Secure Frequent Pattern Mining by Fully Homomorphic Encryption with Ciphertext Packing

Hiroki Imabayashi, Yu Ishimaki, Akira Umayabara,
Hiroki Sato, and Hayato Yamana

Waseda Univ., Japan, Yamana Lab.

Outline

1. **Background** - What problems need to be solved ? –
2. **Proposal** - How to make the mining efficient? –
3. **Evaluation Results**
4. **Conclusion**

Background - Needs and Settings -

Processing data while preserving both input & output privacy

Input / Output Privacy

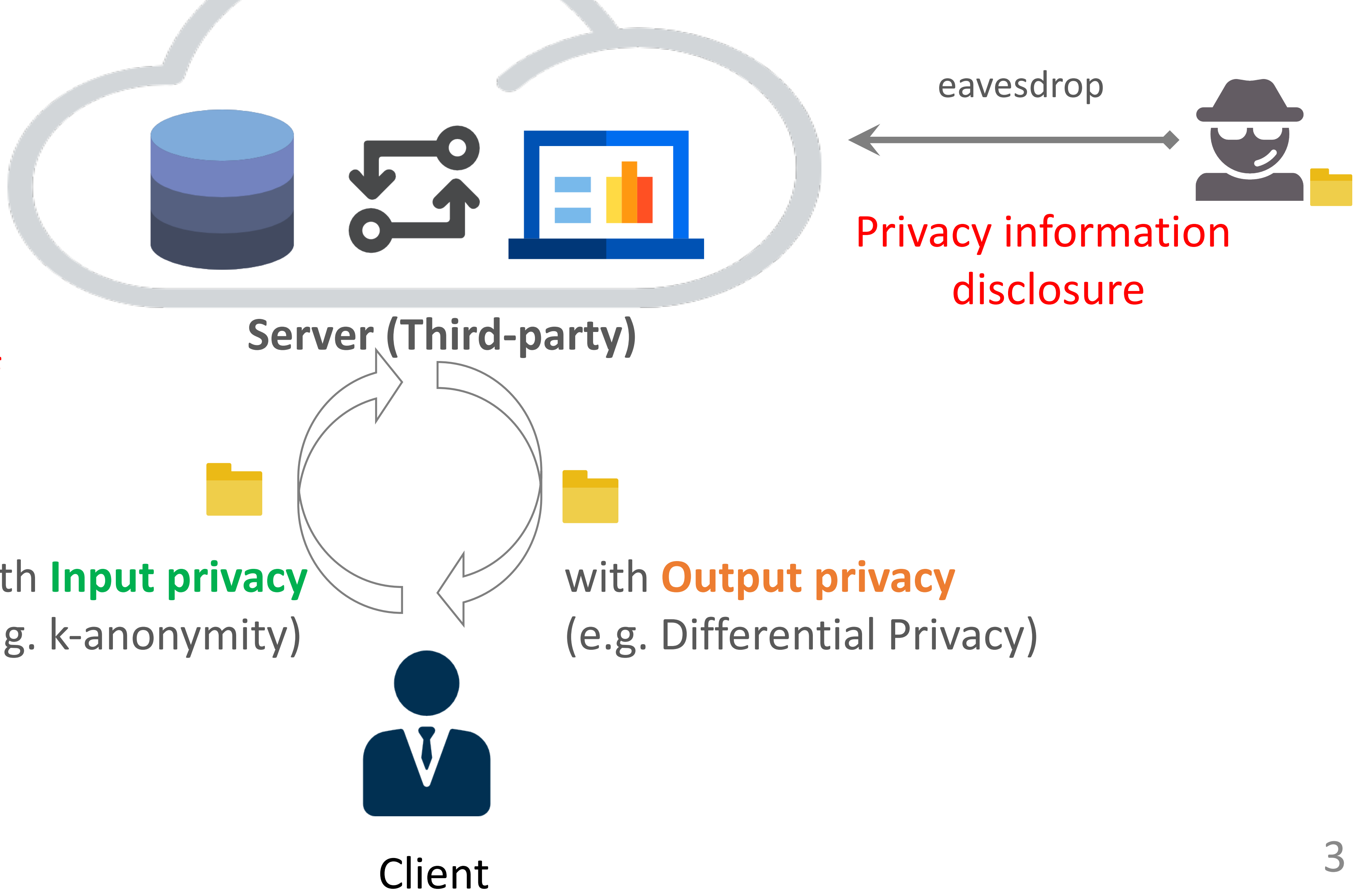
operations to data

- abstraction
- noise-addition
- perturbation

→ can not hide data itself

→ low mining accuracy

Confidential data



eavesdrop

Privacy information disclosure

Server (Third-party)

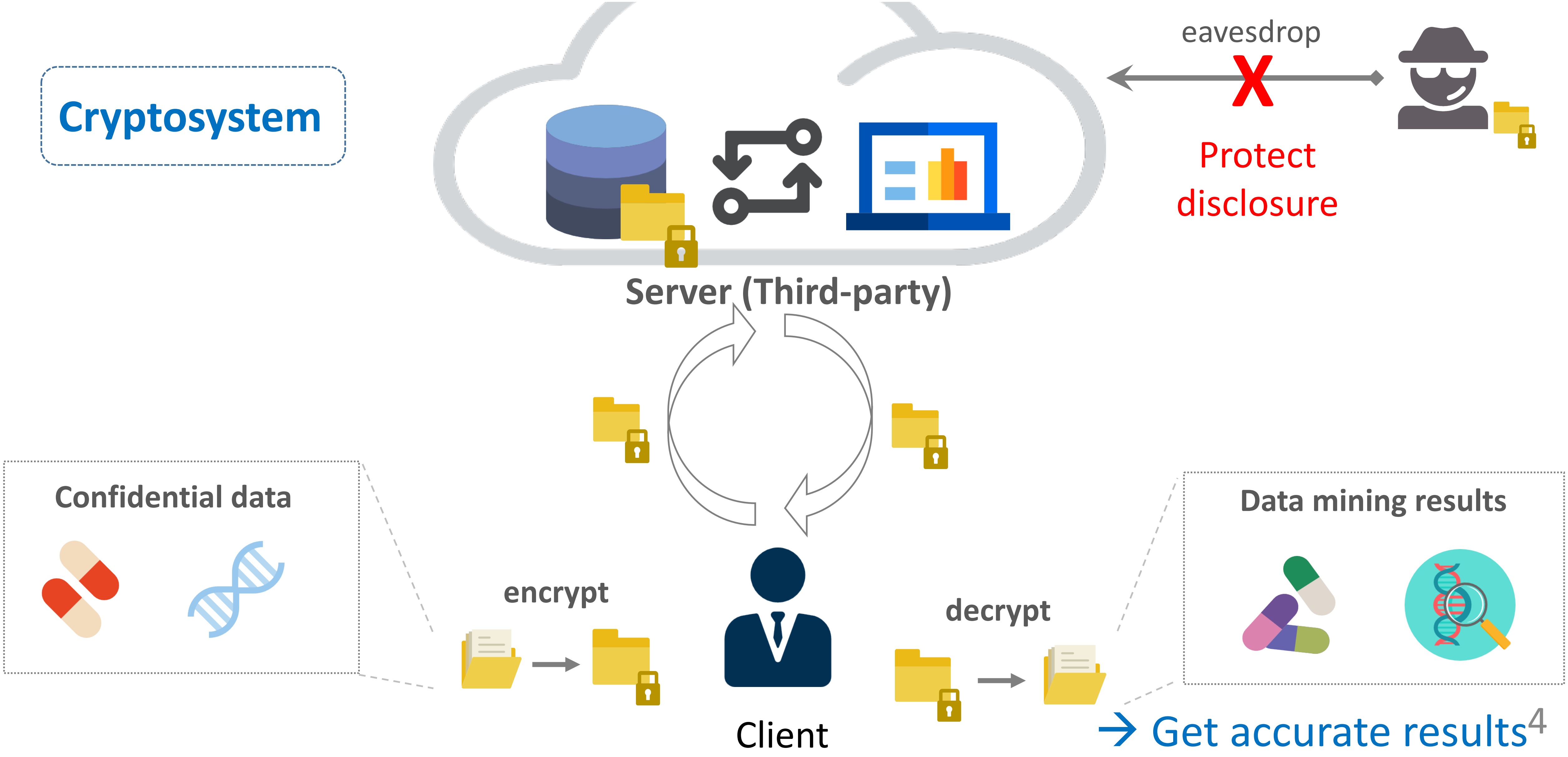
with **Input privacy**
(e.g. k-anonymity)

with **Output privacy**
(e.g. Differential Privacy)

