing, et al. "PPMLAC: high performance chipset architecture for secure multi-party computation." Proceedings of the 49th Annual International Symposium on Computer Architecture. 2022.

Threat Modelling



1. What must be kept secret and from whom?
2. Which parties must know about the function being performed?
3. How many times can the professor use the data from the students for computation?
4. Can Professor Simha know the name (ID) of the student from whom the ciphertext is coming?
5. What else can go wrong?
6. Assumptions made

Bob's Side Chip



COLUMBIA
UNIVERSITY