Client

Processing data while preserving both input & output privacy

Cryptosystem

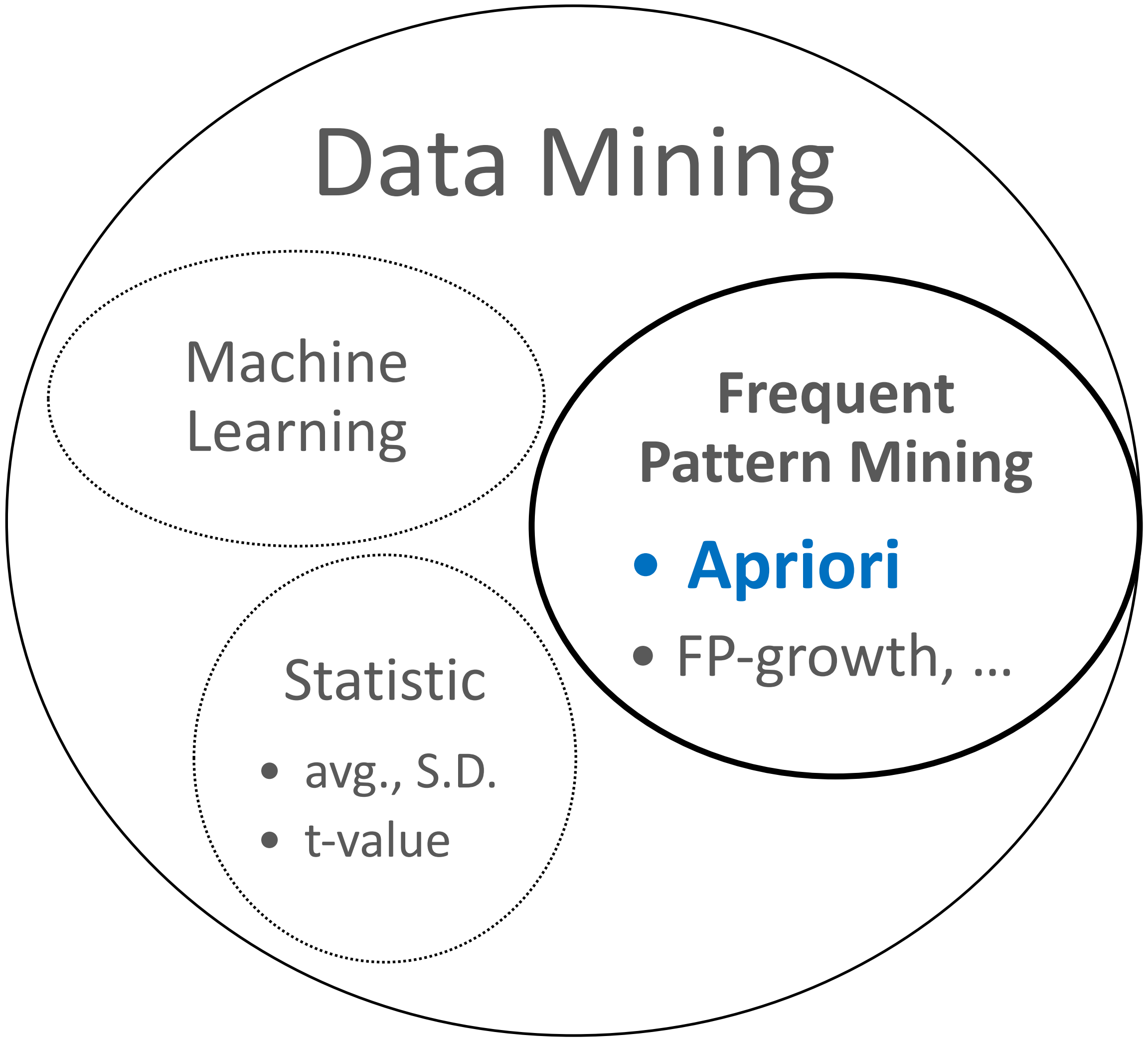


→ Get accurate results

Background - Needs and Settings -

Apriori as a frequent pattern mining algorithm

Which item pairs were appeared frequently **at once**?



Data Mining

Machine Learning

Statistic

- avg., S.D.
- t-value

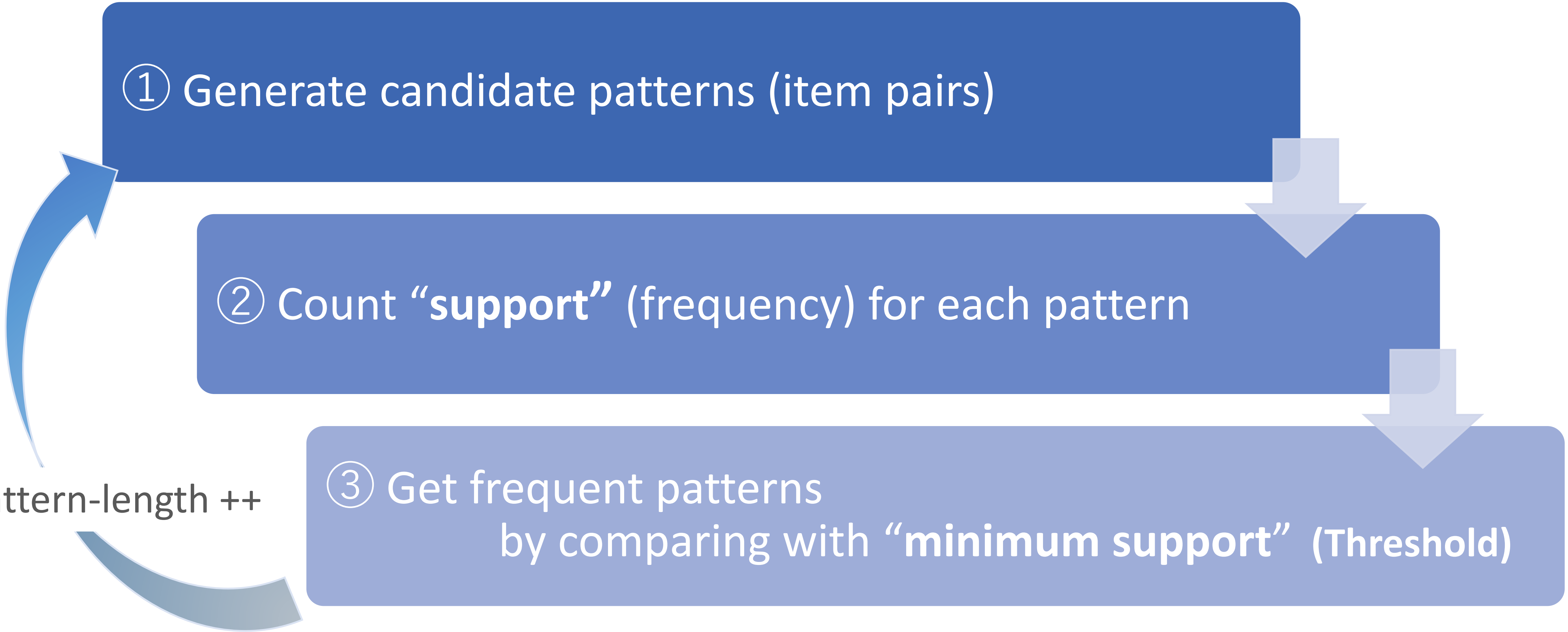
Frequent Pattern Mining

- **Apriori**
- FP-growth, …

Background - Apriori as frequent pattern mining -

Apriori Algorithm

pattern-length = 1
① {apple}, {orange}, {banana}, {melon}
② 8, 6, 5, 3
③ {apple}, {orange}, {banana} (minimum support = 5)



① Generate candidate patterns (item pairs)

② Count “**support**” (frequency) for each pattern

③ Get frequent patterns by comparing with “**minimum support**” **(Threshold)**

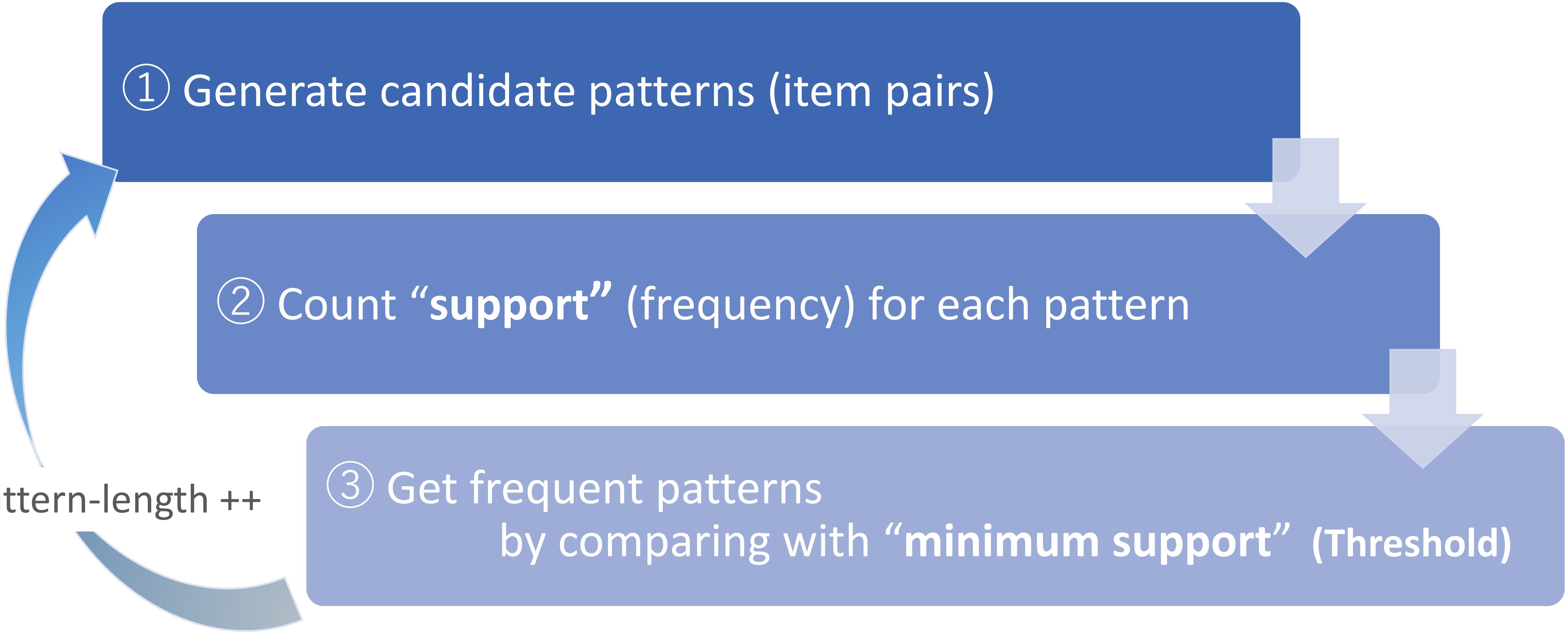
Pattern-length ++

Background - Apriori as frequent pattern mining -

Apriori Algorithm

{apple}, {orange}, {banana}

pattern-length = 2
① {apple, orange}, {apple, banana}, {orange, banana}
② 6, 5, 3
③ {apple, orange}, {apple, banana} (minimum support = 5)



① Generate candidate patterns (item pairs)

② Count “**support**” (frequency) for each pattern

③ Get frequent patterns
by comparing with “**minimum support**” **(Threshold)**

Pattern-length ++

Background - Apriori as frequent pattern mining -

Apriori Algorithm

{apple, orange}, {apple, banana}

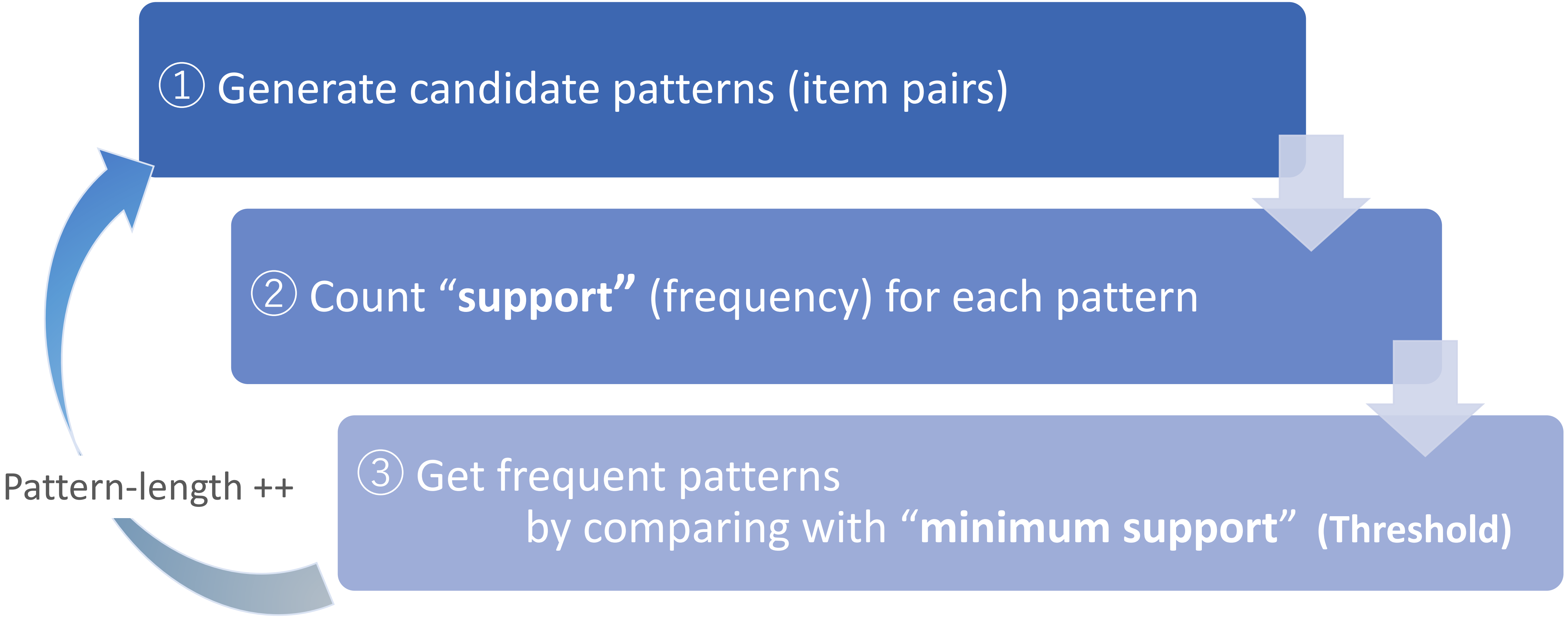
pattern-length = 3

① {apple, orange, banana}

② 3

③ none

(minimum support = 5)



① Generate candidate patterns (item pairs)

② Count “**support**” (frequency) for each pattern

③ Get frequent patterns
by comparing with “**minimum support**” **(Threshold)**

Pattern-length ++

Background - Apriori as frequent pattern mining -

How Apriori works with a transaction dataset?

Trans. ID	Item Set
T1	{a, b, e}
T2	{a, b, c, d}
T3	{b, e}
T4	{a, b, c, e, f}
T5	{a, b, c, d}
T6	{a, b, c, d, f}

Definition

support:
“Frequency of pattern”

minimum support:
”Threshold of frequent”

(minimum support = 3)

Pattern length	① Candidate Patterns	② Supports	③ Frequent Patterns
1	{a}, {b}, {c}, {d}, {e}, {f}	5, 6, 5, 3, 3, 2	{a}, {b}, {c}, {d}, {e}
2	{a, b}, {a, c}, {a, d}, {a, e},{b, c}, {b, d}, {b, e}, {c, d}, {c, e}, {d, e}	5, 4, 3, 2, 3, 3, 2, 3, 1, 0	{a, b}, {a, c}, {a, d}, {b, c}, {b, d}, {c, d}
3	{a, b, c}, {a, b, d}, {a, c, d}, {b, c, d}	4, 3, 2, 3	{a, b, c}, {a, b, d}, {b, c, d}
4	{a, b, c, d}	3	{a, b, c, d}

Background - Counting support for FHE -

How to calculate the “support” over ciphertexts?

Trans. ID	Item Set
T1	{a, b, e}
T2	{a, b, c, d}
T3	{b, e}
T4	{a, b, c, e, f}
T5	{a, b, c, d}
T6	{a, b, d, f}

⇨

Items Trans	a	b	c	d	e	f
T1	1	1	0	0	1	0
T2	1	1	1	1	0	0
T3	0	1	0	0	1	0
T4	1	1	1	0	1	1
T5	1	1	1	1	0	0
T6	1	1	0	1	0	1

binary representation

ex.
support of
{a, b, c}

a		b		c		
1	x	1	x	0	=	0
1	x	1	x	1	=	1
0	x	1	x	0	=	0
1	x	1	x	1	=	1
1	x	1	x	1	=	1
1	x	1	x	0	=	0

Sum up
3

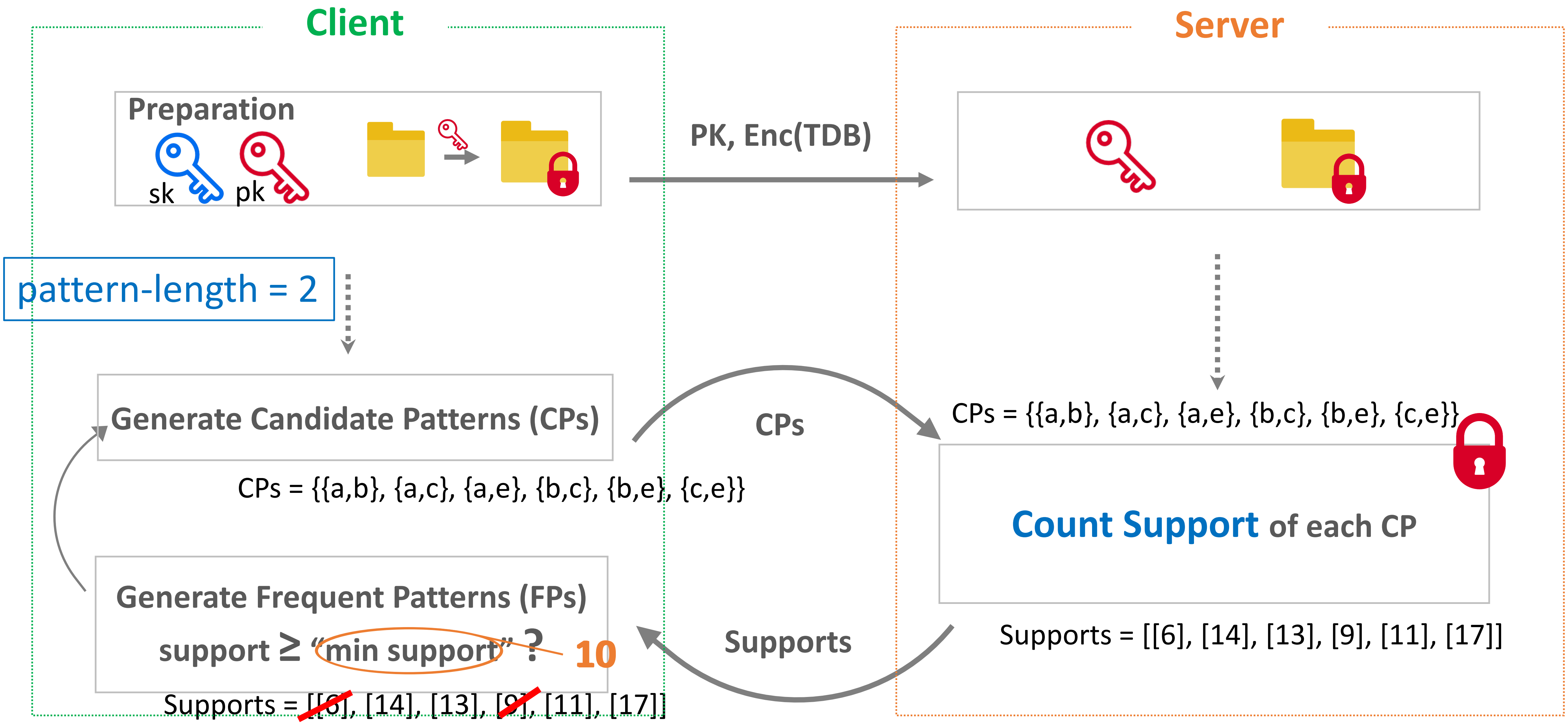
“Mult” & “Add”
are required

⇩

Fully Homomorphic Encryption

Related work - Privacy Preserving Protocol for Counting Candidates (P3CC) [Liu et al. 15] -

Server executes only “Support-Counting” to skip comparison over ciphertexts



Client

Server

Preparation

sk

pk

PK, Enc(TDB)

pattern-length = 2

Generate Candidate Patterns (CPs)

CPs = {{a,b}, {a,c}, {a,e}, {b,c}, {b,e}, {c,e}}

CPs

CPs = {{a,b}, {a,c}, {a,e}, {b,c}, {b,e}, {c,e}}

Count Support of each CP

Supports = [[6], [14], [13], [9], [11], [17]]

Supports

Generate Frequent Patterns (FPs)

support ≥ “min support” ? 10

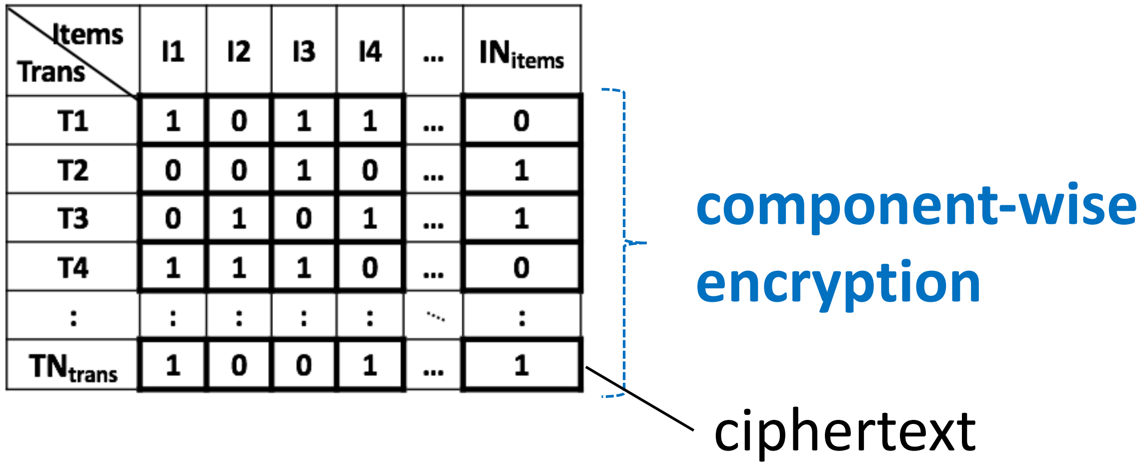
Supports = [[~~6~~], [14], [13], [~~9~~], [11], [17]]

FPs = {{a,c}, {a,e}, {b,e}, {c,e}}

No comparison over encrypted data

Related work - Component-wise encryption in P3CC -

P3CC's component-wise encryption scheme has large time/space complexities



Items Trans	I1	I2	I3	I4	…	IN_{items}
T1	1	0	1	1	…	0
T2	0	0	1	0	…	1
T3	0	1	0	1	…	1
T4	1	1	1	0	…	0
:	:	:	:	:	⋱	:
TN_{trans}	1	0	0	1	…	1

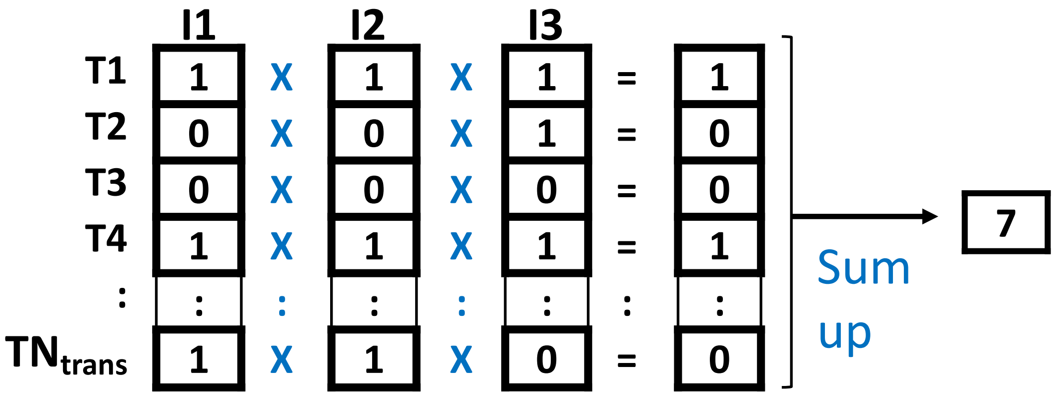
component-wise encryption

ciphertext

Needs many ciphertexts

↳ memory usage ⇧

ex. The support of **{I1, I2, I3}** is calculated **component-wisely**



	I1		I2		I3		
T1	1	X	1	X	1	=	1
T2	0	X	0	X	1	=	0
T3	0	X	0	X	0	=	0
T4	1	X	1	X	1	=	1
:	:	:	:	:	:	:	:
TN_{trans}	1	X	1	X	0	=	0

Sum up

7

Execute many multiplications

↳ execution time ⇧

Proposal - Solution 1. Ciphertext-packing -

Ciphertext-Packing reduces both time/space complexities

Items / Trans	I1	I2	I3	I4	…	IN_{item}
T1	1	0	1	1	…	0
T2	0	0	1	0	…	1
T3	0	1	0	1	…	1
T4	1	1	0	0	…	0
:	:	:	:	:	⋱	:
TN_{trans}	1	0	0	1	…	1

column-wise encryption

Reduce #ciphertexts to $1/N_{trans}$

↳ memory usage ⇩

ex. The support of **{I1, I2, I3}** is calculated by **batching**

$$\begin{array}{ccccccccc} & \text{I1} & & \text{I2} & & \text{I3} & & & \\ \text{T1} & 1 & & 1 & & 1 & & 1 & \\ \text{T2} & 0 & & 1 & & 0 & & 0 & \\ \text{T3} & 0 & & 0 & & 0 & & 0 & \\ \text{T4} & 1 & & 1 & & 1 & & 1 & \\ : & : & \otimes & : & \otimes & : & = & : & \\ TN_{trans} & 1 & & 0 & & 0 & & 0 & \\ & 0 & & 0 & & 0 & & 0 & \\ & : & & : & & : & & : & \\ & 0 & & 0 & & 0 & & 0 & \end{array} \xrightarrow{\text{Sum up*}} \boxed{7}$$

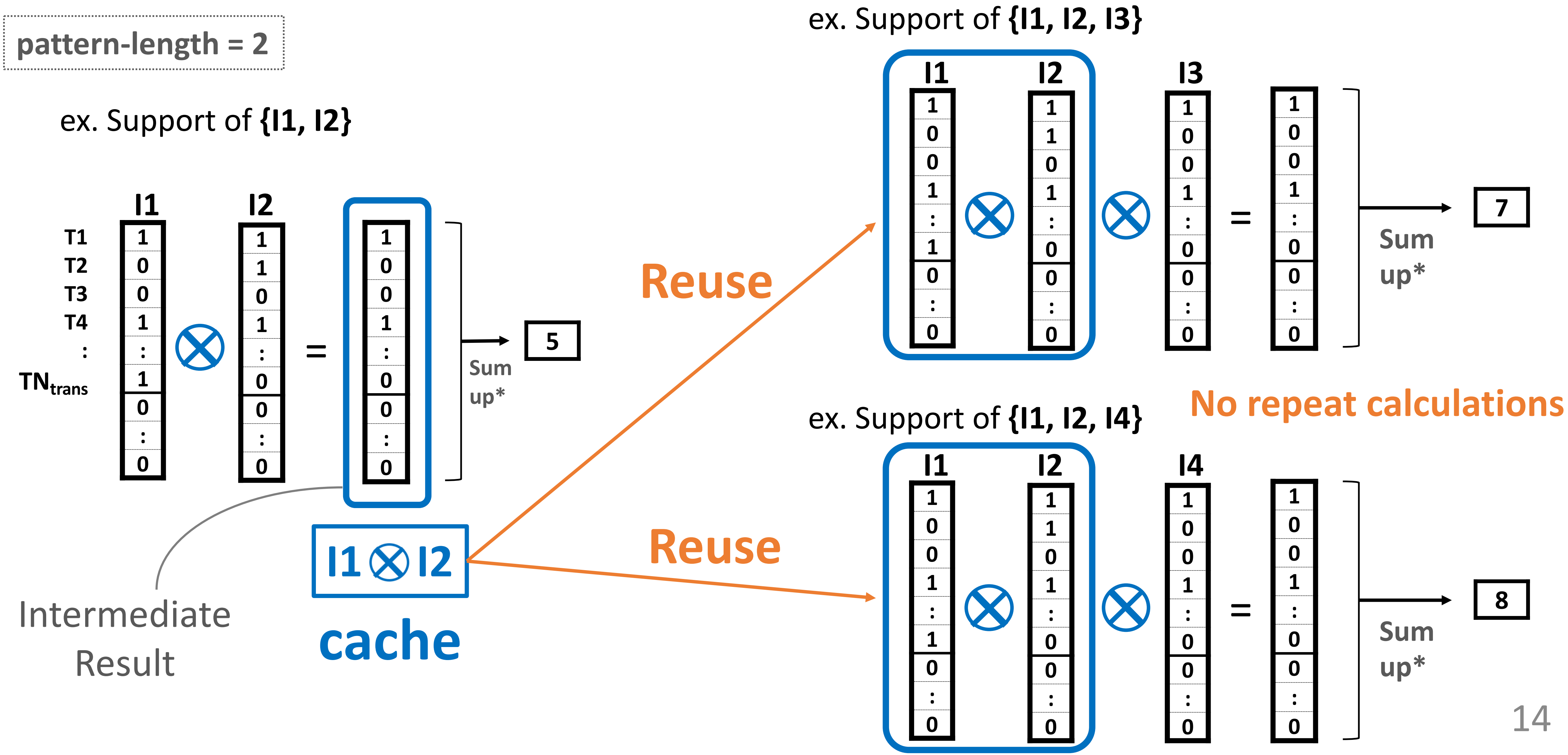
Execute fewer multiplications

↳ execution time ⇩

Proposal - Solution 2. Ciphertext-caching -

How ciphertext-caching works?

pattern-length = 3



pattern-length = 2

ex. Support of **{I1, I2}**

ex. Support of **{I1, I2, I3}**

ex. Support of **{I1, I2, I4}**

Reuse

Reuse

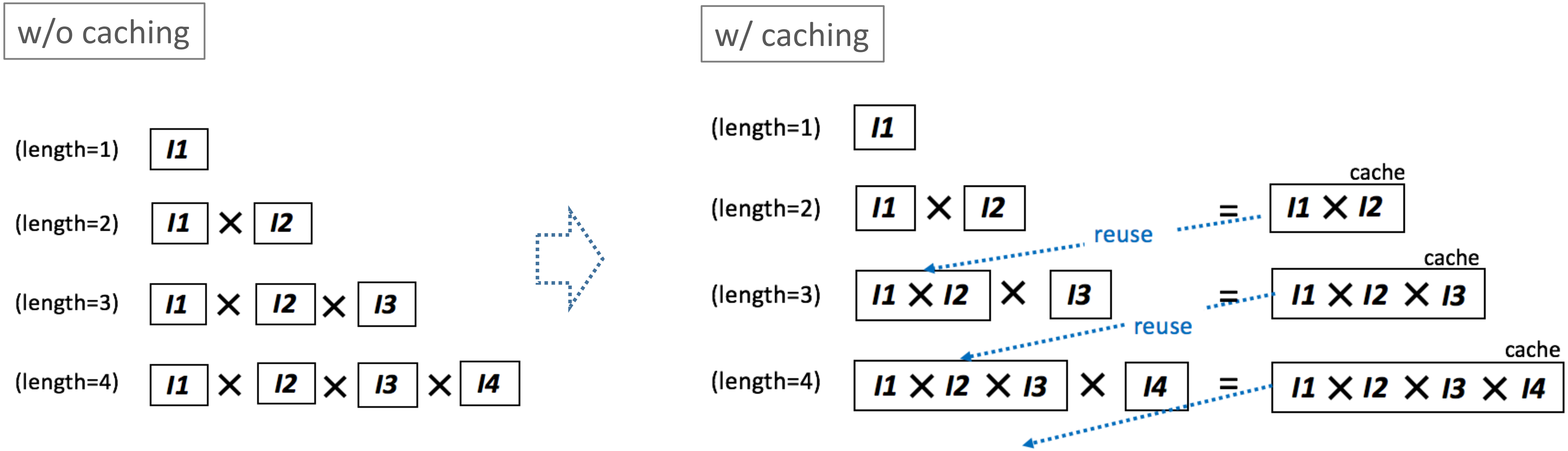
No repeat calculations

Intermediate Result

cache

Proposal - Solution 2. Ciphertext-caching -

Ciphertext-caching make the support-counting execution faster



w/o caching
(length=1) I1
(length=2) I1 × I2
(length=3) I1 × I2 × I3
(length=4) I1 × I2 × I3 × I4
w/ caching
(length=1) I1
(length=2) I1 × I2 = I1 × I2
cache
reuse
(length=3) I1 × I2 × I3 = I1 × I2 × I3
cache
reuse
(length=4) I1 × I2 × I3 × I4 = I1 × I2 × I3 × I4
cache

Repeating same multiplications for each step
=> Wasteful calculations

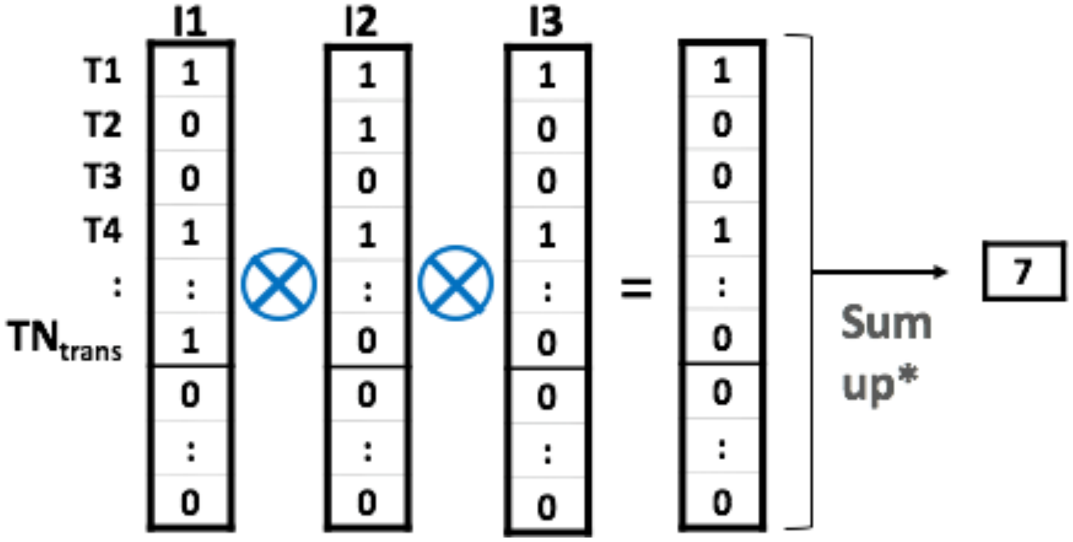
Only one time multiplication for each step
=> Execution time ⇩

Ciphertext-Packing/Caching techniques improve time and space complexities

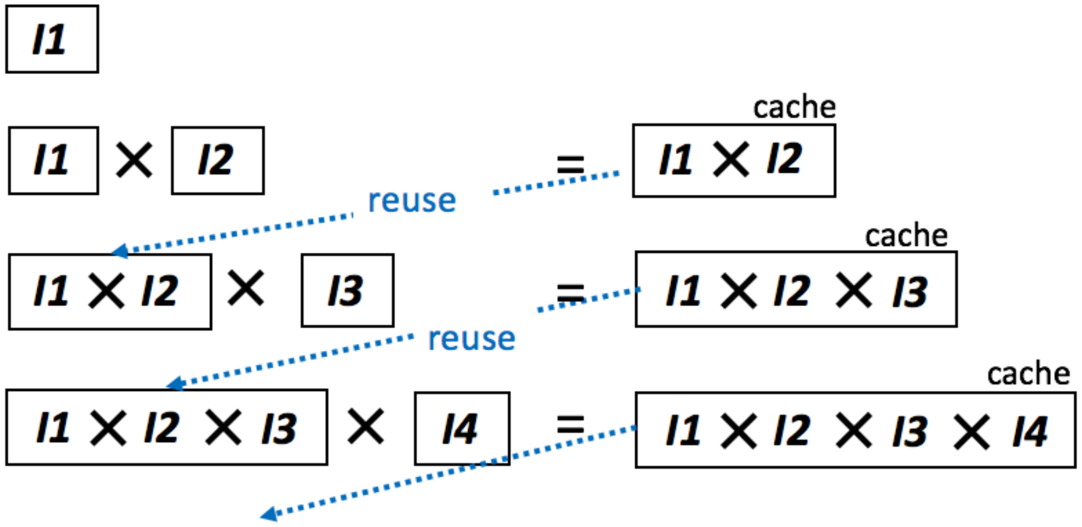
Problems to be improved:

- memory space ← (1)
- execution time ← (1), (2)

1) ciphertext-packing method



2) ciphertext-caching algorithm


I1
I1 × I2 = I1 × I2
cache
reuse
I1 × I2 × I3 = I1 × I2 × I3
cache
reuse
I1 × I2 × I3 × I4 = I1 × I2 × I3 × I4
cache

Experimental Setup

10GHz Ethernet

Client:
CPU: Intel Xeon CPU E5- 2643v3(3.4GHz)
memory: 512GB
(runs on 12-thread)

Server:
CPU: Intel Xeon CPU E7-8880 v3(2.3GHz)
memory: 1TB
(runs on 24-thread)

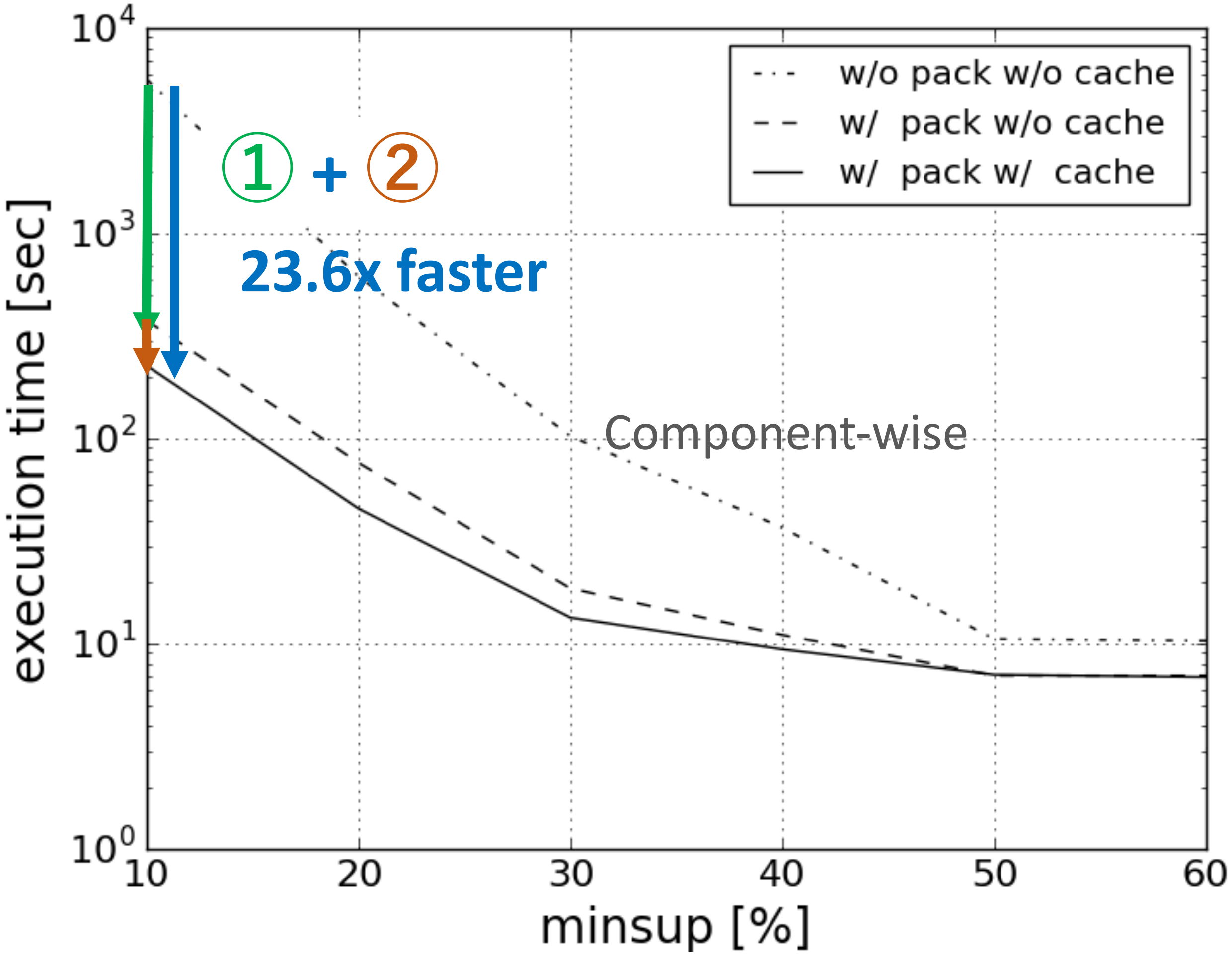
Dataset*:
- #Transaction: 100,
- #item ID: 50,
- Avg. #item in a transaction: 10

Library:
- HElib (FHE library)
- NTL mathmatical library
- GMP multiple-precision arithmetic library

（*Dataset was generated by IBM Quest Synthetic Data Generator ）

Evaluation Results - Scheme with Ciphertext-packing/caching techniques -

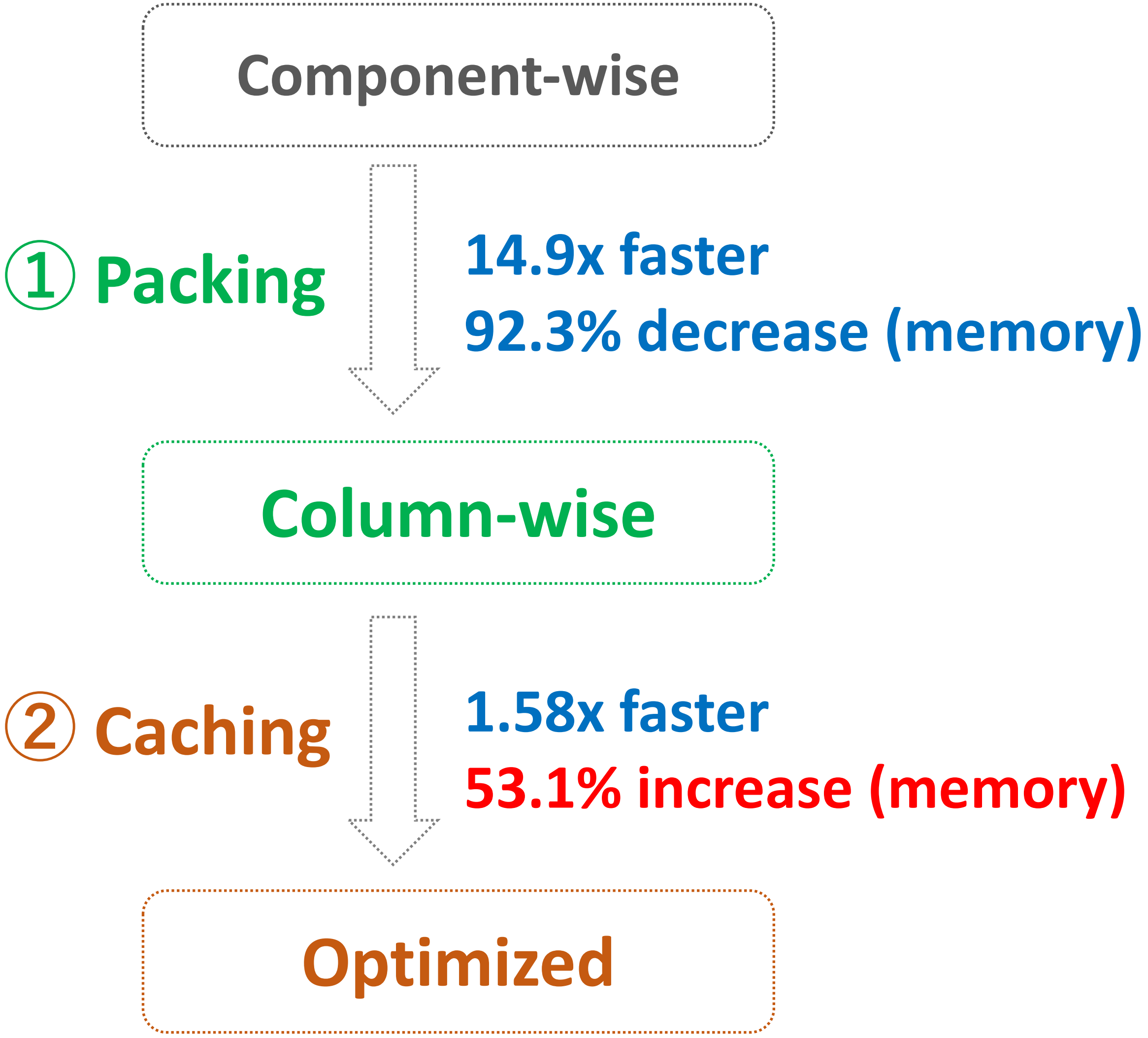
The scheme with packing & caching runs 23.6x faster than the scheme without them



① + ②

23.6x faster

Component-wise



Component-wise

① Packing

14.9x faster

92.3% decrease (memory)

Column-wise

② Caching

1.58x faster

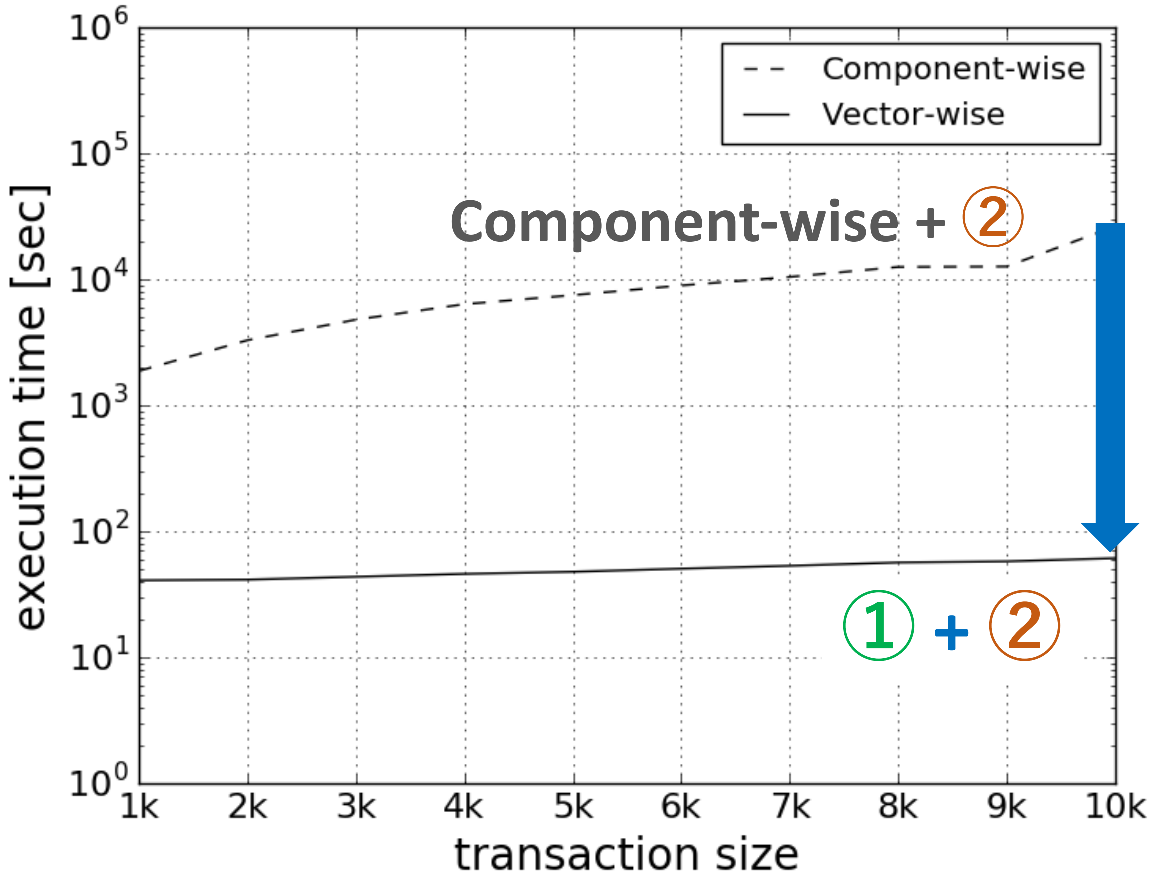
53.1% increase (memory)

Optimized

#trans = 100, #items = 50

Evaluation Results - Optimized scheme with security and relatively large data sizes -

Scheme with the ciphertext-packing/caching hardly depends on the transaction size



Varying num. of transactions

430x faster

94.7% decrease (memory)

① Packing

② Caching

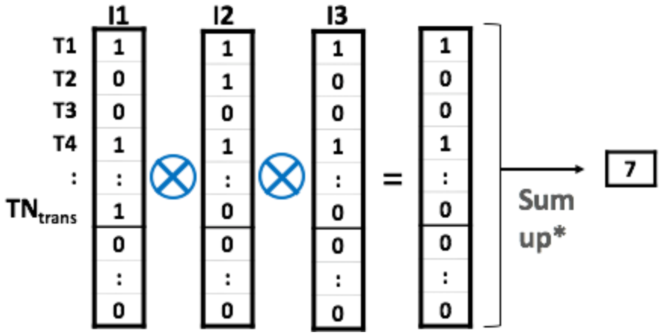
#items = 50

Conclusion - Apriori by FHE can be accelerated by Ciphertext-Packing/Caching technique -

Apriori (frequent pattern mining) by FHE

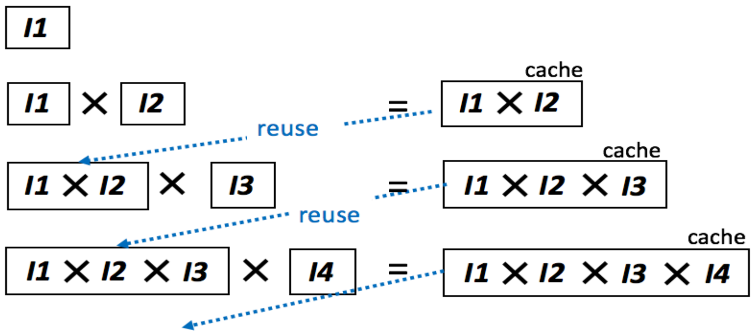
accelerate

Ciphertext-Packing



time and space complexities ⇩

Ciphertext-Caching



I1
I1 × I2 = I1 × I2
cache
reuse
I1 × I2 × I3 = I1 × I2 × I3
cache
reuse
I1 × I2 × I3 × I4 = I1 × I2 × I3 × I4
cache

time complexity ⇩

Problem remaining

The ciphertext-caching algorithm uses additional memory space

=> Needs to prune wasteful caches that is not reused later

Thank you for listening!

Any questions?